MOTOROLA

Freescale Semiconductor, Inc.

EC000 Core Processor

(SCM68000)

User's Manual

© MOTOROLA, 1995

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.

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.

Freescale Semiconductor, Inc.

— 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) 690-5593
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 *EC000 Core Processor User's Manual* describes the programming, capabilities, and operation of the SCM68000 (EC000 core); the *MC68000 Family Programmer's Reference Manual* provides instruction details for the EC000 core; and the *FlexCore Product Brief* provides a brief description of the FlexCore program.

The organization of this manual is as follows:

TRADEMARKS

- Composer, Verilog, Verifault, and Veritime are trademarks of Cadence Design Systems, Inc.
- Synopsys is a registered trademark of Synopsys, Inc.
- TDS is a registered trademark of Summit Design, Inc.

MOTOROLA   **EC000 CORE PROCESSOR USER'S MANUAL**   v
For More Information On This Product,
Go to: www.freescale.com

Freescale Semiconductor, Inc.

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

TABLE OF CONTENTS

Section 1
Overview

Section 2
Signal Description

Section 3
Bus Operation

Section 4
Exception Processing

viii   **EC000 CORE PROCESSOR USER'S MANUAL**   MOTOROLA
For More Information On This Product,
Go to: www.freescale.com

Section 5
8-Bit Instruction Execution Times

Section 6
16-Bit Instruction Execution Times

Freescale Semiconductor, Inc.

Section 7
Electrical Characteristics

LIST OF ILLUSTRATIONS

MOTOROLA   **EC000 CORE PROCESSOR USER'S MANUAL**   xi
For More Information On This Product,
Go to: www.freescale.com

Freescale Semiconductor, Inc.

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

LIST OF TABLES

MOTOROLA   EC000 CORE PROCESSOR USER'S MANUAL   xiii
For More Information On This Product,
Go to: www.freescale.com

Freescale Semiconductor, Inc.

List of Tables

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

SECTION 1
OVERVIEW

This document contains a summary of the use and operation of the SCM68000 micropro-
cessor core (also referred to as the EC000 core)[1] and a detailed set of timing and electrical
specifications. Refer to the *M68000 8-/16-/32-Bit Microprocessor User's Manual*
(M68000UM/AD) for detailed information on the operation of the instruction set, addressing
modes, and bus architecture for this core.

The SCM68000 is a core implementation of the MC68000 32-bit microprocessor and is
designed to be used as part of the FlexCore Program. In the FlexCore program, high-volume
manufacturers can create their own integrated microprocessor containing a core processor,
such as the SCM68000, and their own proprietary technology. A FlexCore integrated pro-
cessor allows significant reductions in component count, power consumption, board space,
and cost while yielding much higher system reliability and performance.

The main features of the SCM68000 include:

- Low-Power HCMOS Implementation Requires Only 15 mA at 3.3 V

- 32-Bit Performance for 16-Bit Applications—2.7 MIPS at 16 MHz

- Statically Selectable 8-Bit or 16-Bit Data Bus Operation

- 32-Bit Address Bus Directly Addresses up to 4 Gbytes of Address Space

- Static Operation Provides Almost Zero Power Consumption During Idle Periods

- Sixteen General-Purpose 32-Bit Data and Address Registers

- Fifty-Six Powerful Instruction Types That Support High-Level Programming Languages

- Fourteen Addressing Modes and Five Main Data Types Allow Compact, Efficient Code

- Seven Priority Level Interrupt Control

- Special Core Interfacing Signals

- Emulation Support Signals Including Pipeline Refill, Processor Status, and Interrupt
  Pending Signals

- Both 3.3-V and 5-V Operation

The SCM68000 has a statically selectable 8-bit or 16-bit data bus. The address bus is 32-
bits wide and may be used as either a 24-bit address bus as on the MC68000 microproces-
sors, or as a 32-bit address bus to fully support the internal architecture. The 32-bit address

---

[1.] The SCM68000 is the name of the Verilog model for the EC000 core. The remainder of this section
will refer to the EC000 core as only the SCM68000.

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

Freescale Semiconductor, Inc.

bus allows direct addressing of up to 4 Gbytes. Logic can be added to implement dynamic bus sizing.

The SCM68000 is upward code compatible with all other members of the M68000 microprocessor family. Any user-mode programs using the SCM68000 instruction set will run unchanged on any MC680x0, MC68EC0x0, or MC683xx processor. This is possible because the user programming model is identical for all processors and the instruction sets, addressing modes, and data types for the SCM68000 are proper subsets of the complete architecture.

The SCM68000 also includes some functions not found on the standard MC68000 and MC68EC000 microprocessors such as the processor status, pipeline refill, and interrupt pending signals. These signals permit emulation support and facilitate interfacing between the SCM68000 and on-chip logic.

1.1 FLEXCORE INTEGRATED PROCESSORS

FlexCore allows designers of high-volume digital systems and third-party technology providers to place their proprietary circuitry on chip with a Motorola microprocessor. By using Flex-Core, a designer can reduce the total system cost, component count, and power consumption while providing higher performance and greater reliability. Up to 100,000 gates or more of custom logic, memory, and peripheral modules can be added to a core processor to produce the most cost-effective solution for a designer's system. The core processors provide special power-management features such as 5 V, 3.3 V, and static operation. The 68000 Family of core processors offers the designer a range of performance from 3 to 12 million instructions per second (MIPS) (to be extended to 100 MIPS) while maintaining complete code compatibility throughout the Family. The 68000 processors have a proven architecture with a broad base of application and system software support, including real-time kernels, operating systems, and compilers, in addition to a wide range of tools to support software development. In the future, additional processing architectures will be included in the FlexCore program, including PowerPC™ and digital signal processing (DSP). Figure 1-1 shows a typical die layout for a FlexCore integrated processor.

Complete product lines can be created using FlexCore by implementing one base design using a variety of core processors. Designers already familiar with 68000 Family design can easily migrate to FlexCore processors as the core processors use the same bus interfaces found on the standard 68000 Family members. Additionally, many peripheral modules and memory elements are available for integration. Motorola has developed a complete design system to put into the hands of the customer that includes both a broad cell-based library and effective computer-aided design (CAD) tools. By building on Motorola's proven 68000 microprocessor architecture and superior manufacturing capabilities, FlexCore offers designers the best path to higher system integration.
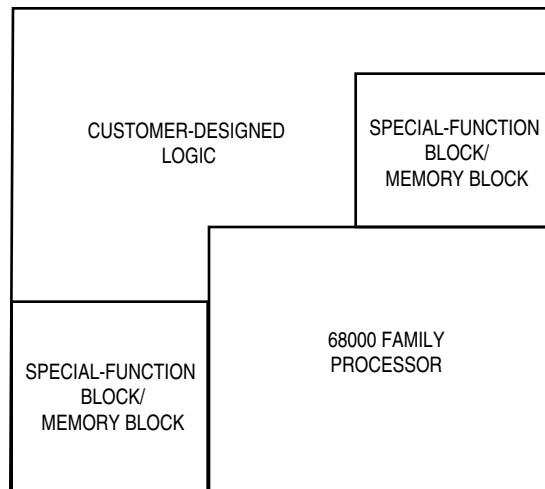
Figure 1-1. FlexCore Integrated Processor Typical Die Layout

FlexCore custom processors are ideal for:

- High-volume users of 8-, 16-, and 32-bit integrated solutions requiring higher system performance whose needs are not met by standard 68300 Family devices.

- Designers of high-volume applications who need to reduce cost, space, and/or power consumption.

- Third-party technology providers who want to deliver their proprietary application-specific technology to a worldwide marketplace.

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

To implement the application-specific logic, the designer uses Motorola's standard cell library. This library offers an extensive range of design elements, memory configurations, and an expanding array of peripheral modules. Each cell in the library has been designed for optimum size and performance. The added flexibility of high-speed, high-density cells allows the designer to achieve the most cost-effective solution while satisfying critical timing requirements. The standard cell library has been thoroughly characterized and maintained to ensure a smooth transition from a simulated design to working silicon. A custom part may also become a standard product if both Motorola and the customer desire to do so. Standard products are sold on the open market, allowing costs to be spread over additional units, resulting in lower component prices for high-volume users.

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

Freescale Semiconductor, Inc.

1.1.1 FlexCore Advantages

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

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

1.1.2 FlexCore Module Types

The three types of FlexCore modules are:

- Hard Module
  —Not alterable
  —Laid out
  —Has a tech file
  —Has a defined test scheme
- Soft Module
  —Netlist
  —Not alterable other than by clock tree insertion
  —Not laid out
  —Has a defined test scheme
  —Simulation test fixture
- Parameterizable
  —Alterable via insertion of predefined parameters
  —Behavioral model
  —Definition of parameters defines test scheme
  —Customer selects parameter values and Motorola synthesizes the design

The SCM68000 core processor is available as a hard module.

1.2 DEVELOPMENT CYCLE

There are several steps that must be followed in order to create a FlexCore integrated microprocessor with an SCM68000. Figure 1-2 illustrates the standard cell design flow and the tools required to complete each step. These steps include:

- Convert Design to Standard Cells Design—Begin by implementing the required system functions with an SCM68000, peripherals, memory blocks, and cells from the Motorola standard cell library.

- Capture Design on Workstation—Use the engineering workstation to capture the logic schematic of cells and their interconnections.

- Logic Synthesis—The structural level description of the design is mapped to a more efficient structural description, which is accomplished by converting the Boolean equations for the design to a two-level sum of products representation and minimized.

- Generate Test Patterns—The stimulus and test patterns for the design are generated for the functional simulation.

- Functional Simulation—Ensure that the logic of the schematic is functionally sound by using Verilog, the encrypted C models and synthesis models provided by Motorola. No timing information is yet associated with the simulations, and all propagation delays are preset to 1 ns.

- Calculate Node Delays— Motorola software (mdaDecal) calculates the estimated propagation delays of each node in the circuit. The design software estimates delays based on the fanout, drive characteristics, and estimated interconnect capacitances of the netlist and reveals potential timing problems.

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

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

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

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

- Interconnect Analysis—After the cells are placed and routed, the interconnect capacitances are extracted. These capacitances replace those estimated earlier during the calculation of the node delays.

- Re-Simulate—The circuit is re-simulated with Verilog to ensure no problems have arisen due to a change in load conditions. If changes have occurred or the simulation is different in any way, the test vectors must also be extracted again.

Freescale Semiconductor, Inc.

```
┌─────────────────┐   ┌─────────────────┐   ┌─────────────────────┐   ┌─────────────────────┐
│ SYNTHESIS       │   │ ENCRYPTED C     │   │ GENERATE TEST       │   │ CONVERT DESIGN TO   │
│ MODULES         │   │ MODULES         │   │ PATTERNS            │   │ STANDARD CELLS      │
│                 │   │                 │   │ (STL/SYNOPSYS)      │   │                     │
└─────────────────┘   └─────────────────┘   └─────────────────────┘   └─────────────────────┘
         │                    │                      │                          │
         ▼                    │                      │                          ▼
┌─────────────────┐           │                      │              ┌─────────────────────┐
│ LOGIC SYNTHESIS │           │                      │              │ CAPTURE DESIGN ON   │
│ (SYNOPSYS)      │           │                      │              │ WORKSTATION         │
│                 │           │                      │              │ (COMPOSER, VERILOG  │
└─────────────────┘           │                      │              │ HDL, VHDL)          │
         │                    │                      │              └─────────────────────┘
         │                    │                      │                          │
         └────────────┬───────┴──────────┬───────────┴──────────────┬───────────┘
                      ▼                   ▼                          ▼
              ┌──────────────────────────────────────┐
              │ FUNCTIONAL SIMULATION                 │──────────────────────────┐
              │ (VERILOG)                             │                          │
              └──────────────────────────────────────┘                          │
                             │                                                   │
                             ▼                                                   │
              ┌──────────────────────────────────────┐                          │
              │ CALCULATE NODE DELAYS                 │                          │
              │ (mdaDECAL)                            │                          │
              └──────────────────────────────────────┘                          │
                             │                                                   │
                             ▼                                                   ▼
              ┌──────────────────────────┐          ┌──────────────────────────┐
              │ PATH DELAY ANALYSIS       │          │ PERFORM FAULT GRADING     │
              │ (VERITIME)                │          │ (VERIFAULT)               │
              └──────────────────────────┘          └──────────────────────────┘
                             │                                    │
                             ▼                                    │
              ┌──────────────────────────┐                       │
              │ PERFORM REAL-TIME         │                       │
              │ SIMULATION                │                       │
              │ (VERILOG)                 │                       │
              └──────────────────────────┘                       │
                             │                                    │
                             ▼                                    │
              ┌──────────────────────────┐                       │
              │ EXTRACT TEST VECTORS      │◄──────────────────────┘
              │ (Summit Design)           │
              └──────────────────────────┘
                             │
                             ▼
              ┌──────────────────────────┐
              │ AUTOMATIC PLACE & ROUTE   │────────────────┐
              └──────────────────────────┘                │
                             │                             │
                             ▼                             │
              ┌──────────────────────────┐                │
              │ INTERCONNECT ANALYSIS     │                │
              │ (mdaDECAL)                │                │
              └──────────────────────────┘                │
                             │                             ▼
                             ▼              ┌──────────────────────────┐
              ┌──────────────────────────┐  │ NETLIST COMPARISON        │
              │ RE-SIMULATE               │  │ (LVS)                     │
              │ (VERILOG)                 │  └──────────────────────────┘
              └──────────────────────────┘                │
                             │                             │
                             ▼                             │
              ┌──────────────────────────┐                │
              │ PATTERN, MASK AND         │◄───────────────┘
              │ WAFER GENERATION          │
              └──────────────────────────┘
                             │
                             ▼
              ┌──────────────────────────┐
              │ ASSEMBLY / TEST           │
              └──────────────────────────┘
                             │
                             ▼
              ┌──────────────────────────┐   ┌──────────────────────────┐
              │ SHIP TESTED PROTOTYPES    │◄──│ FINAL TEST PROGRAM        │
              └──────────────────────────┘   └──────────────────────────┘

      ▨ MOTOROLA      □ CUSTOMER
```

Figure 1-2. Standard Cell Design Flow

EC000 CORE PROCESSOR USER'S MANUAL
For More Information On This Product,
Go to: www.freescale.com

1.3 PROGRAMMING MODEL

The SCM68000 programming model is illustrated in Figure 1-3. It is separated into two modes of access: user and supervisor. The user mode provides the execution environment for the majority of application programs. The supervisor mode, which allows some additional instructions and privileges, is used by the operating system and other system software. Detailed information about the programming model can be found in the *M68000 Family Programmer's Reference Manual* (M68000PM/AD).
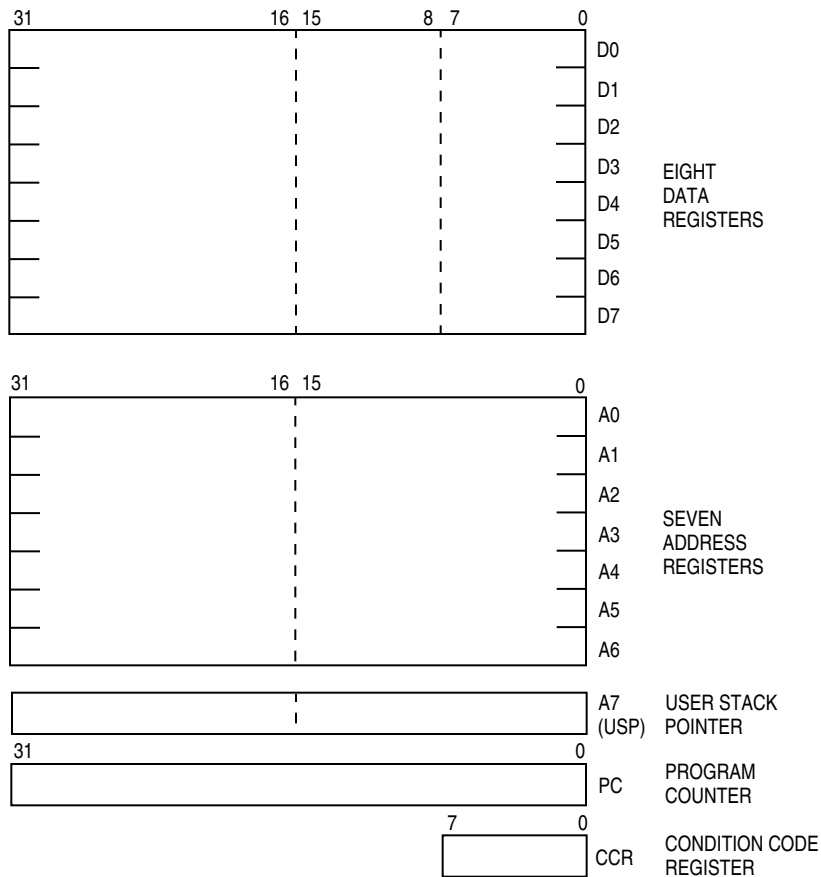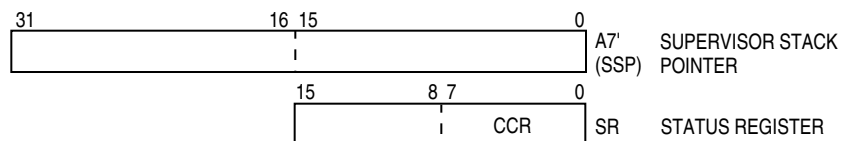
Figure 1-3. Programming Model

Freescale Semiconductor, Inc.

The user mode (see Figure 1-3(a)) provides access to 16 32-bit general-purpose registers (D0–D7, A0–A7), a 32-bit program counter, and an 8-bit condition code register. The first eight registers (D0–D7) are used as data registers for byte (8-bit), word (16-bit), and long-word (32-bit) operations. The second set of seven registers (A0–A6) and the user stack pointer (A7/USP) can be used as software stack pointers and base address registers. In addition, the address registers can be used for word and long-word operations. All of the 16 registers can be used as index registers.

The supervisor mode (see Figure 1-3(b)) provides access to two supplementary registers, the status register (high-order byte) and the supervisor stack pointer (A7'/SSP). The status register (SR) (see Figure 1-4) contains the interrupt mask (eight levels available) and the following condition codes: overflow (V), zero (Z), negative (N), carry (C), and extend (X). Additional status bits indicate whether the SCM68000 is in the trace (T) mode and/or in the supervisor (S) state. Bits 5, 6, 7, 11, 12, and 14 are undefined and reserved for future expansion.
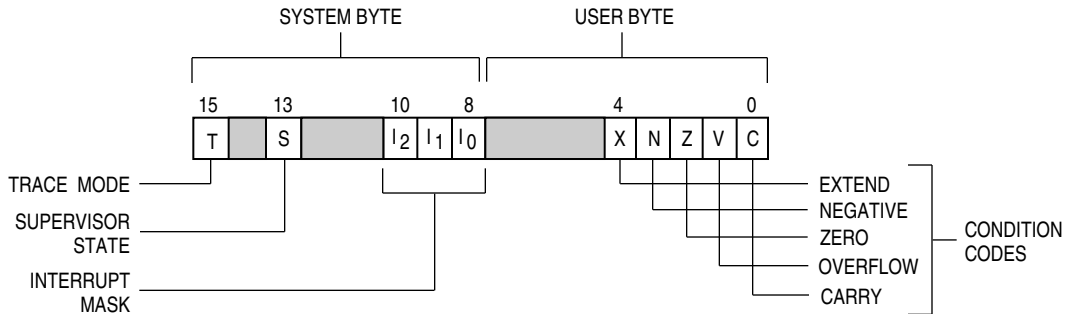
Figure 1-4. Status Register

EC000 CORE PROCESSOR USER'S MANUAL
For More Information On This Product,
Go to: www.freescale.com

1.4 DATA TYPES AND ADDRESSING MODES

Detailed information about the data types and addressing modes can be found in the *M68000 Family Programmer's Reference Manual* (M68000PM/AD). The SCM68000 supports the five basic data types of the M68000 family:

1. Bit

2. Binary-Coded-Decimal (BCD) Digit (4 Bits)

3. Byte (8 Bits)

4. Word (16 Bits)

5. Long Word (32 Bits)

In addition, the instruction set supports operations on other data formats such as memory addresses, status word, data, etc.

The SCM68000 also supports the basic addressing modes of the M68000 family. The register indirect addressing modes support postincrementing, predecrementing, offsetting, and indexing capabilities. The program counter relative mode also supports indexing and offsetting. Table 1-1 lists a summary of the data addressing modes for the SCM68000.

Table 1-1. Data Addressing Modes

Addressing Modes	Generation	Syntax

Register Direct Addressing
  Data Register Direct
  Address Register Direct
	EA = Dn
	EA = An
		Dn
		An

Absolute Data Addressing
  Absolute Short
  Absolute Long
	EA = (Next Word)
	EA = (Next Two Words)
		(xxx).W
		(xxx).L

Program Counter Relative Addressing
  Relative with Offset
  Relative with Index and Offset
	EA = (PC) + d_{16}
	EA = (PC) + d8
		(d_{16},PC)
		(d8,PC,Xn)

Register Indirect Addressing
  Register Indirect
  Postincrement Register Indirect
  Predecrement Register Indirect
  Register Indirect with Offset
  Indexed Register Indirect with Offset
	EA = (An)
	EA = (An), An ← An + N
	An ← An − N, EA = (An)
	EA = (An) + d_{16}
	EA = (An) + (Xn) + d8
		(An)
		(An)+
		−(An)
		(d_{16},An)
		(d8,An,Xn)

Immediate Data Addressing
  Immediate
  Quick Immediate
	DATA = Next Word(s)
	Inherent Data
		#<data>

Implied Addressing Implied Register	EA = SR, USP, SSP, PC	SR, USP, SSP, PC

NOTES:

EA	=	Effective Address
Dn	=	Data Register
An	=	Address Register
( )	=	Contents of
PC	=	Program Counter
d8	=	8-Bit Offset (Displacement)
d_{16}	=	16-Bit Offset (Displacement)
N	=	1 for byte, 2 for word, and 4 for long word. If An is the stack pointer and the operand size is byte, N = 2 to keep the stack pointer on a word boundary.
←	=	Replaces
Xn	=	Address or Data Register Used as Index Register
SR	=	Status Register
USP	=	User Stack Pointer
SSP	=	Supervisor Stack Pointer
(xxx)	=	Absolute Address

Freescale Semiconductor, Inc.

1.5 DATA ORGANIZATION

The eight data registers support data operands of 1, 8, 16, or 32 bits. The seven address registers and the active stack pointer support address operands of 32 bits.

1.5.1 Data Registers

Each data register is 32 bits wide. Byte operands occupy the low-order 8 bits, word operands, the low-order 16 bits, and long-word operands, the entire 32 bits. The least significant bit is addressed as bit zero; the most significant bit is addressed as bit 31.

When a data register is used as either a source or a destination operand, only the appropriate low-order portion is changed; the remaining high-order portion is neither used nor changed. For example, if 8 bits are to be moved into a data register, bits 0 through 7 will be modified and bits 8 through 31 will not be changed.

1.5.2 Address Registers

Each address register (and the stack pointer) is 32 bits wide and holds a full 32-bit address. Address registers do not support byte-sized operands. Therefore, when an address register is used as a source operand, either the low-order word or the entire long-word operand is used, depending upon the operation size. When an address register is used as the destination operand, the entire register is affected, regardless of the operation size. If the operation size is word, operands are sign-extended to 32 bits before the operation is performed.

1.5.3 Data Organization In Memory

Bytes are individually addressable. As shown in Figure 1-5, the high-order byte of a word has the same address as the word. The low-order byte has an odd address, one count higher. Instructions and multibyte data are accessed only on word (even byte) boundaries. If a long-word operand is located at address n (n even), then the second word of that operand is located at address n+2.
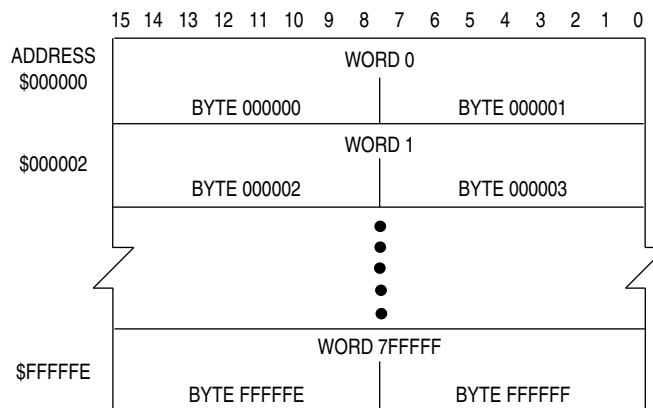
Figure 1-5. Word Organization in Memory

EC000 CORE PROCESSOR USER'S MANUAL

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

The data types supported by the SCM68000 are bit data, integer data of 8, 16, and 32 bits, 32-bit addresses, and binary-coded-decimal data. Each data type is stored in memory as shown in Figure 1-6.

1.6 INSTRUCTION SET SUMMARY

Table 1-2 lists the notational conventions used throughout this manual unless otherwise specified. Table 1-3 lists the SCM68000 instruction set by opcode. In the syntax descriptions, the left operand is the source operand, and the right operand is the destination operand.

Freescale Semiconductor, Inc.

BIT DATA:

1 BYTE = 8 BITS

7	6	5	4	3	2	1	0

INTEGER DATA:

1 BYTE = 8 BITS

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
n	**MSB**			BYTE 0				**LSB**				BYTE 1					n + 1
n + 2				BYTE 2								BYTE 3					n + 3

1 WORD = 16 BITS

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
n	**MSB**							WORD 0								**LSB**
n + 2								WORD 1								
n + 4								WORD 2								

1 LONG WORD = 32 BITS

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
n	**MSB**							HIGH ORDER								
n + 2							LONG WORD 0 LOW ORDER									**LSB**
n + 4																
n + 6							LONG WORD 1									
n + 8																
n + 10							LONG WORD 2									

ADDRESSES:

1 ADDRESS = 32 BITS

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
n	**MSB**							HIGH ORDER								
n + 2							ADDRESS 0 LOW ORDER									**LSB**
n + 4																
n + 6							ADDRESS 1									
n + 8																
n + 10							ADDRESS 2									

MSB = MOST SIGNIFICANT BIT
LSB = LEAST SIGNIFICANT BIT

DECIMAL DATA:

2 BINARY CODED DECIMAL DIGITS = 1 BYTE

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
MSD		BCD 0				BCD 1		**LSD**		BCD 2				BCD 3		
	BCD 4				BCD 5				BCD 6				BCD 7			

MSD = MOST SIGNIFICANT DIGIT
LSD = LEAST SIGNIFICANT DIGIT

Figure 1-6. Data Organization in Memory

EC000 CORE PROCESSOR USER'S MANUAL MOTOROLA
For More Information On This Product,
Go to: www.freescale.com

Freescale Semiconductor, Inc.

Freescale Semiconductor, Inc.

Table 1-2. Notational Conventions

Single- and Double-Operand Operations	
≠	Not equal.
+	Arithmetic addition or postincrement indicator.
−	Arithmetic subtraction or predecrement indicator.
×	Arithmetic multiplication.
÷	Arithmetic division or conjunction symbol.
~	Invert; operand is logically complemented.
Λ	Logical AND
V	Logical OR
⊕	Logical exclusive OR
→	Source operand is moved to destination operand.
↔	Two operands are exchanged.
<	Relational test; true if source operand is less than destination operand.
>	Relational test; true if source operand is greater than destination operand.
<operand>	Data used as an operand.
<operand> tested	Operand is compared to zero and the condition codes are set appropriately.
<operand> sign-extended <operand>	All bits of the upper portion are made equal to the high-order bit of the lower portion.
<operand> shifted by <count>	The source operand is shifted by the number of count.
<operand> rotated by <count>	The source operand is rotated by the number of count.
bit number of <operand>	Selects a single bit of the operand.
Other Operations	
TRAP	$1 \rightarrow$ S-bit of SR; SSP − 4 → SSP; PC → (SSP); SSP − 2 → SSP; SR → (SSP); Vector Address → PC
STOP	Enter the stopped state, waiting for interrupts.
<operand>$_{10}$	The operand is BCD; operations are performed in decimal.
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.
Register Specification	
An	Any Address Register n (example: A3 is address register 3)
Ax, Ay	Source and destination address registers, respectively.
Dn	Any Data Register n (example: D5 is data register 5)
Dx, Dy	Source and destination data registers, respectively.
Rn	Any Address or Data Register
Rx, Ry	Any source and destination registers, respectively.
Xn	Index Register—An, Dn, or suppressed.
Data Format and Type	
<fmt>	Operand Data Format: Byte (B), Word (W), Long (L)
Subfields and Qualifiers	
#<xxx> or #<data>	Immediate data following the instruction word(s).
( )	Identifies an indirect address in a register.
[ ]	Identifies an indirect address in memory.
d$_n$	Displacement Value, n Bits Wide (example: d$_{16}$ is a 16-bit displacement).
Register Names	
CCR	Condition Code Register (lower byte of status register)
PC	Program Counter
SR	Status Register

EC000 CORE PROCESSOR USER'S MANUAL

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

Freescale Semiconductor, Inc.

Table 1-2. Notational Conventions (Continued)

Register Codes	
C	Carry Bit in CCR
cc	Condition Codes from CCR
N	Negative Bit in CCR
U	Undefined, Reserved for Motorola Use
V	Overflow Bit in CCR
X	Extend Bit in CCR
Z	Zero Bit in CCR
Stack Pointers	
SP	Active Stack Pointer
SSP	Supervisor (Master or Interrupt) Stack Pointer
USP	User Stack Pointer
Miscellaneous	
Í	Effective Address
<label>	Assembly Program Label
<list>	List of registers, for example D3–D0.

Table 1-3. Instruction Set Summary

Opcode	Operation	Syntax
ABCD	$Source_{10}$ + $Destination_{10}$ + X $\rightarrow$ Destination	ABCD Dy,Dx ABCD –(Ay), –(Ax)
ADD	Source + Destination $\rightarrow$ Destination	ADD <ea>,Dn ADD Dn,<ea>
ADDA	Source + Destination $\rightarrow$ Destination	ADDA <ea>,An
ADDI	Immediate Data + Destination $\rightarrow$ Destination	ADDI # <data>,<ea>
ADDQ	Immediate Data + Destination $\rightarrow$ Destination	ADDQ # <data>,<ea>
ADDX	Source + Destination + X $\rightarrow$ Destination	ADDX Dy, Dx ADDX –(Ay), –(Ax)
AND	Source Λ Destination $\rightarrow$ Destination	AND <ea>,Dn AND Dn,<ea>
ANDI	Immediate Data Λ Destination $\rightarrow$ Destination	ANDI # <data>, <ea>
ANDI to CCR	Source Λ CCR $\rightarrow$ CCR	ANDI # <data>, CCR
ANDI to SR	If supervisor state then Source Λ SR $\rightarrow$ SR else TRAP to Privilege Violation Trap	ANDI # <data>, SR
ASL, ASR	Destination Shifted by <count> $\rightarrow$ Destination	ASd Dx,Dy ASd # <data>,Dy ASd <ea>
Bcc	If (condition true) then PC + d_n $\rightarrow$ PC	Bcc <label>
BCHG	~ (<bit number> of Destination) $\rightarrow$ Z; ~ (<bit number> of Destination) $\rightarrow$ <bit number> of Destination	BCHG Dn,<ea> BCHG # <data>,<ea>
BCLR	~ (<bit number> of Destination) $\rightarrow$ Z; 0 $\rightarrow$ <bit number> of Destination	BCLR Dn,<ea> BCLR # <data>,<ea>
BKPT	Run breakpoint acknowledge cycle; TRAP as illegal instruction	BKPT # <data>
BRA	PC + d_n $\rightarrow$ PC	BRA <label>
BSET	~ (<bit number> of Destination) $\rightarrow$ Z; 1 $\rightarrow$ <bit number> of Destination	BSET Dn,<ea> BSET # <data>,<ea>
BSR	SP – 4 $\rightarrow$ SP; PC $\rightarrow$ (SP); PC + d_n $\rightarrow$ PC	BSR <label>
BTST	~ (<bit number> of Destination) $\rightarrow$ Z;	BTST Dn,<ea> BTST # <data>,<ea>
CHK	If Dn < 0 or Dn > Source then TRAP to CHK Instruction Vector	CHK <ea>,Dn
CLR	0 $\rightarrow$ Destination	CLR <ea>
CMP	Destination – Source $\rightarrow$ cc	CMP <ea>,Dn

EC000 CORE PROCESSOR USER'S MANUAL

Freescale Semiconductor, Inc.

Table 1-3. Instruction Set Summary (Continued)

CMPA	Destination – Source → cc	CMPA <ea>,An
CMPI	Destination – Immediate Data → cc	CMPI # <data>,<ea>
CMPM	Destination – Source → cc	CMPM (Ay)+, (Ax)+
DBcc	If condition false then (Dn – 1 → Dn; If Dn ≠ –1 then PC + d_n → PC)	DBcc Dn,<label>
DIVS	Destination ÷ Source → Destination	DIVS.W <ea>,Dn32/16 → 16r:16q
DIVU	Destination ÷ Source → Destination	DIVU.W <ea>,Dn32/16 → 16r:16q
EOR	Source ⊕ Destination → Destination	EOR Dn,<ea>
EORI	Immediate Data ⊕ Destination → Destination	EORI # <data>,<ea>
EORI to CCR	Source ⊕ CCR → CCR	EORI # <data>,CCR
EORI to SR	If supervisor state then Source ⊕SR → SR else TRAP to Privilege Violation Trap	EORI # <data>,SR
EXG	Rx ↔ Ry	EXG Dx,Dy EXG Ax,Ay EXG Dx,Ay EXG Ay,Dx
EXT	Destination Sign-Extended → Destination	EXT.W Dnextend byte to word EXT.L Dnextend word to long word
ILLEGAL	SSP – 4 → SSP; PC → (SSP); SSP – 2 → SSP; SR → (SSP); Illegal Instruction Vector Address → PC	ILLEGAL
JMP	Destination Address → PC	JMP <ea>
JSR	SP – 4 → SP; PC → (SP) Destination Address → PC	JSR <ea>
LEA	<ea> → An	LEA <ea>,An
LINK	SP – 4 → SP; An → (SP) SP → An, SP + d_n → SP	LINK An, # <displacement>
LSL,LSR	Destination Shifted by <count> → Destination	LSd Dx,Dy LSd # <data>,Dy LSd Í
MOVE	Source → Destination	MOVE <ea>,<ea>
MOVEA	Source → Destination	MOVEA <ea>,An
MOVE to CCR	Source → CCR	MOVE <ea>,CCR
MOVE from SR	SR → Destination	MOVE SR,<ea>
MOVE to SR	If supervisor state then Source → SR else TRAP to Privilege Violation Trap	MOVE <ea>,SR
MOVE USP	If supervisor state then USP → An or An → USP else TRAP to Privilege Violation Trap	MOVE USP,An MOVE An,USP
MOVEM	Registers → Destination; Source → Registers	MOVEM <list>,<ea> MOVEM <ea>,<list>
MOVEP	Source → Destination	MOVEP Dx,(d_{16},Ay) MOVEP (d_{16},Ay),Dx
MOVEQ	Immediate Data → Destination	MOVEQ # <data>,Dn
MULS	Source × Destination → Destination	MULS.W <ea>,Dn16 x 16 → 32
MULU	Source × Destination → Destination	MULU.W <ea>,Dn16 x 16 → 32
NBCD	0 – (Destination$_{10}$) – X → Destination	NBCD <ea>
NEG	0 – (Destination) → Destination	NEG <ea>
NEGX	0 – (Destination) – X → Destination	NEGX <ea>
NOP	None	NOP
NOT	~Destination → Destination	NOT <ea>
OR	Source V Destination → Destination	OR <ea>,Dn OR Dn,<ea>

EC000 CORE PROCESSOR USER'S MANUAL

Freescale Semiconductor, Inc.

Table 1-3. Instruction Set Summary (Continued)

ORI	Immediate Data V Destination → Destination	ORI # <data>,<ea>
ORI to CCR	Source V CCR → CCR	ORI # <data>,CCR
ORI to SR	If supervisor state then Source V SR → SR else TRAP to Privilege Violation Trap	ORI # <data>,SR
PEA	Sp – 4 → SP; <ea> → (SP)	PEA <ea>
RESET	If supervisor state then Assert RESETOB Line else TRAP to Privilege Violation Trap	RESET
ROL, ROR	Destination Rotated by <count> → Destination	ROd Dx,Dy ROd # <data>,Dy ROd Í
ROXL, ROXR	Destination Rotated with X by <count> → Destination	ROXd Dx,Dy ROXd # <data>,Dy ROXd Í

RTE	If supervisor state
	      then (SP) → SR; SP + 2 → SP; (SP) → PC;
	      SP + 4 → SP;
	      restore state and deallocate stack according to (SP)
	else TRAP to Privilege Violation Trap
		RTE

RTR	(SP) → CCR; SP + 2 → SP; (SP) → PC; SP + 4 → SP	RTR
RTS	(SP) → PC; SP + 4 → SP	RTS
SBCD	Destination$_{10}$ – Source$_{10}$ – X → Destination	SBCD Dx,Dy SBCD –(Äx),–(Ay)
Scc	If condition true then 1s → Destination else 0s → Destination	Scc <ea>
STOP	If supervisor state then Immediate Data → SR; STOP else TRAP to Privilege Violation Trap	STOP # <data>
SUB	Destination – Source → Destination	SUB <ea>,Dn SUB Dn,<ea>
SUBA	Destination – Source → Destination	SUBA <ea>,An
SUBI	Destination – Immediate Data → Destination	SUBI # <data>,<ea>
SUBQ	Destination – Immediate Data → Destination	SUBQ # <data>,<ea>
SUBX	Destination – Source – X → Destination	SUBX Dx,Dy SUBX –(Äx),–(Ay)
SWAP	Register [31:16] ↔ Register [15:0]	SWAP Dn
TAS	Destination Tested → Condition Codes; 1 → bit 7 of Destination	TAS <ea>
TRAP	1 → S-bit of SR; SSP – 4 → SSP; PC → (SSP); SSP – 2 → SSP; SR → (SSP); Vector Address → PC	TRAP # <vector>
TRAPV	If V then TRAP to TRAPV Instruciton Vector	TRAPV
TST	Destination Tested → Condition Codes	TST <ea>
UNLK	An → SP; (SP) → An; SP + 4 → SP	UNLK An

NOTE: d is direction, L or R.

EC000 CORE PROCESSOR USER'S MANUAL

SECTION 2
SIGNAL DESCRIPTION

This section contains descriptions of the SCM68000 (EC000 core)[1] input and output signals. The input and output signals are shown in Figure 2-1. Table 2-1 lists the pins, signal names, type, and whether they are three-stateable. The following paragraphs provide brief descriptions of the signals and references (where applicable) to other paragraphs that contain more information about the signals.

NOTE

The terms *assertion* and *negation* are used extensively in this manual to avoid confusion when describing a mixture of "active-low" and "active-high" signals. The term *assert* or *assertion* is used to indicate that a signal is active or true, independently of whether that level is represented by a high or low voltage. The term *negate* or *negation* is used to indicate that a signal is inactive or false.

2.1 ADDRESS BUS (A31–A0)

This 32-bit, unidirectional, three-state bus is capable of addressing 4 Gbytes of address space. This bus provides the address for bus operation during all cycles except interrupt acknowledge cycles. During interrupt acknowledge cycles, address lines A1, A2, and A3 provide the level number of the interrupt being acknowledged, and address lines A31–A4 and A0 are driven to a logic high.

2.2 DATA BUS (D15–D0)

This 16-bit, bidirectional, three-state bus is the general-purpose data-path. The data bus transfers and accepts data in either word or byte length if the SCM68000 is operating in the 16-bit mode. If the SCM68000 is operating in the 8-bit mode, it drives the entire bus during writes, but only the lower eight bits (D7–D0) contain valid data. In the 8-bit mode, the SCM68000 ignores the data on data lines D15–D8 during read cycles. During an interrupt acknowledge cycle, the external device supplies the vector number on data lines D7–D0.

2.3 CLOCK (CLKI, CLKO)

The CLKI input is internally buffered for development of the internal clocks needed by the SCM68000. This clock signal is a constant-frequency square wave that requires no stretching or shaping. The clock signal must conform to minimum and maximum pulse-width times

---

[1.] The SCM68000 is the name of the Verilog model for the EC000 core. The remainder of this section will refer to the EC000 core as only the SCM68000.
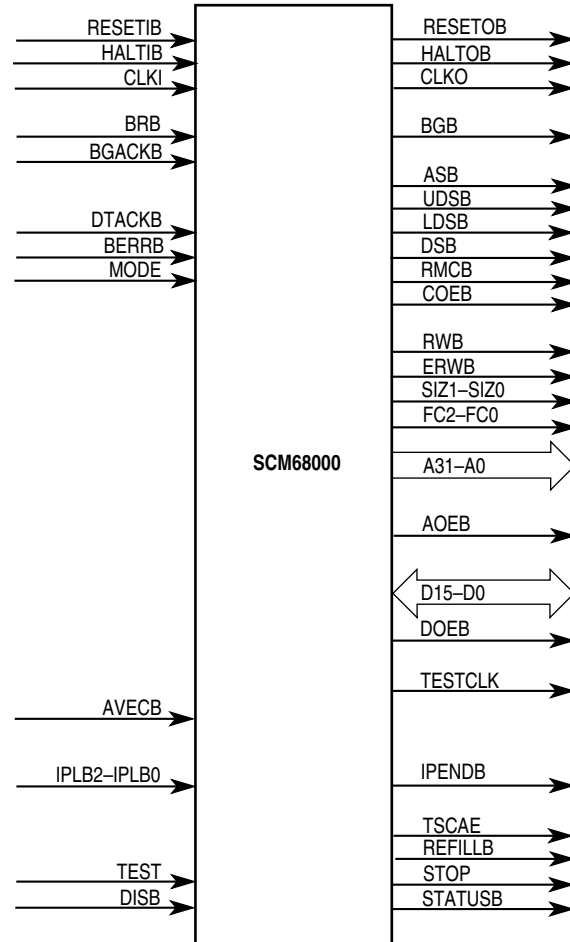
Figure 2-1. Input/Output Signals

Table 2-1. Signal Summary

Signal Name	Mnemonic	Input/ Output	Active State	Output Circuit	Hi-Z on HALTIB or STOP Instruction	Hi-Z on Bus Relinquish or RESETIB
Address Bus	A31–A0	Output	High	TS	No	Yes
Address Output Enable	AOEB	Output	Low	STD	No	No
Address Strobe	ASB	Output	Low	TS	No	Yes
Autovector	AVECB	Input	Low	STD	N/A	N/A
Bus Error	BERRB	Input	Low	STD	N/A	N/A
Bus Grant	BGB	Output	Low	STD	No	No
Bus Grant Acknowledge	BGACKB	Input	Low	STD	N/A	N/A
Bus Request	BRB	Input	Low	STD	N/A	N/A
Clock In	CLKI	Input	High	STD	N/A	N/A
Clock Out	CLKO	Output	High	STD	No	No
Control Output Enable	COEB	Output	Low	STD	No	No
Data Bus	D15–D0	Input/Output	High	TS	Yes	Yes
Disable Control	DISB	Input	Low	STD	N/A	N/A
Data Output Enable	DOEB	Output	Low	STD	No	No

Table 2-1. Signal Summary

Signal Name	Mnemonic	Input/ Output	Active State	Output Circuit	Hi-Z on HALTIB or STOP Instruction	Hi-Z on Bus Relinquish or RESETIB
Data Strobe	DSB	Output	Low	TS	No	Yes
Data Transfer Acknowledge	DTACKB	Input	Low	STD	N/A	N/A
Early Read Write	ERWB	Output	Low	TS	No	Yes
Function Code	FC2–FC0	Output	High	TS	No	Yes
Halt In	HALTIB	Input	Low	STD	N/A	N/A
Halt Out	HALTOB	Output	Low	STD	No	No
Interrupt Pending	IPENDB	Output	Low	STD	No	No
Interrupt Control	IPLB2–IPLB0	Input	Low	STD	N/A	N/A
Lower Data Strobe	LDSB	Output	Low	TS	No	Yes
Mode	MODE	Input	High	STD	N/A	N/A
CPU Pipe Refill	REFILLB	Output	Low	STD	No	No
Reset In	RESETIB	Input	Low	STD	N/A	N/A
Reset Out	RESETOB	Output	Low	STD	No	No
Read-Modify-Write	RMCB	Output	Low	TS	No	Yes
Read/Write	RWB	Output	Low	TS	No	Yes
Data Transfer Size	SIZ1, SIZ0	Output	High	TS	No	Yes
Microsequencer Status Indication	STATUSB	Output	Low	STD	No	No
Stop	STOP	Output	High	STD	No	No
Test	TEST	Input	High	STD	N/A	N/A
Test Clock	TESTCLK	Output	High	STD	No	No
Address Three-State Control	TSCAE	Output	High	STD	No	No
Upper Data Strobe	UDSB	Output	Low	TS	No	Yes

NOTE: TS = Three-State Output
STD = Standard CMOS Output

listed in the **Section 7 Electrical Characteristics**. The CLKO output follows CLKI to provide a reference for testing.

2.4 ASYNCHRONOUS BUS CONTROL

The following signals control asynchronous data transfers: address strobe, read/write, early read/write, data strobe, upper data strobe, lower data strobe, read-modify-write, data transfer size, and data transfer acknowledge. These signals are described in the following paragraphs.

2.4.1 Address Strobe (ASB)

This active low, three-state signal indicates that the information on the address bus is a valid address.

2.4.2 Read/Write (RWB) and Early Read/Write (ERWB)

The active-low, three-state RWB output signal defines the data bus transfer as a read or write cycle. The active-low, three-state ERWB output signal indicates a write cycle one-half clock cycle earlier than the read/write signal. Negation times are the same for both signals. These signals relate to the data strobe signals described in the following paragraphs.

2.4.3 Upper and Lower Data Strobes (UDSB, LDSB), and Data Strobe (DSB)

These active-low, three-state signals and RWB control the flow of data on the data bus. Table 2-2 lists the combinations of these signals and the corresponding data on the bus. When the RWB line is a logic high, the SCM68000 reads from the data bus. When the RWB line is a logic low, the SCM68000 writes to the data bus. In the case of an 8-bit write in 16-bit mode, the same data will be on both D7–D0 and D15–D8.

Table 2-2. Upper and Lower Data Strobe Control of Data Bus

UDSB	LDSB	RWB	D15–D8	D7–D0
High	High	—	No Valid Data	No Valid Data
Low	Low	High	Valid Data Bits 15–8	Valid Data Bits 7–0
High	Low	High	No Valid Data	Valid Data Bits 7–0
Low	High	High	Valid Data Bits 15–8	No Valid Data
Low	Low	Low	Valid Data Bits 15-8	Valid Data Bits 7–0
High	Low	Low	Valid Data Bits 7–0	Valid Data Bits 7–0
Low	High	Low	Valid Data Bits 15–8	Valid Data Bits 15–8

In 8-bit mode, UDSB is always forced high and only the LDSB signal and RWB are used to control the flow of data on the data bus. Table 2-3 lists the combinations of these signals and the corresponding data on the bus. When the RWB line is a logic high, the SCM68000 reads from the data bus. When the RWB line is a logic low, the SCM68000 drives the data bus.

Table 2-3. Lower Data Strobe Control of Data Bus

LDSB	RWB	Data Bus Operation
High	—	No Valid Data
Low	High	Read Cycle
Low	Low	Write Cycle

DSB is an active-low, three-state output signal that is asserted whenever LDSB or UDSB is asserted.

2.4.4 Data Transfer Acknowledge (DTACKB)

This active-low input indicates the completion of the data transfer. When the SCM68000 recognizes DTACKB during a read cycle, data is latched and the bus cycle is terminated. When DTACKB is recognized during a write cycle, the data bus enters a high-impedance state and the bus cycle is terminated.

2.4.5 Data Transfer Size (SIZ1–SIZ0)

These active-high, three-state output signals provide information on the size of the operand transfer. These outputs indicate the number of bytes to be transferred in the current bus cycle. Table 2-4 indicates the size signal encoding.

Table 2-4. Data Transfer Size

Size Code Output		
SIZ1	**SIZ0**	**Size**
1	0	Byte
0	1	Word

2.4.6 Read-Modify-Write (RMCB)

This active-low, three-state output line is logic low during read-modify-write cycles and indicates an indivisible bus sequence. This is described in **3.1.3 Read-Modify-Write Cycle**.

2.5 BUS ARBITRATION CONTROL

The bus request, bus grant, and bus grant acknowledge signals form the bus arbitration control signals that determine which device will be the bus master device. There are two possible arbitration protocols: 2-wire and 3-wire. In the 2-wire protocol, BGACKB is not used and must be negated.

2.5.1 Bus Request (BRB)

This active-low input is the combination of bus request signals from all other devices that could be bus masters. This signal indicates to the SCM68000 that some other device needs to become the bus master. Bus requests can be issued at any time during a cycle or between cycles.

2.5.2 Bus Grant (BGB)

This active-low output indicates to all other potential bus master devices that the SCM68000 will relinquish bus control at the end of the current bus cycle.

2.5.3 Bus Grant Acknowledge (BGACKB)—3-Wire Protocol Only

This active-low input indicates that some other device has become the bus master. This signal should not be asserted until the following four conditions are met:

1. A bus grant has been received.

2. Address strobe is negated, which indicates that the SCM68000 is not using the bus.

3. Data transfer acknowledge is negated, which indicates that neither memory nor peripherals are using the bus.

4. Bus grant acknowledge is negated, which indicates that no other device is still claiming to be the bus master.

2.6 INTERRUPT CONTROL (IPLB2–IPLB0)

These active-low input signals indicate the encoded priority level of the device requesting an interrupt. Level 7, which cannot be masked, has the highest priority; level 0 indicates that no interrupts have been requested. IPLB0 is the least significant bit of the encoded level, and IPLB2 is the most significant bit. For each interrupt request, these signals must maintain the interrupt request level until the SCM68000 acknowledges the interrupt to guarantee that

the interrupt is recognized. Table 2-5 lists the interrupt levels, the states of IPLB2–IPLB0 that define each level, and the mask value that allows an interrupt at each level.

Table 2-5. Interrupt Levels and Mask Values

Requested Interrupt Level	Control Line Status			Interrupt Mask Level Required for Recognition
	IPLB2	IPLB1	IPLB0	
0	High	High	High	No Interrupt Is Requested
1	High	High	Low	0
2	High	Low	High	1–0
3	High	Low	Low	2–0
4	Low	High	High	3–0
5	Low	High	Low	4–0
6	Low	Low	High	5–0
7	Low	Low	Low	7–0

2.7 SYSTEM CONTROL

The system control inputs are used to reset, halt, disable, and test the SCM68000 as well as signal a bus error to the SCM68000 and choose either the 8-bit or 16-bit mode. The two outputs reset the external devices in the system and signal to those devices when the SCM68000 has stopped executing instructions because of an error. The system control signals are described in the following paragraphs.

2.7.1 Bus Error (BERRB)

This input signal indicates a problem in the current bus cycle. The problem may be the following:

1. No response from a device.

2. No interrupt vector number returned.

3. An illegal access request rejected by a memory management unit.

4. Some other application-dependent error.

The SCM68000 either retries the bus cycle or performs exception processing, as determined by interaction between the bus error signal and the halt signal.

2.7.2 Reset External/Internal (RESETIB, RESETOB)

The assertion of the active-low input, RESETIB can start a system initialization sequence by resetting the SCM68000. The SCM68000 assertion of RESETOB (from executing a RESET instruction) resets all external devices of a system without affecting the internal state of the SCM68000. The interaction of RESETIB, RESETOB, and HALTIB is described in **4.3.1 Reset**.

2.7.3 Halt External/Internal (HALTIB, HALTOB)

Asserting the active-low input, HALTIB causes the SCM68000 to stop bus activity at the completion of the current bus cycle. This operation places all control signals in the inactive state and places the data bus in a high-impedance state (see Table 2-1).

When the SCM68000 has stopped executing instructions (in the case of a double bus fault condition, for example), the active-low output, HALTOB, is asserted by the SCM68000 to indicate the condition to external devices.

2.7.4 Mode (MODE)

This input selects between the 8-bit and 16-bit operating modes. If this input is grounded during reset, the SCM68000 comes out of reset in the 8-bit mode. If this input is tied to a logic high during reset, the SCM68000 comes out of reset in the 16-bit mode. Changing this input during normal operation may produce unpredictable results.

2.7.5 Disable Control (DISB)

This active-low signal is designed to place the SCM68000 into a quiescent state allowing other sections of the circuit to be tested without interference from the SCM68000. When this signal is asserted, the SCM68000 responds with the following with minimum gate delay if the clock is stopped:

- All three-state outputs will be placed into a high-impedance state.
- The bus grant (BGB), clock output (CLKO), halt output (HALTOB), reset output (RESE-TOB), microsequencer status (STATUSB), stop instruction indicator (STOP), and test clock (TESTCLK) signals remain at the state they were in when the clock was stopped.
- The remaining outputs are disabled, forcing them into an inactive state.

If the clock is running, the SCM68000 responds with the following with minimum gate delay:

- All three-state outputs will be placed into a high-impedance state.
- The clock output (CLKO) continues to follow the clock input (CLKI).
- The microsequencer status (STATUSB) signal is forced to a logic low.
- The test clock (TESTCLK) signal is forced to a logic low.
- The remaining outputs are disabled, forcing them into their inactive states.

When DISB is asserted, it is internally gated with the internal SCM68000 reset and halt signals after the input synchronizer. The user must ensure that the system is reset as discussed in **4.3.1 Reset**.

2.7.6 Test Mode (TEST)

This active-high input signal allows the SCM68000 to enter the test mode. This permits application of standard M68000 family test mode patterns to the SCM68000.

2.7.7 Test Clock (TESTCLK)

If the SCM680000 is properly reset during simulation, the TESTCLK signal will begin to pulse. A single period of the test clock consists of ten SCM68000 clock periods (six clocks low, four clocks high). This signal is generated by an internal ring counter that may come up in any state. (At power-on, it is impossible to guarantee phase relationship of TESTCLK to CLKI.) The TESTCLK signal is a free-running clock that runs regardless of the state of the MPU bus. For more information on resetting the SCM68000 for simulation, see **4.3.1.2 Initializing the SCM68000 for Simulation**.

2.7.8 Autovector (AVECB)

This active-low input signal indicates that the SCM68000 should use automatic vectoring for an interrupt during an interrupt acknowledge cycle. AVECB should be asserted only during an interrupt acknowledge cycle or erratic controller operation may occur.

2.8 THREE-STATE CONTROL

The following signals are the enable signals to put SCM68000 signals into a high-impedence state.

2.8.1 Address Output Enable (AOEB)

This active-low output signal is negated to put the address lines (A31–A0), function codes (FC2–FC0), size codes (SIZ1–SIZ0), early read/write (ERWB), and read/write (RWB) into a high-impedance state.

2.8.2 Control Output Enable (COEB)

This active-low output signal is negated to put the address strobe (ASB), data strobe (DSB), lower data strobe (LDSB), read-modify-write (RMCB), and upper data strobe (UDSB) into a high-impedance state.

2.8.3 Data Output Enable (DOEB)

This active-low output signal is negated to put the data lines of the SCM68000 into a high-impedance state.

2.9 PROCESSOR STATUS

These signals are used to indicate pending interrupts and when the SCM68000 is between bus cycles or at instruction boundaries. They also show when the instruction pipe is refilling and when the processor has been stopped. The signals are described in the following paragraphs.

2.9.1 Function Codes (FC2–FC0)

These active-high, three-state function code outputs indicate the mode (user or supervisor) and the address space currently being accessed as listed in Table 2-6. The function code outputs are valid whenever ASB is active.

Table 2-6. Function Code Outputs

Function Code Output			Cycle Time
FC2	FC1	FC0	
Low	Low	Low	(Undefined, Reserved)
Low	Low	High	User Data
Low	High	Low	User Program
Low	High	High	(Undefined, Reserved)
High	Low	Low	(Undefined, Reserved)
High	Low	High	Supervisor Data
High	High	Low	Supervisor Program
High	High	High	Interrupt Acknowledge

2.9.2 Address Three-State Control (TSCAE)

This active-high output signal is asserted between bus cycle accesses of the SCM68000.

2.9.3 Stop Instruction Indicator (STOP)

This output line pulses at one fourth the rate of the CLKI signal with an active time of one clock period when the STOP instruction is executed.

2.9.4 Interrupt Pending (IPENDB)

This active-low output signal indicates a valid interrupt has been recognized.

2.9.5 CPU Pipe Refill (REFILLB)

This active-low output signal is asserted for one clock period to indicate that a refill of the CPU pipe is occurring due to a change in program flow. This is used for emulator support.

2.9.6 Microsequencer Status Indication (STATUSB)

This active-low output signal indicates microsequencer status and is used for emulator support. The number of clock cycles for which this signal is asserted indicates the status of the SCM68000. When the SCM68000 approaches an instruction boundary, this signal is normally asserted for one clock cycle. Table 2-7 indicates exceptions that are indicated by the assertion of this signal for more than one cycle.

Table 2-7. Status Indication Exceptions

Asserted For	Indicates
One Clock	Sequencer at instruction boundary - will begin execution of next instruction
Two Clocks	Sequencer at instruction boundary - will not begin the next instruction immediately due to: • Pending Interrupt Exception or • Pending Trace Exception or • Illegal Instruction Exception or • Pending Breakpoint Instruction Exception or • Privileged Instruction Exception
Three Clocks	Exception processing to begin for: • Bus Error or • Address Error or • A-line Instruction or • Spurious Interrupt or • Illegal Instruction or • Privileged Instruction or • Auto vectored Interrupt or • F-line Instruction
Continuously	Core is: • Halted • Reset

2.10 MULTIPLEXING PINS

When a design is implemented, certain pins need to be multiplexed to the pads for testing purposes. Motorola recommends that all the pins on the SCM68000 be multiplexed to offer a means for testing the processor with test vectors provided by Motorola. This will provide maximum fault coverage. Varying degrees of fault coverage can be obtained depending on which pins the user does or does not multiplex.

Freescale Semiconductor, Inc.

All pins must be multiplexed according to the requirements in Table 2-8. However, it is necessary that the proper three-state control signal be used to control the three-state drivers as stated in **2.8 Three-State Control**. Table 2-8 shows a list of pins and the priority with which they need to be multiplexed. The priority column has three possible responses: required, required if used, not required, and internal. The "required if used" response means that the pin must be muxed out if the pin is used in the current design. The "internal" response means that the signal may be used internally and must not be muxed out.

Table 2-8. Pin Multiplexing Priority

Signal Name	Pin Name	Input/Output	Priority
Address Bus	A31–A24	Output	Required If Used
Address Bus	A23–A0	Output	Required
Address Output Enable	AOEB	Output	Internal
Address Strobe	ASB	Output	Required
Bus Error	BERRB	Input	Required
Bus Grant	BGB	Output	Required
Bus Grant Acknowledge	BGACKB	Input	Required
Bus Request	BRB	Input	Required
Clock In	CLKI	Input	Required
Clock Out	CLKO	Output	Required If Used
Control Output Enable	COEB	Output	Internal
Data Bus	D15–D0	Input/Output	Required
Disable Control	DISB	Input	Internal
Data Output Enable	DOEB	Output	Internal
Data Strobe	DSB	Output	Required If Used
Data Transfer Acknowledge	DTACKB	Input	Required
Test Clock	TESTCLK	Output	Not Required
Early Read Write	ERWB	Output	Required If Used
Function Code	FC2–FC0	Output	Required
Halt In*	HALTIB	Input	Required
Halt Out*	HALTOB	Output	Required
Interrupt Pending	IPENDB	Output	Required If Used
Interrupt Control	IPLB2–IPLB0	Input	Required
Lower Data Strobe	LDSB	Output	Required
Mode	MODE	Input	Required
CPU Pipe Refill	REFILLB	Output	Required If Used
Reset In*	RESETIB	Input	Required
Reset Out*	RESETOB	Output	Required
Read-Modify-Write	RMCB	Output	Required If Used
Read/Write	RWB	Output	Required
Data Transfer Size	SIZ1, SIZ0	Output	Required If Used
Microsequencer Status Indication	STATUSB	Output	Required If Used
Stop	STOP	Output	Required If Used
Test	TEST	Input	Required
Address Three-State Control	TSCAE	Output	Required If Used
Upper Data Strobe	UDSB	Output	Required
Autovector	AVECB	Input	Required

* HALTIB and HALTOB may share a single pin, HALTB, that is functionally equivalent to the circuit in Figure 4-10.
  RESETIB and RESETOB may share a single pin, RESETB, that is functionally equivalent to the circuit in Figure 4-10.

DISB must be negated by the pin multiplexing circuitry. If HALTOB and/or RESETOB are multiplexed to a three-state output, the internal pin multiplexing circuitry must assert the appropriate output enable.

EC000 CORE PROCESSOR USER'S MANUAL

Freescale Semiconductor, Inc.

SECTION 3
BUS OPERATION

This section describes control signals and bus operation during data transfer operations, bus arbitration, and bus error and halt conditions.

NOTE

The terms *assertion* and *negation* are used extensively in this manual to avoid confusion when describing a mixture of "active-low" and "active-high" signals. The term *assert* or *assertion* is used to indicate that a signal is active or true, independently of whether that level is represented by a high or low voltage. The term *negate* or *negation* is used to indicate that a signal is inactive or false.

3.1 DATA TRANSFER OPERATIONS

Transfer of data between devices involves the following signals:

1. Address bus (A31–A0)

2. Data bus (D7–D0 and/or D15–D8)

3. Control signals

The address and data buses are separate parallel buses used to transfer data using an asynchronous bus protocol. Control signals indicate the beginning and type of a bus cycle as well as the address space and size of the transfer. The selected device then controls the length of the cycle by terminating it using the control signals. In all bus cycles, the bus master assumes responsibility for de-skewing the acknowledge and data signals from the slave device.

The SCM68000 (EC000 core)[1] operates in either of two modes: 8-bit or 16-bit mode. The 8-bit mode is selected by grounding the MODE pin while the 16-bit mode is selected by pulling the MODE pin to a logic high (see **2.7.4 Mode (MODE)** for more information on the MODE signal).

During operation in the 8-bit mode, all bus cycles use LDSB, and one byte of data is transferred on data bus bits D7 through D0. UDSB is never asserted, and data bus bits D15 through D8 are undefined. For word or long-word operations, data is transferred in two and four bus cycles, respectively.

[1.] The SCM68000 is the name of the Verilog model for the EC000 core. The remainder of this section will refer to the EC000 core as only the SCM68000.

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

During operation in the 16-bit mode, byte operations can occur on either D15–D8 or D7–D0, depending on A0. If A0 is zero, the upper byte is used and UDSB is asserted. If A0 is one, the lower byte is used and LDSB is asserted. For word and long-word operations, A0 is always zero, data bits D15 through D0 are used, and both LDSB and UDSB are asserted. For long-word operations, data is transferred in two bus cycles with A1 indicating which half of the long word is being transferred. The actual order of the long-word halves is instruction and address-mode dependent.

The following paragraphs describe the read cycle, write cycle, read-modify-write cycle, and CPU space cycle. The indivisible read-modify-write cycle allows interlocked multiprocessor communications. A CPU space cycle is a special cycle used for interrupt acknowledge cycles.

3.1.1 Read Cycle

During a read cycle, the SCM68000 receives data from memory or from a peripheral device. When data is received, the SCM68000 correctly positions the byte internally.

The word read cycle flowchart is shown in Figure 3-1. The byte read cycle flowcharts for the 8-bit and 16-bit modes are shown in Figure 3-2 and Figure 3-3, respectively. The read cycle and write cycle timing diagrams are shown in Figure 3-4 and Figure 3-5. The word and byte read cycle timing diagram for operation in the 16-bit mode is shown in Figure 3-6.

BUS MASTER   SLAVE

ADDRESS THE DEVICE

1) SET RWB AND ERWB TO READ
2) PLACE FUNCTION CODE ON FC2–FC0
3) PLACE ADDRESS ON A31–A0
4) ASSERT ADDRESS STROBE (ASB)
5) ASSERT UPPER DATA STROBE
   (UDSB), LOWER DATA STROBE (LDSB),
   AND DATA STROBE (DSB)

OUTPUT THE DATA

1) DECODE ADDRESS
2) PLACE DATA ON D15–D0
3) ASSERT DATA TRANSFER
   ACKNOWLEDGE (DTACKB)

ACQUIRE THE DATA

1) LATCH DATA
2) NEGATE UDSB, LDSB, AND DSB
3) NEGATE ASB

TERMINATE THE CYCLE

1) REMOVE DATA FROM D15–D0
2) NEGATE DTACKB

START NEXT CYCLE

Figure 3-1. Word Read Cycle Flowchart for 16-Bit Mode

BUS MASTER   SLAVE

ADDRESS THE DEVICE

1) SET RWB AND ERWB TO READ
2) PLACE FUNCTION CODE ON FC2–FC0
3) PLACE ADDRESS ON A31–A0
4) ASSERT ADDRESS STROBE (ASB)
5) ASSERT LOWER DATA STROBE (LDSB)
   AND DATA STROBE (DSB)

OUTPUT THE DATA

1) DECODE ADDRESS
2) PLACE DATA ON D7–D0
3) ASSERT DATA TRANSFER
   ACKNOWLEDGE (DTACKB)

ACQUIRE THE DATA

1) LATCH DATA
2) NEGATE LDSB AND DSB
3) NEGATE ASB

TERMINATE THE CYCLE

1) REMOVE DATA FROM D7–D0
2) NEGATE DTACKB

START NEXT CYCLE

Figure 3-2. Byte Read Cycle Flowchart for 8-Bit Mode

BUS MASTER   SLAVE

ADDRESS THE DEVICE

1) SET RWB AND ERWB TO READ
2) PLACE FUNCTION CODE ON FC2–FC0
3) PLACE ADDRESS ON A31-A0
4) ASSERT ADDRESS STROBE (ASB)
5) ASSERT UPPER DATA STROBE (UDSB)
   OR LOWER DATA STROBE (LDSB)
   (BASED ON A0), AND DATA STROBE
   (DSB)

OUTPUT THE DATA

1) DECODE ADDRESS
2) PLACE DATA ON D7–D0 OR D15–D8
   (BASED ON UDSB OR LDSB)
3) ASSERT DATA TRANSFER
   ACKNOWLEDGE (DTACKB)

ACQUIRE THE DATA

1) LATCH DATA
2) NEGATE UDSB, LDSB, AND DSB
3) NEGATE ASB

TERMINATE THE CYCLE

1) REMOVE DATA FROM D7–D0
   OR D15–D8
2) NEGATE DTACKB

START NEXT CYCLE

Figure 3-3. Byte Read Cycle Flowchart for 16-Bit Mode

Freescale Semiconductor, Inc.

Figure 3-4. Read and Write Cycle Timing Diagram for 8-Bit Mode

EC000 CORE PROCESSOR USER'S MANUAL

For More Information On This Product,
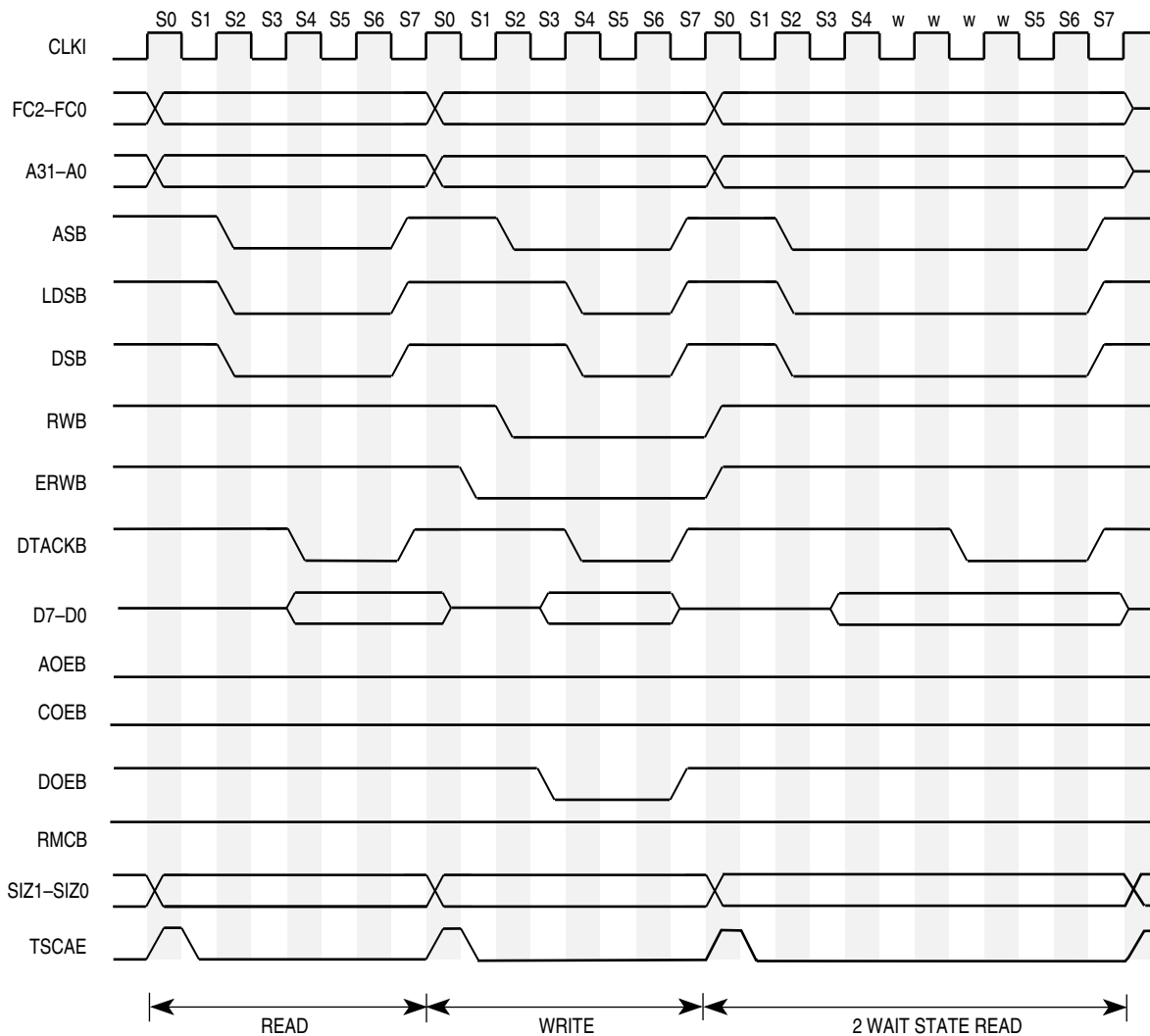**Go to: www.freescale.com**
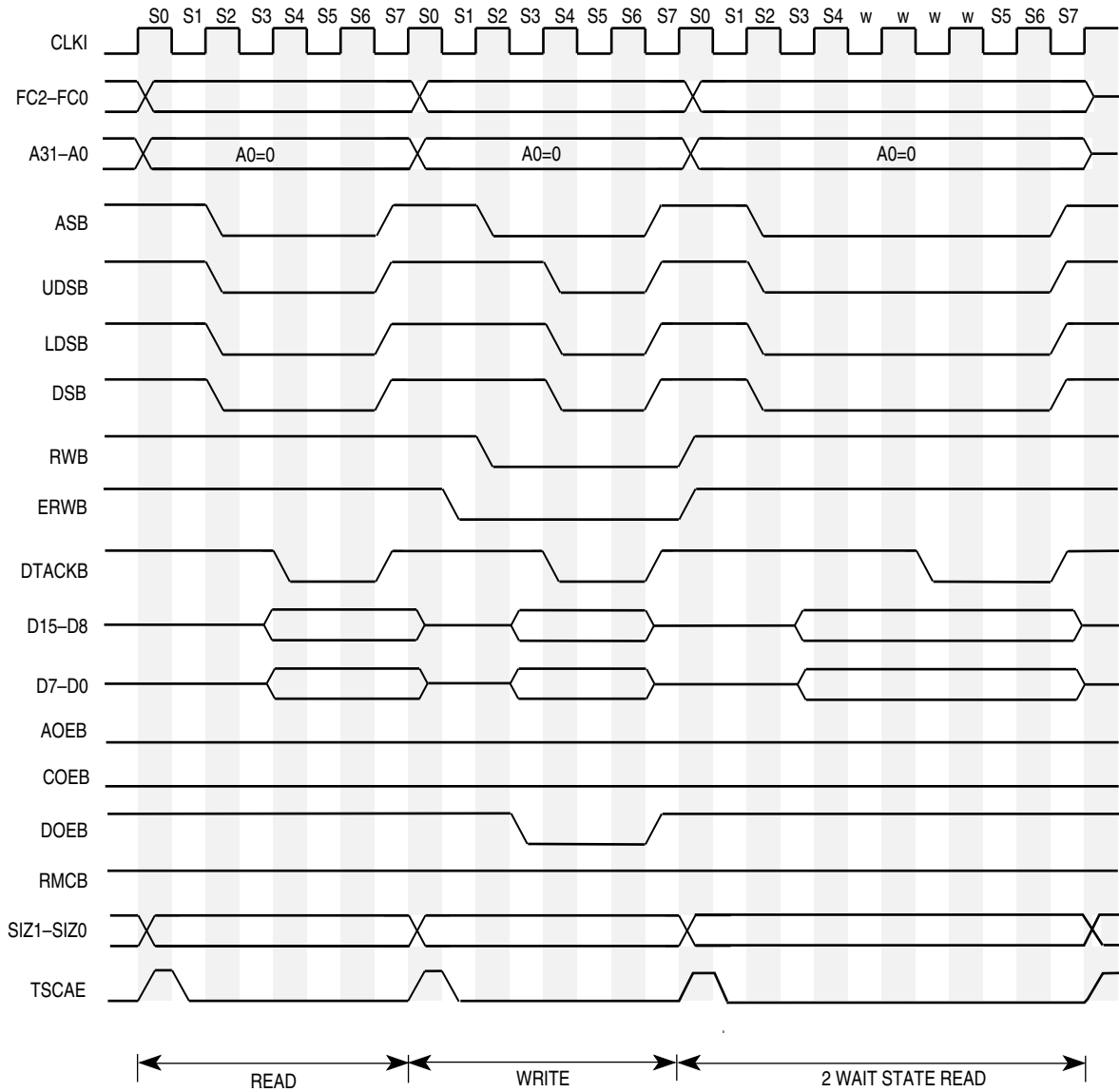
Figure 3-5. Read and Write Cycle Timing Diagram for 16-Bit Mode

Freescale Semiconductor, Inc.
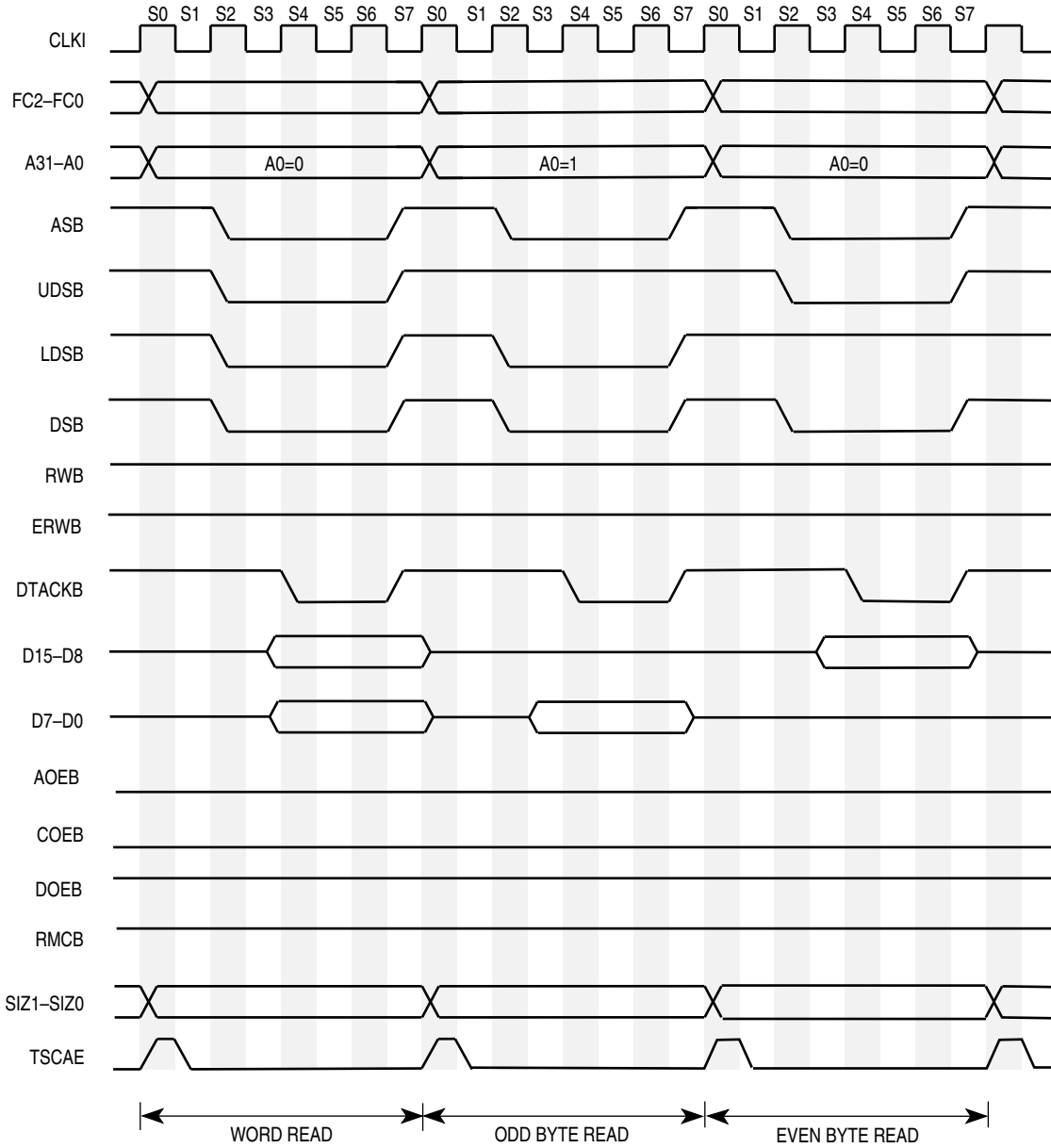
Figure 3-6. Word and Byte Read Cycle Timing Diagram for 16-Bit Mode

A bus cycle has a minimum of eight states. The various signals are asserted during specific states of a read cycle, as follows:

STATE 0

The read cycle starts in state 0 (S0). The SCM68000 places valid function codes on FC2–FC0, and a valid address on the address bus. RWB and ERWB are driven to logic highs to identify a read cycle, and TSCAE is driven to a logic high to indicate that the SCM68000 is between bus cycles.

EC000 CORE PROCESSOR USER'S MANUAL
For More Information On This Product,
Go to: www.freescale.com

STATE 1

Entering state 1 (S1), TSCAE is driven low to indicate the beginning of the bus cycle.

STATE 2

On the rising edge of state 2 (S2), the SCM68000 asserts ASB, UDSB, LDSB, and DSB.

STATE 3

During state 3 (S3), no bus signals are altered.

STATE 4

During state 4 (S4), the SCM68000 waits for a cycle termination signal (DTACKB or BERRB) If neither termination signal is asserted before the falling edge at the end of S4, the SCM68000 inserts wait states (full clock cycles) until either DTACKB or BERRB is asserted. See **3.7 The Relationship of DTACKB, BERRB, and HALTIB** for a description of how DTACKB and BERRB interact.

CASE 1: DTACKB is received alone or with BERRB (see **3.4 Bus Error and Halt Operation**).

STATE 5

During state 5 (S5), no bus signals are altered.

STATE 6

Sometime between state 2 (S2) and state 6 (S6), data from the device is driven onto the data bus.

STATE 7

On the falling edge of the clock entering state 7 (S7), the SCM68000 latches data from the addressed device and negates ASB, UDSB, LDSB, and DSB. The device negates DTACKB or BERRB at this time.

CASE 2: BERRB is received without DTACKB (see **3.4 Bus Error and Halt Operation**).

STATE 5

During state 5 (S5), no bus signals are altered.

STATE 6

During state 6 (S6), no bus signals are altered.

STATE 7

During state 7 (S7), no bus signals are altered.

STATE 8

During state 8 (S8), no bus signals are altered.

STATE 9

> During state 9 (S9), ASB, UDSB, LDSB, and DSB are negated. The device negates BERRB at this time.

3.1.2 Write Cycle

During a write cycle, the SCM68000 sends data to the memory or to a peripheral device.

The word write cycle flowchart is shown in Figure 3-7. The byte write cycle flowcharts for the 8-bit and 16-bit modes are shown in Figure 3-8 and Figure 3-9, respectively. The byte write cycle timing diagram for the 8-bit mode of operation is shown in Figure 3-10. The word and byte write cycle for the 16-bit mode of operation is shown in Figure 3-11.

BUS MASTER   SLAVE

ADDRESS THE DEVICE

1) PLACE FUNCTION CODE ON FC2–FC0
2) PLACE ADDRESS ON A31–A0
3) ASSERT ADDRESS STROBE (ASB)
4) SET RWB AND ERWB TO WRITE
5) PLACE DATA ON D15–D0
6) ASSERT UPPER DATA STROBE
   (UDSB), LOWER DATA STROBE (LDSB),
   AND DATA STROBE (DSB)

INPUT THE DATA

1) DECODE ADDRESS
2) STORE DATA ON D15–D0
3) ASSERT DATA TRANSFER
   ACKNOWLEDGE (DTACKB)

TERMINATE OUTPUT TRANSFER

1) NEGATE UDSB, LDSB, AND DSB
2) NEGATE ASB
3) REMOVE DATA FROM D15–D0
4) SET RWB AND ERWB TO READ

TERMINATE THE CYCLE

1) NEGATE DTACKB

START NEXT CYCLE

Figure 3-7. Word Write Cycle Flowchart for 16-Bit Mode

BUS MASTER

SLAVE

```
┌─────────────────────────────────────┐
│         ADDRESS THE DEVICE           │
├─────────────────────────────────────┤
│ 1) PLACE FUNCTION CODE ON FC2–FC0    │
│ 2) PLACE ADDRESS ON A31–A0           │
│ 3) ASSERT ADDRESS STROBE (ASB)       │
│ 4) SET RWB AND ERWB TO WRITE         │
│ 5) PLACE DATA ON D7–D0               │
│ 6) ASSERT LOWER DATA STROBE (LDSB)   │
│    AND DATA STROBE (DSB)             │
└─────────────────────────────────────┘
```

```
┌─────────────────────────────────────┐
│          INPUT THE DATA              │
├─────────────────────────────────────┤
│ 1) DECODE ADDRESS                    │
│ 2) STORE DATA ON D7–D0               │
│ 3) ASSERT DATA TRANSFER              │
│    ACKNOWLEDGE (DTACKB)              │
└─────────────────────────────────────┘
```

```
┌─────────────────────────────────────┐
│      TERMINATE OUTPUT TRANSFER       │
├─────────────────────────────────────┤
│ 1) NEGATE  LDSB AND DSB              │
│ 2) NEGATE ASB                        │
│ 3) REMOVE DATA FROM D7–D0            │
│ 4) SET RWB AND ERWB TO READ          │
└─────────────────────────────────────┘
```

```
┌─────────────────────────────────────┐
│         TERMINATE THE CYCLE          │
├─────────────────────────────────────┤
│ 1) NEGATE DTACKB                     │
│                                      │
└─────────────────────────────────────┘
```

```
┌─────────────────────────────────────┐
│          START NEXT CYCLE            │
└─────────────────────────────────────┘
```

Figure 3-8. Byte Write Cycle Flowchart for 8-Bit Mode

BUS MASTER

SLAVE

```
┌─────────────────────────────────────┐
│         ADDRESS THE DEVICE           │
├─────────────────────────────────────┤
│ 1) PLACE FUNCTION CODE ON FC2–FC0    │
│ 2) PLACE ADDRESS ON A31–A0           │
│ 3) ASSERT ADDRESS STROBE (ASB)       │
│ 4) SET RWB AND ERWB TO WRITE         │
│ 5) PLACE DATA ON D7–D0 OR D15–D8     │
│    (ACCORDING TO  A0)                │
│ 6) ASSERT UPPER DATA STROBE (UDSB)   │
│    OR LOWER DATA STROBE (LDSB) AND   │
│    DATA STROBE (DSB) (BASED ON A0)   │
└─────────────────────────────────────┘
```

```
┌───────────────────────────────────────┐
│           INPUT THE DATA               │
├───────────────────────────────────────┤
│ 1) DECODE ADDRESS                      │
│ 2) STORE DATA ON D7–D0 IF LDSB IS      │
│    ASSERTED.  STORE DATA ON D15–D8     │
│    IF UDSB IS ASSERTED                 │
│ 3) ASSERT DATA TRANSFER                │
│    ACKNOWLEDGE (DTACKB)                │
└───────────────────────────────────────┘
```

```
┌─────────────────────────────────────┐
│      TERMINATE OUTPUT TRANSFER       │
├─────────────────────────────────────┤
│ 1) NEGATE UDSB, LDSB, AND DSB        │
│ 2) NEGATE ASB                        │
│ 3) REMOVE DATA FROM D7–D0 OR         │
│    D15–D8                            │
│ 4) SET RWB AND ERWB TO READ          │
└─────────────────────────────────────┘
```

```
┌─────────────────────────────────────┐
│         TERMINATE THE CYCLE          │
├─────────────────────────────────────┤
│ 1) NEGATE DTACKB                     │
└─────────────────────────────────────┘
```

```
┌─────────────────────────────────────┐
│          START NEXT CYCLE            │
└─────────────────────────────────────┘
```

Figure 3-9. Byte Write Cycle Flowchart for 16-Bit Mode

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

Freescale Semiconductor, Inc.
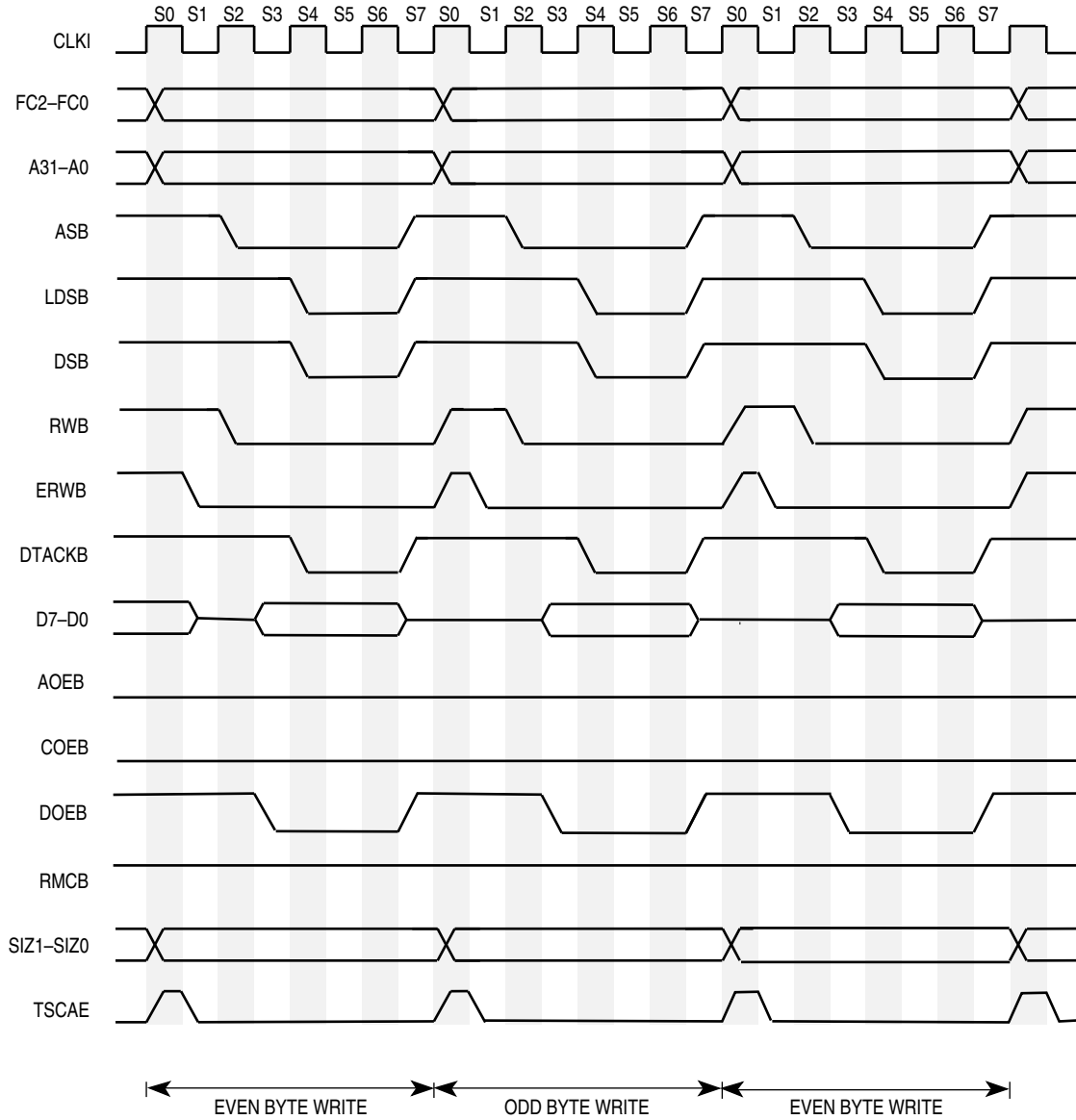
Figure 3-10. Write Cycle Timing Diagram for 8-Bit Mode

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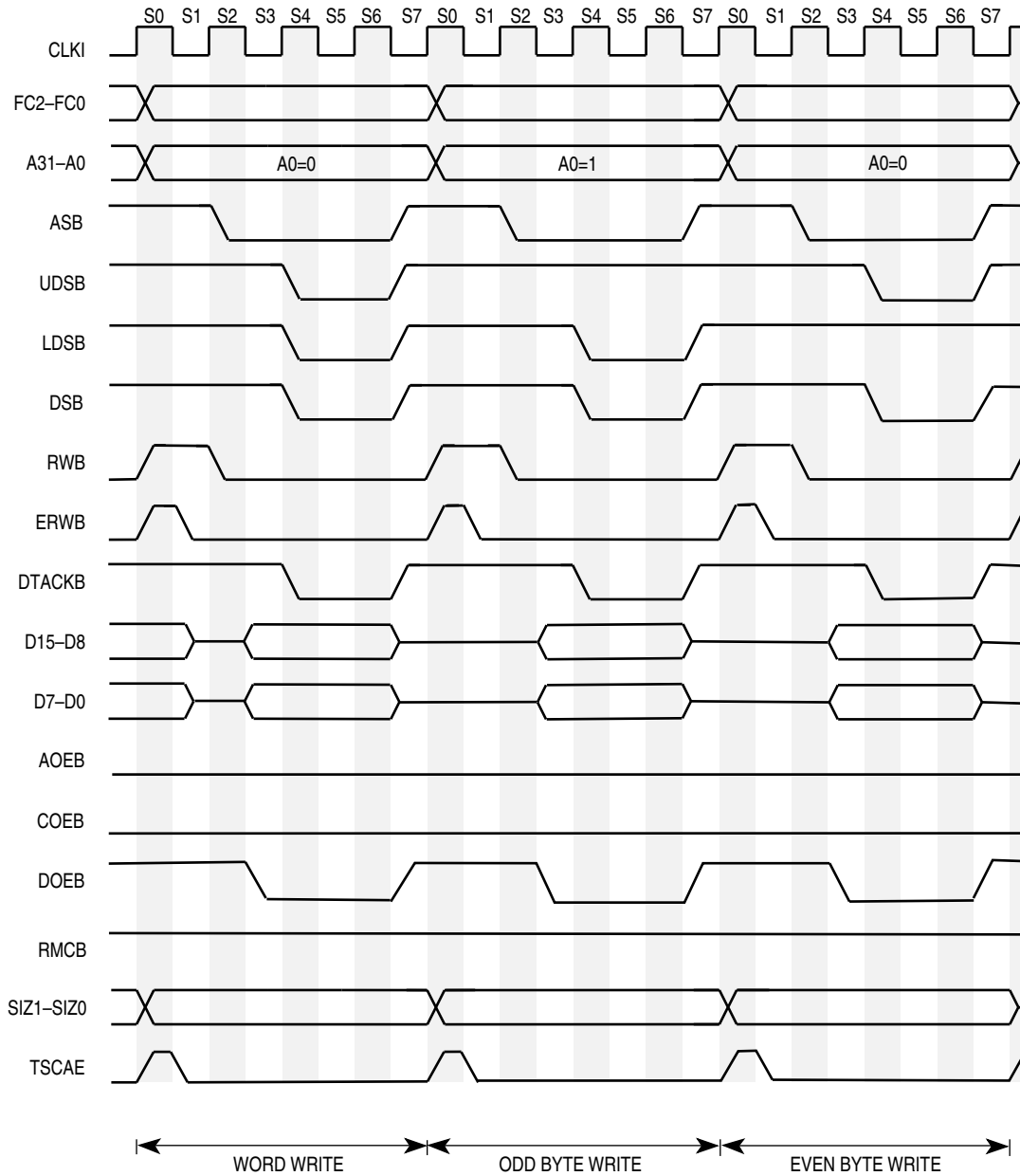**Go to: www.freescale.com**

Figure 3-11. Word and Byte Write Cycle Timing Diagram for 16-Bit Mode

The descriptions of the eight states of a write cycle are as follows:

STATE 0

    The write cycle starts in state 0 (S0). The SCM68000 places valid function codes on FC2–FC0 and a valid address on the address bus. RWB and ERWB are driven to a logic high. TSCAE is driven to a logic high to indicate that the SCM68000 is between bus cycles.

STATE 1

    Entering state 1 (S1), the SCM68000 drives TSCAE and ERWB to logic lows.

STATE 2

On the rising edge of state 2 (S2), the SCM68000 asserts ASB and drives RWB to a logic low.

STATE 3

During state 3 (S3), the data bus is driven out of the high-impedance state as data is placed on the bus.

STATE 4

At the rising edge of state 4 (S4), the SCM68000 asserts DSB, and UDSB and/or LDSB. The SCM68000 waits for a cycle termination signal (DTACKB or BERRB). If neither termination signal is asserted before the falling edge at the end of S4, the SCM68000 inserts wait states (full clock cycles) until either DTACKB or BERRB is asserted. See **3.7 The Relationship of DTACKB, BERRB, and HALTIB** for a description of how DTACKB and BERRB interact.

CASE 1: DTACKB is received alone or with BERRB (see **3.4 Bus Error and Halt Operation**).

STATE 5

During state 5 (S5), no bus signals are altered.

STATE 6

During state 6 (S6), no bus signals are altered.

STATE 7

On the falling edge of the clock entering state 7 (S7), the SCM68000 negates ASB, UDSB, LDSB, and DSB. As the clock rises at the end of S7, the SCM68000 places the data bus in the high-impedance state and drives RWB and ERWB to a logic high. The device negates DTACKB or BERRB at this time.

CASE 2: BERRB is received without DTACKB (see **3.4 Bus Error and Halt Operation**).

STATE 5

During state 5 (S5), no bus signals are altered.

STATE 6

During state 6 (S6), no bus signals are altered.

STATE 7

During state 7 (S7), no bus signals are altered.

STATE 8

During state 8 (S8), no bus signals are altered.

STATE 9

During state 9 (S9), ASB, UDSB, LDSB, and DSB are negated. The device negates BERRB at this time. At the end of S9, the data bus is placed in the high-impedance state, and RWB and ERWB are driven to a logic high.

3.1.3 Read-Modify-Write Cycle

The read-modify-write cycle performs a read operation, modifies the data in the arithmetic logic unit, and writes the data back to the same address. The address strobe (ASB) remains asserted throughout the entire cycle, making the cycle indivisible. The test and set (TAS) instruction uses this cycle to provide a signaling capability without deadlock between processors in a multiprocessing environment. The TAS instruction (the only instruction that uses the read-modify-write cycle) only operates on bytes. Thus, all read-modify-write cycles are byte operations. The read-modify-write flowchart is shown in Figure 3-12 and the timing diagram is shown in Figure 3-13.
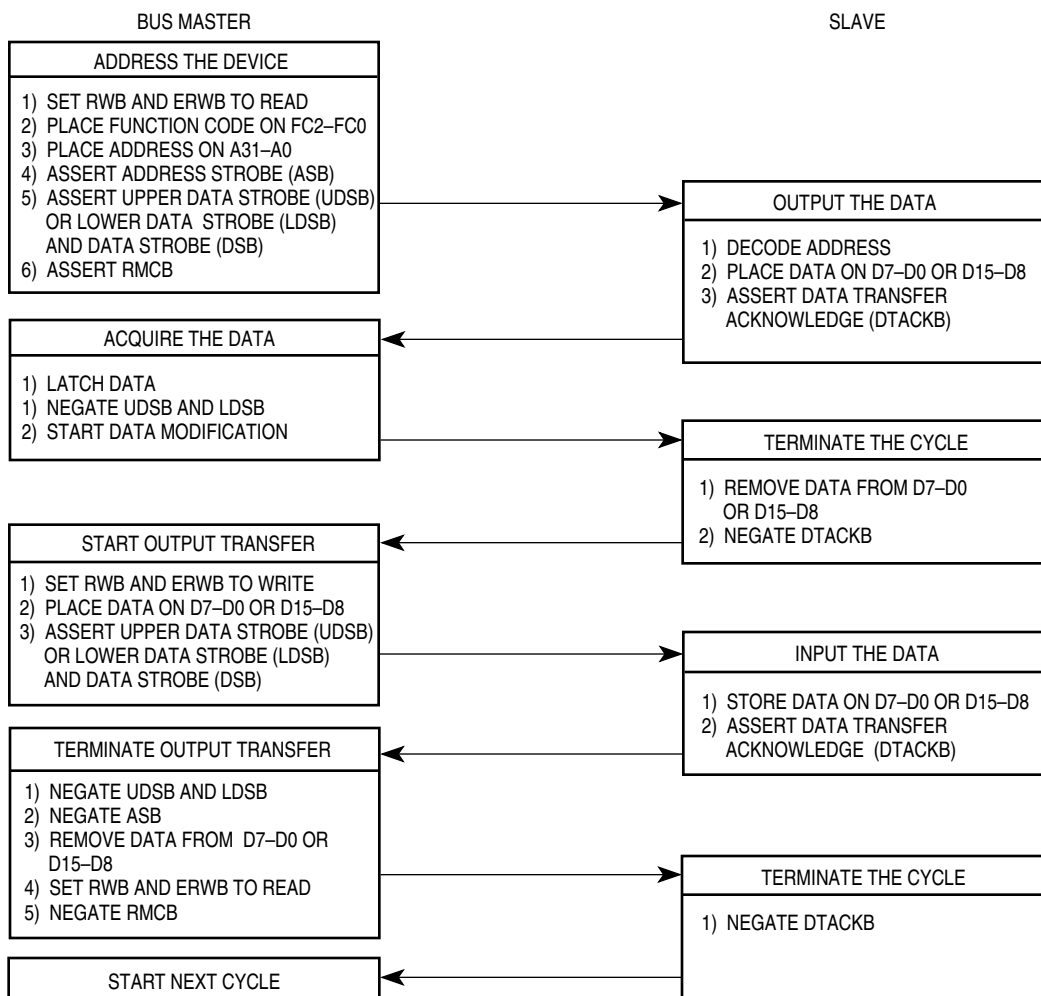
Figure 3-12. Read-Modify-Write Cycle Flowchart
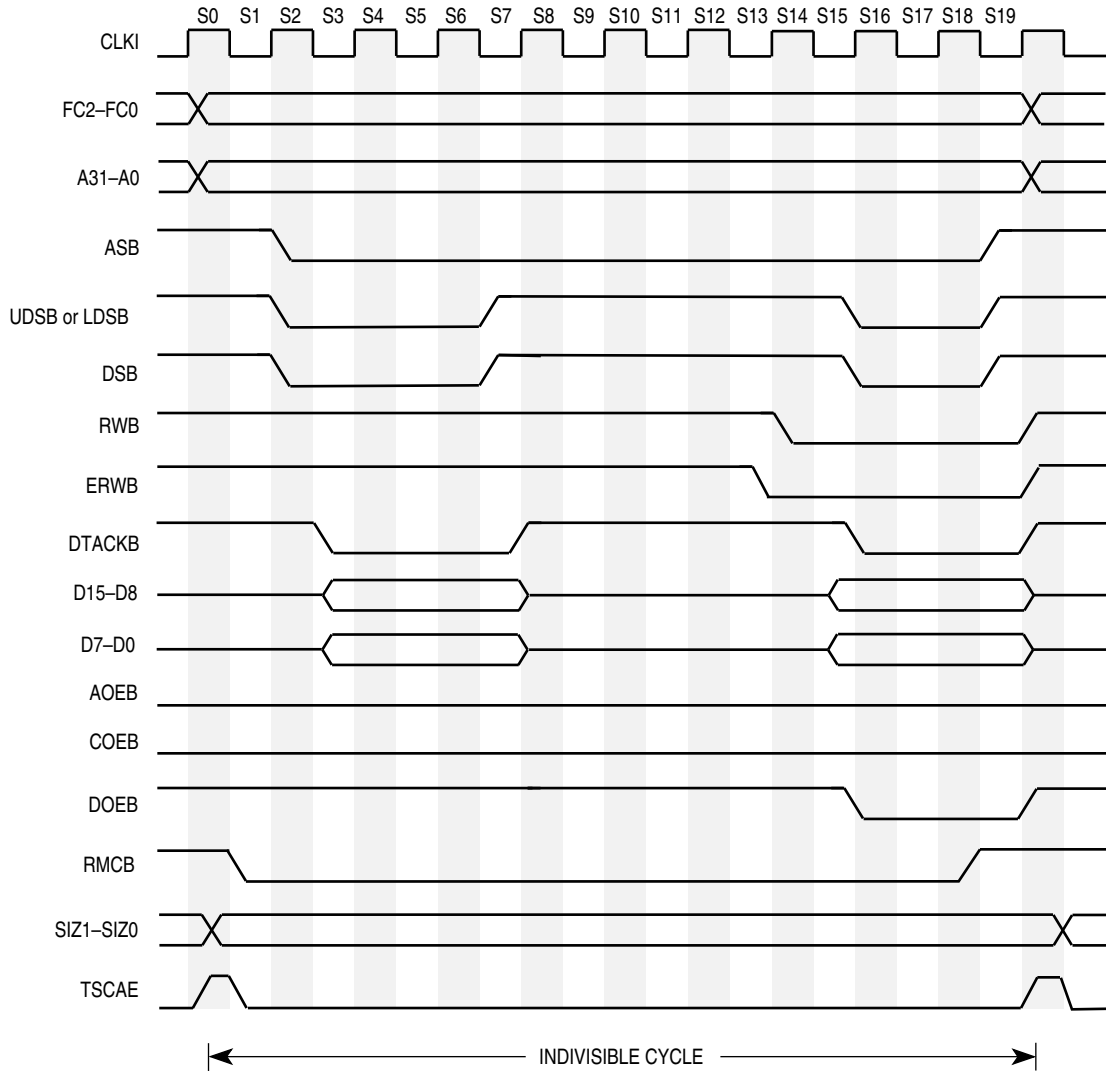
Figure 3-13. Read-Modify-Write Cycle Timing Diagram

The descriptions of the read-modify-write cycle states are as follows:

STATE 0

The read cycle starts in state 0 (S0). The SCM68000 places valid function codes on FC2–FC0 and a valid address on the address bus. RWB and ERWB are driven to a logic high to identify a read cycle, and TSCAE is driven to a logic high to indicate that the SCM68000 is between bus cycles.

STATE 1

Entering state 1 (S1), TSCAE is driven low to indicate the beginning of the bus cycle, and RMCB is driven low to indicate a read-modify-write cycle.

STATE 2

On the rising edge of state 2 (S2), the SCM68000 asserts ASB, UDSB or LDSB, and DSB.

STATE 3

During state 3 (S3), no bus signals are altered.

STATE 4

During state 4 (S4), the SCM68000 waits for a cycle termination signal (DTACKB or BERRB). If neither termination signal is asserted before the falling edge at the end of S4, the SCM68000 inserts wait states (full clock cycles) until either DTACKB or BERRB is asserted. See **3.7 The Relationship of DTACKB, BERRB, and HALTIB** for a description of how DTACKB and BERRB interact.

CASE READ 1: Only DTACK is received.

STATE 5

During state 5 (S5), no bus signals are altered.

STATE 6

During state 6 (S6), data from the device is driven onto the data bus.

STATE 7

On the falling edge of the clock entering state 7 (S7), the SCM68000 accepts data from the device and negates UDSB or LDSB, and DSB. The device negates DTACKB at this time.

STATES 8–11

The bus signals are unaltered during state 8 (S8)through state 11 (S11), during which the arithmetic logic unit makes appropriate modifications to the data.

STATE 12

The write portion of the cycle starts in state 12 (S12). The valid function codes on FC2–FC0, the address bus lines, ASB, RWB, and ERWB remain unaltered.

STATE 13

During state 13 (S13), ERWB is driven to a logic low.

STATE 14

On the rising edge of state 14 (S14), the SCM68000 drives RWB to a logic low.

STATE 15

During state 15 (S15), the data bus is driven out of the high-impedance state as data is placed on the bus.

STATE 16

During state 16 (S16), the SCM68000 waits for a cycle termination signal (DTACKB or BERRB). If neither termination signal is asserted before the falling edge at the end of S16,

the SCM68000 inserts wait states (full clock cycles) until either DTACKB or BERRB is asserted. Also, on the rising edge of S16, the SCM68000 asserts UDSB or LDSB, and DSB.

CASE WRITE 1: DTACKB is received alone or with BERRB (see **3.4 Bus Error and Halt Operation**).

STATE 17

During state 17 (S17), no bus signals are altered.

STATE 18

During state 18 (S18), no bus signals are altered.

STATE 19

On the falling edge of the clock entering state 19 (S19), the SCM68000 negates ASB, UDSB or LDSB, and DSB. As the clock rises at the end of S19, the SCM68000 places the data bus in the high-impedance state and drives RWB and ERWB to a logic high. The device negates DTACKB or BERRB at this time.

CASE READ 2: DTACKB and BERRB are received (see **3.4 Bus Error and Halt Operation**).

STATE 5

During state 5 (S5), no bus signals are altered.

STATE 6

During state 6 (S6), no bus signals are altered and data from the device is ignored.

STATE 7

During state 7 (S7), UDSB or LDSB, and DSB are negated.

STATES 8–10

The bus signals are unaltered during state 8 (S8) through state 10 (S10).

STATE 11

During state 11 (S11), ASB, is negated. The cycle terminates without the write portion of the cycle.

CASE READ 3: Only BERRB is received (see **3.4 Bus Error and Halt Operation**).

STATES 5–8

The bus signals are unaltered during state 5 (S5) through state 8 (S8).

STATE 9

During state 9 (S9), UDSB or LDSB, and DSB are negated.

STATE 10

During state 10 (S10), no bus signals are altered.

STATE 11

During state 11 (S11), ASB, is negated. The cycle terminates without the write portion of the cycle.

CASE WRITE 2: Only BERRB is received (see **3.4 Bus Error and Halt Operation**).

STATES 17–20

The bus signals are unaltered during state 17 (S17) through state 20 (S20).

STATE 21

During state 21 (S21), the SCM68000 negates ASB, UDSB or LDSB, and DSB.

RMCB is driven to a logic high after ASB is negated and before the falling edge of S1 of the next bus cycle. However, the value of RMCB is not guaranteed between bus cycles after ASB is negated for the cases described in this section.

3.2 BUS ARBITRATION

Bus arbitration is a technique used by bus master devices to request, to be granted, and to acknowledge bus mastership. Bus arbitration consists of the following:

1. Asserting a bus mastership request

2. Receiving a grant indicating that the bus is available at the end of the current bus cycle

3. Acknowledging that mastership has been assumed (3-wire bus arbitration only)

There are two ways to arbitrate the SCM68000 bus, 3-wire and 2-wire bus arbitration. Figure 3-14 and Figure 3-16 show 3-wire bus arbitration and Figure 3-15 and Figure 3-17 show 2-wire bus arbitration. BGACKB must be negated for 2-wire bus arbitration.

The timing diagram in Figure 3-16 shows that the bus request is negated within 1.5 clocks of the time that an acknowledge is asserted. This situation occurs when just one external device is requesting the bus. In systems having several devices that can be bus masters, bus request lines from these devices can be ORed at the SCM68000, and more than one bus request signal could occur.

The bus grant signal is negated 1.5 to 3.5 clock cycles after the assertion of the bus grant acknowledge signal. However, if bus request remains asserted (more than one device is requesting the bus), the SCM68000 reasserts bus grant for another request a few clock cycles after bus grant (for the previous request) is negated. In response to this additional assertion of bus grant, external arbitration circuitry selects the next bus master before the current bus master has completed the bus activity.

The timing diagram in Figure 3-17 shows just one external device requesting the bus. The 2-wire bus arbitration is best suited to systems with just one device, besides the CPU, capable of being bus master.
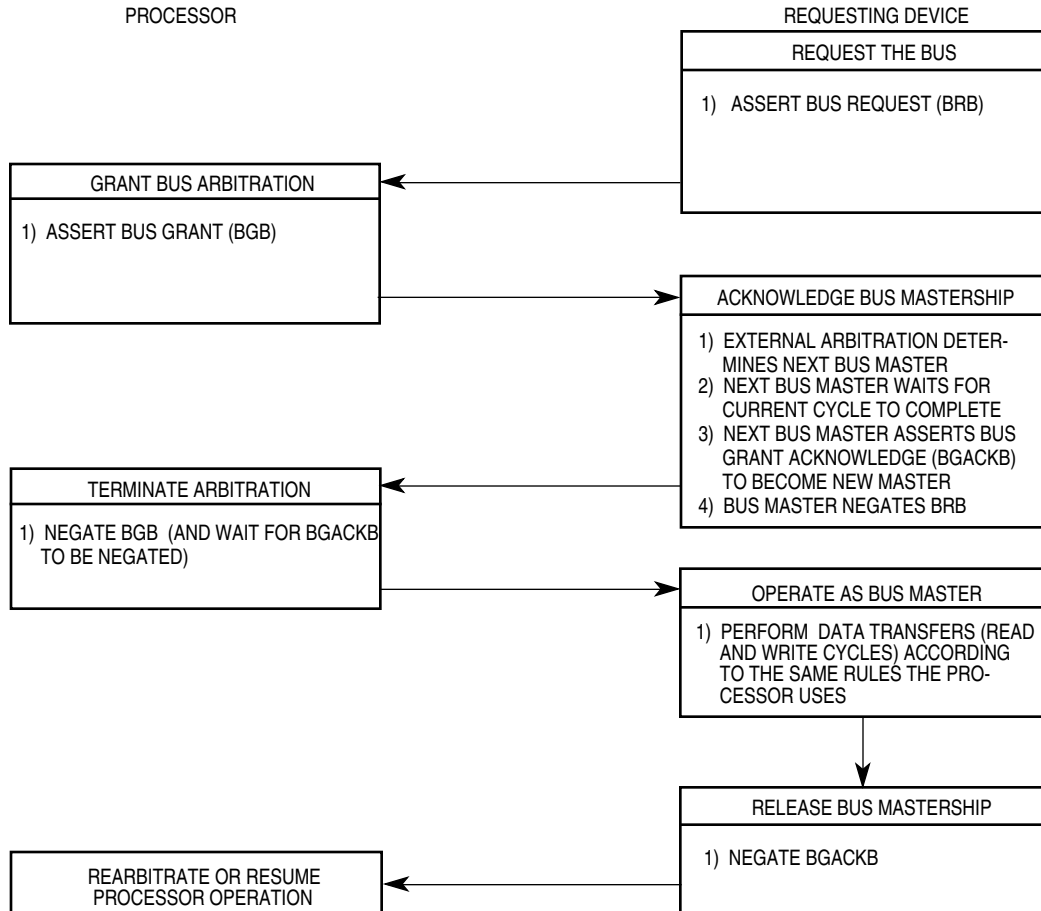
PROCESSOR

REQUESTING DEVICE

REQUEST THE BUS
1) ASSERT BUS REQUEST (BRB)

GRANT BUS ARBITRATION
1) ASSERT BUS GRANT (BGB)

ACKNOWLEDGE BUS MASTERSHIP
1) EXTERNAL ARBITRATION DETER-MINES NEXT BUS MASTER 2) NEXT BUS MASTER WAITS FOR CURRENT CYCLE TO COMPLETE 3) NEXT BUS MASTER ASSERTS BUS GRANT ACKNOWLEDGE (BGACKB) TO BECOME NEW MASTER 4) BUS MASTER NEGATES BRB

TERMINATE ARBITRATION
1) NEGATE BGB (AND WAIT FOR BGACKB TO BE NEGATED)

OPERATE AS BUS MASTER
1) PERFORM DATA TRANSFERS (READ AND WRITE CYCLES) ACCORDING TO THE SAME RULES THE PRO-CESSOR USES

RELEASE BUS MASTERSHIP
1) NEGATE BGACKB

REARBITRATE OR RESUME PROCESSOR OPERATION

Figure 3-14. 3-Wire Bus Arbitration Cycle Flowchart

3.2.1 Requesting the Bus

External devices capable of becoming bus masters assert BRB to request the bus. This signal can be ORed (not necessarily constructed from open-collector devices) from any of the devices in the system that can become bus master. The SCM68000, which is at a lower bus priority level than the external devices, relinquishes the bus after it completes the current bus cycle.

When no acknowledge is received before the bus request signal is negated, the SCM68000 continues to use the bus. Also, BGACKB allows arbitration time for another bus master to be overlapped with bus cycles to lessen bus idle time.

3.2.2 Receiving the Bus Grant

After BRB is asserted, the SCM68000 asserts BGB immediately following internal synchronization. The exception to this is when the SCM68000 has made an internal decision to execute the next bus cycle but has not yet asserted ASB for that cycle. In this case, BGB is delayed until ASB is asserted to indicate to external devices that a bus cycle is in progress.
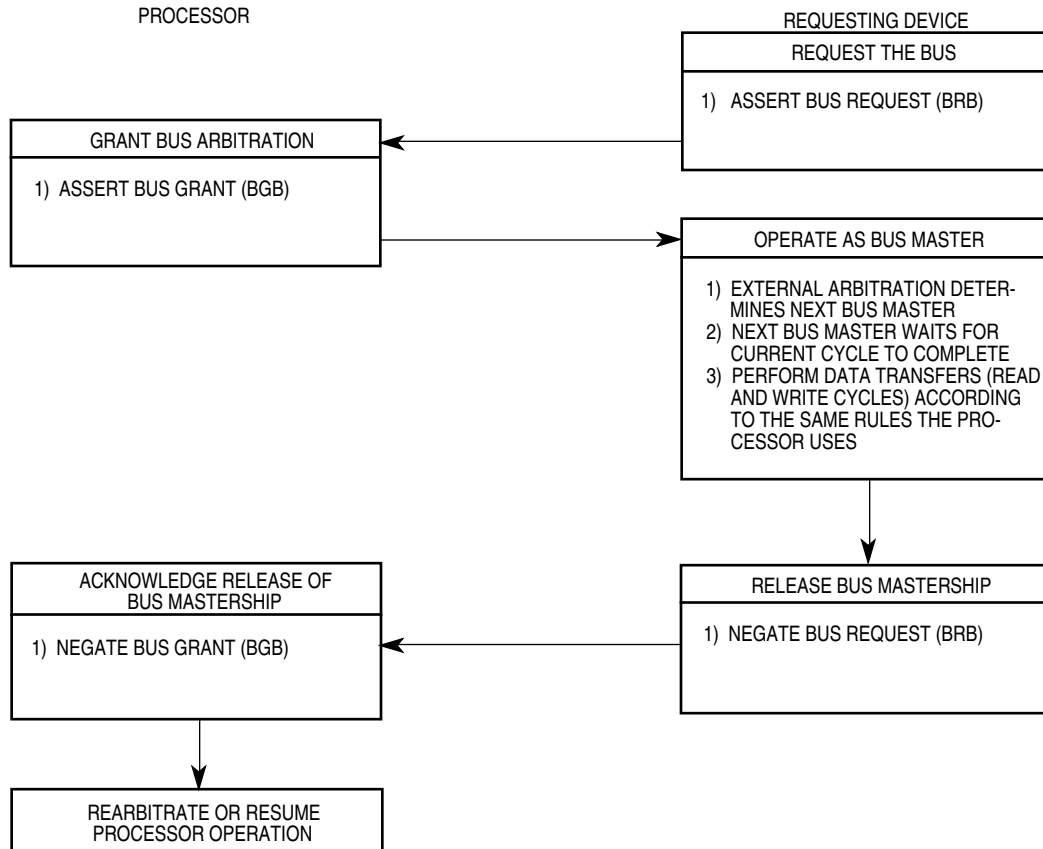
Figure 3-15. 2-Wire Bus Arbitration Cycle Flowchart

BGB can be routed through a daisy-chained network or through a specific priority-encoded network. Any method of external arbitration that observes the protocol can be used.

3.2.3 Acknowledgment of Mastership (3-Wire Bus Arbitration Only)

Upon receiving BGB, the requesting device waits until ASB, DTACKB, and BGACKB are negated before asserting BGACKB. The negation of ASB indicates that the previous bus master has completed its cycle. (No device is allowed to assume bus mastership while ASB is asserted.) The negation of BGACKB indicates that the previous master has released the bus. The negation of DTACKB indicates that the previous slave has terminated the connection to the previous master. (In some applications, DTACKB might not be included in this function; general-purpose devices would be connected using ASB only.) When BGACKB is asserted, the asserting device is bus master until it negates BGACKB. BGACKB should not be negated until after the bus cycle(s) is complete. A device relinquishes control of the bus by negating BGACKB.

The bus request from the granted device should be negated after BGACKB is asserted. If another bus request is pending, BGB is reasserted within a few clocks, as described in **3.3 Bus Arbitration Control**. The SCM68000 does not perform any external bus cycles before reasserting BGB.

Freescale Semiconductor, Inc.

Freescale Semiconductor, Inc.
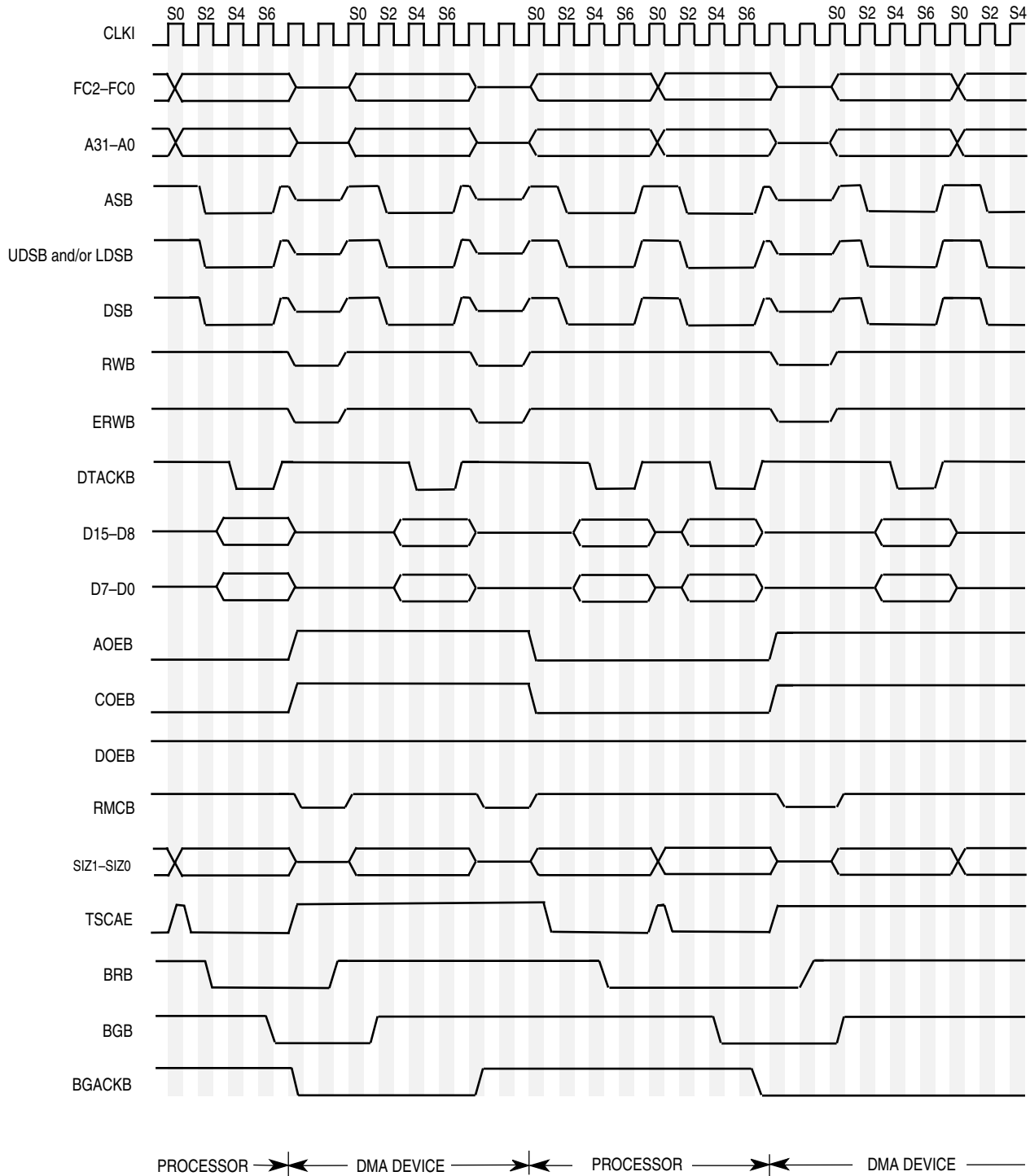
Figure 3-16. 3-Wire Bus Arbitration Timing Diagram

EC000 CORE PROCESSOR USER'S MANUAL
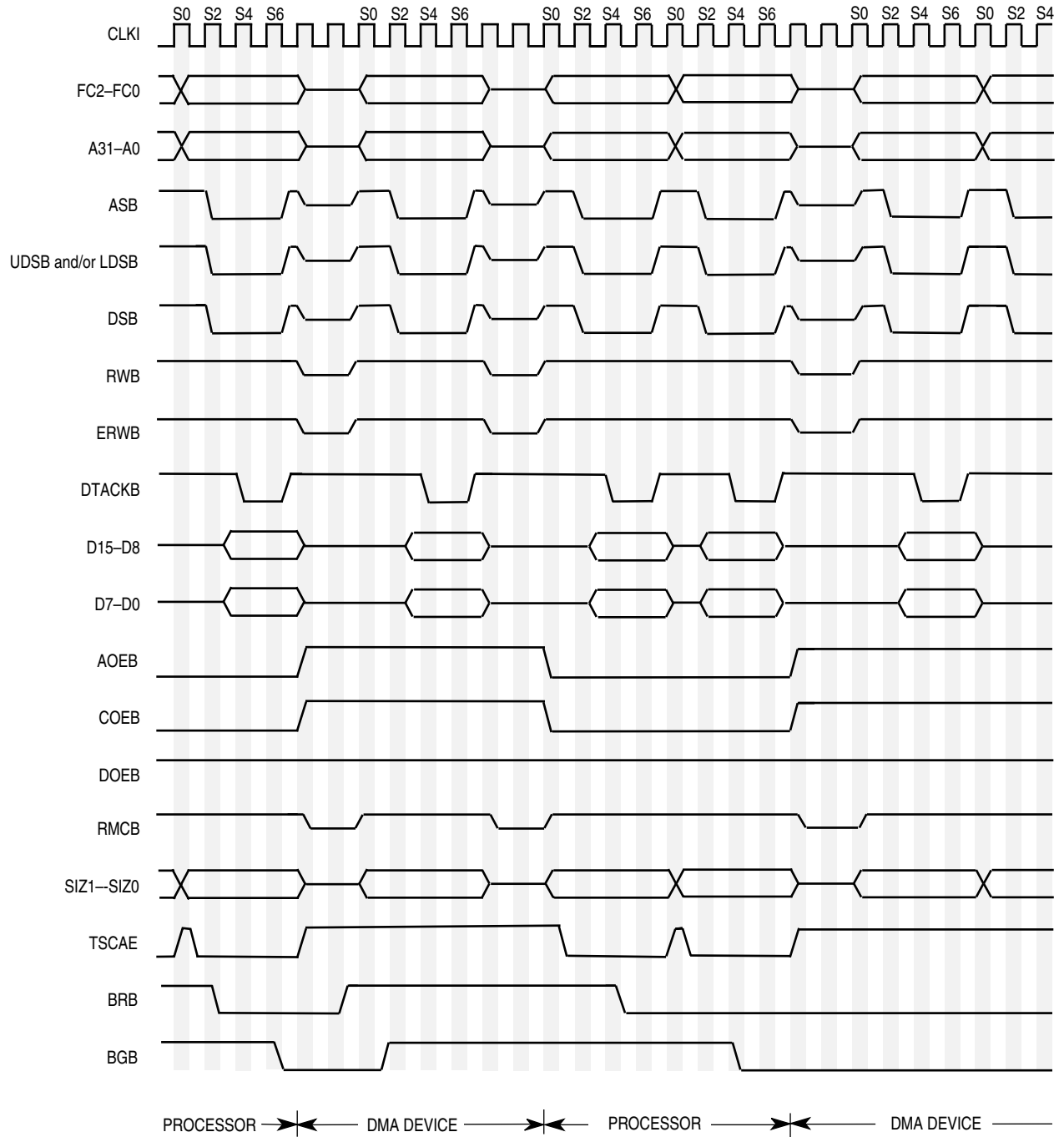
Figure 3-17. 2-Wire Bus Arbitration Timing Diagram

Freescale Semiconductor, Inc.

3.3 BUS ARBITRATION CONTROL

The asynchronous bus arbitration signals are synchronized before being used internally. See **3.5 Asynchronous Operation** for more information on the synchronization of these signals.

Bus arbitration control is implemented with a finite state machine. State diagram (a) in Figure 3-18 applies for 3-wire bus arbitration and state diagram (b) applies for 2-wire bus arbitration, in which BGACKB is permanently negated internally or externally. The same finite state machine is used, but it is effectively a three-state machine because BGACKB is always negated.

In Figure 3-18, input signals R (bus request internal) and A (bus grant acknowledge internal) are the internally synchronized versions of BRB and BGACKB. The BGB output is shown as G (bus grant), and the internal three-state control signal is shown as T (three-state control to bus control logic). If T is true, the address, data, and control buses are placed in the high-impedance state when ASB is negated. All signals are shown in positive logic (active high), regardless of their true active voltage level. State changes (valid outputs) occur on the next rising edge of the clock after the internal signal is valid.

A timing diagram of the bus arbitration sequence during an SCM68000 bus cycle is shown in Figure 3-19 and Figure 3-22. The bus arbitration timing while the bus is inactive (e.g., the SCM68000 is performing internal operations for a multiply instruction) is shown in Figure 3-20 and Figure 3-23.

When a bus request is made after the SCM68000 has begun a bus cycle and before ASB has been asserted (S0), the special sequence shown in Figure 3-21 and Figure 3-24 applies. Instead of being asserted on the next rising edge of clock, BGB is delayed until the second rising edge following its internal assertion.

Figure 3-19, Figure 3-20, and Figure 3-21 apply for 3-wire bus arbitration. Figure 3-22, Figure 3-23, and Figure 3-24 apply for 2-wire bus arbitration.
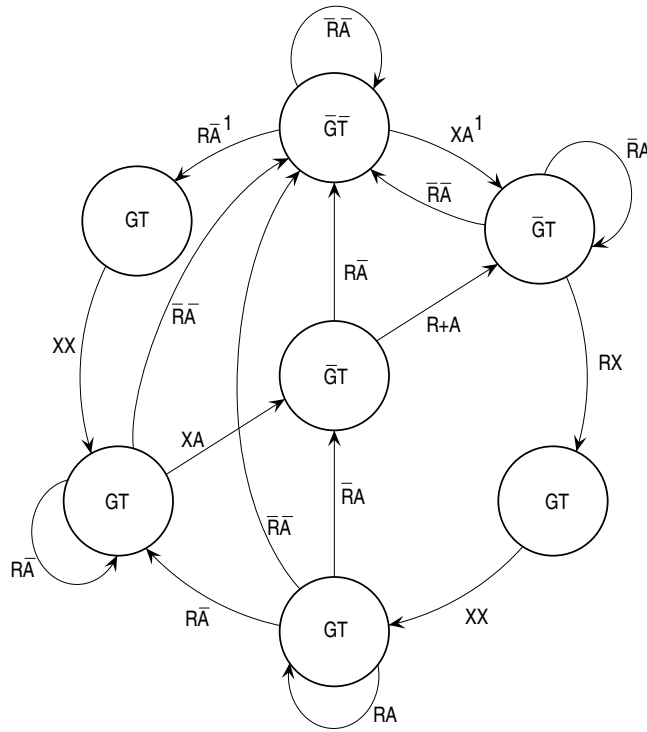
(a) 3-Wire Bus Arbitration
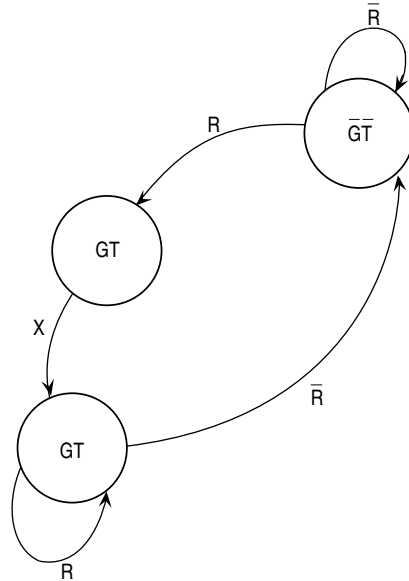
(b) 2-Wire Bus Arbitration

R = Bus Request Internal
A = Bus Grant Acknowledge Internal
G = Bus Grant
T = Three-State Control to Bus Control Logic
X = Don't Care

NOTES:

1. State machine will not change if the bus is S0 or S1. Refer to 3.3 BUS ARBITRATION CONTROL
2. The address bus will be placed in the high-impedance state if T is asserted and ASB is negated.
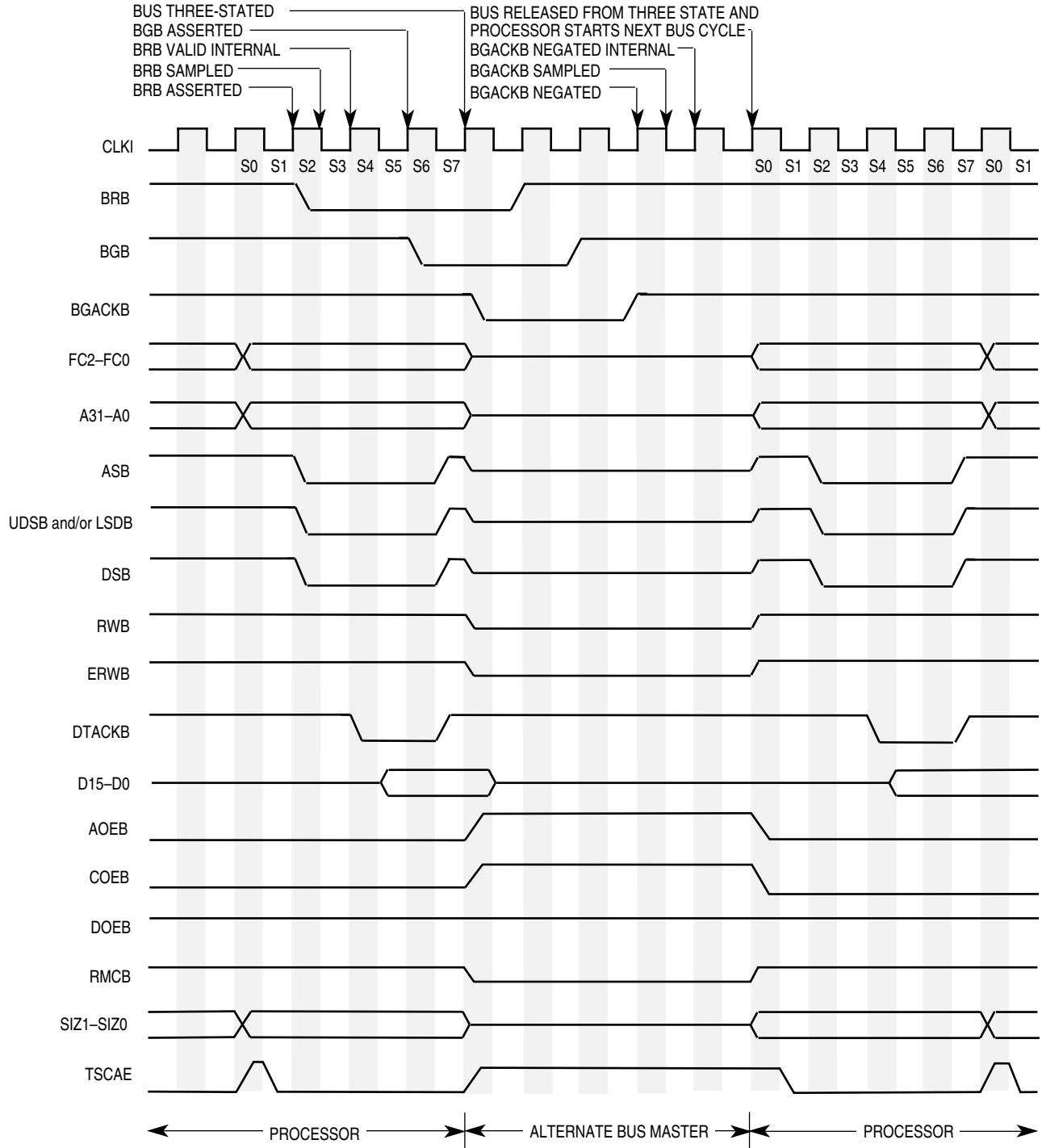
Figure 3-18. Bus Arbitration Unit State Diagrams

Freescale Semiconductor, Inc.

Figure 3-19. 3-Wire Bus Arbitration Timing Diagram—SCM68000 Active
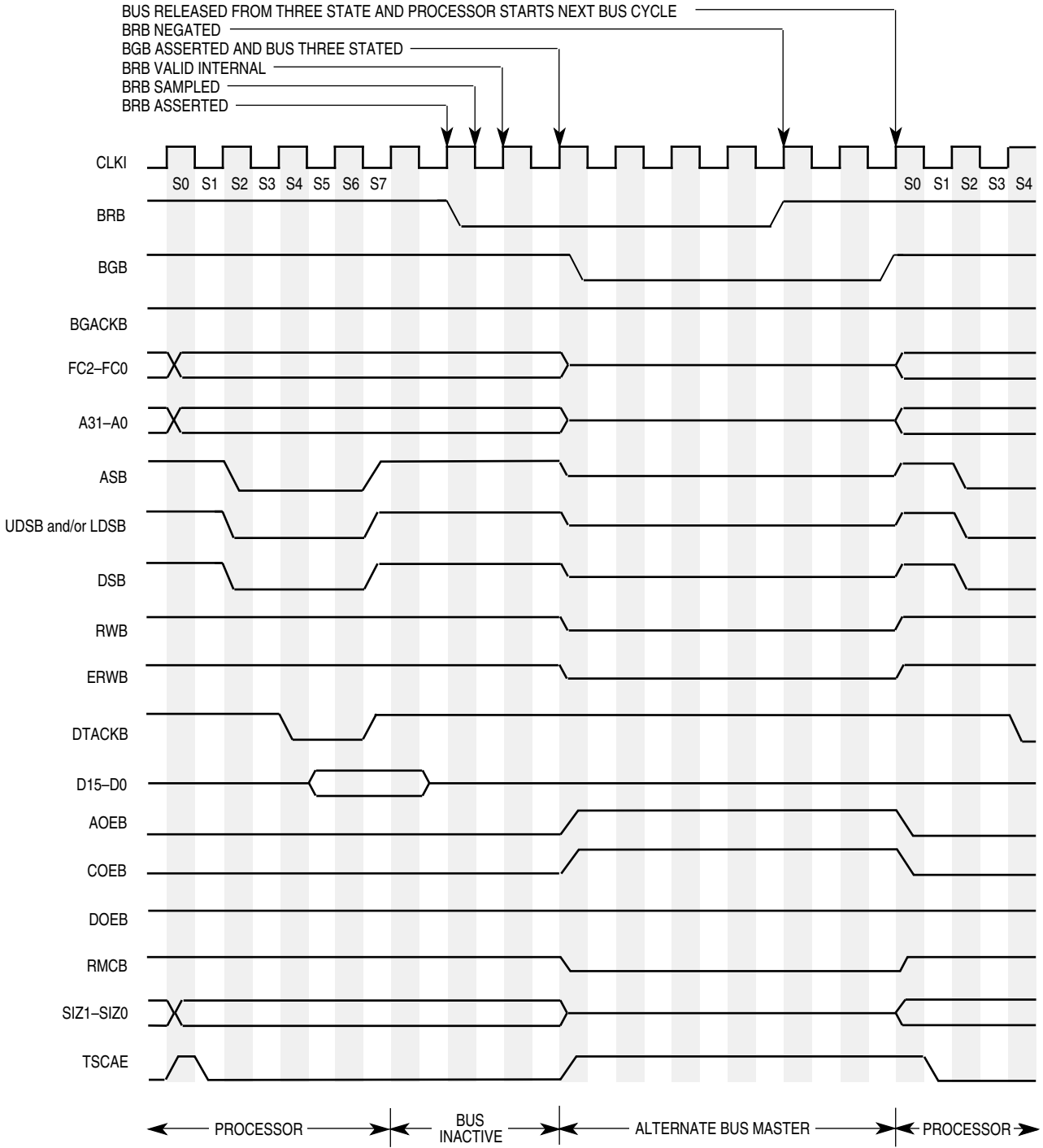
EC000 CORE PROCESSOR USER'S MANUAL MOTOROLA

Figure 3-20. 3-Wire Bus Arbitration Timing Diagram—Bus Inactive
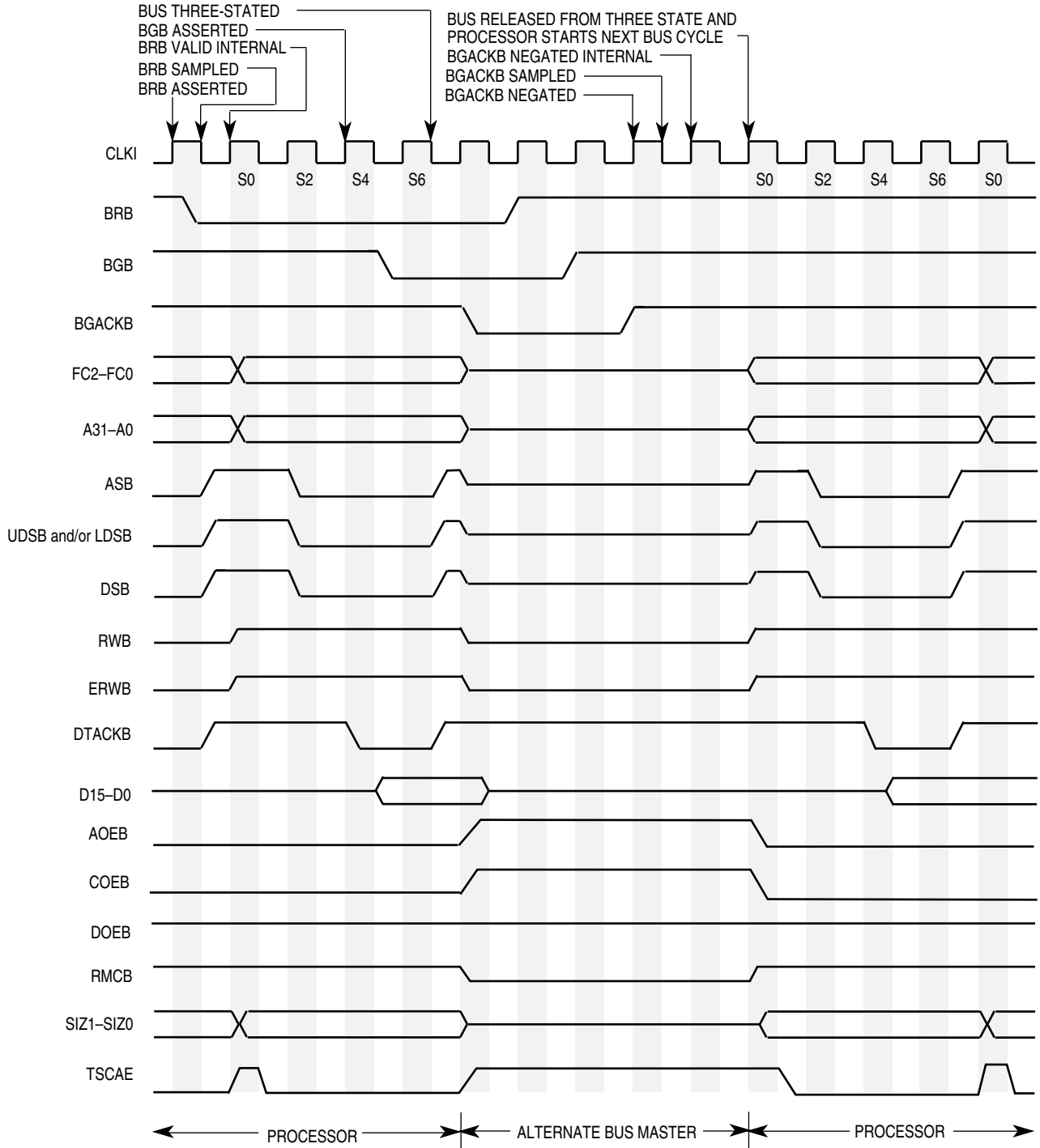
Figure 3-21. 3-Wire Bus Arbitration Timing Diagram—Special Case
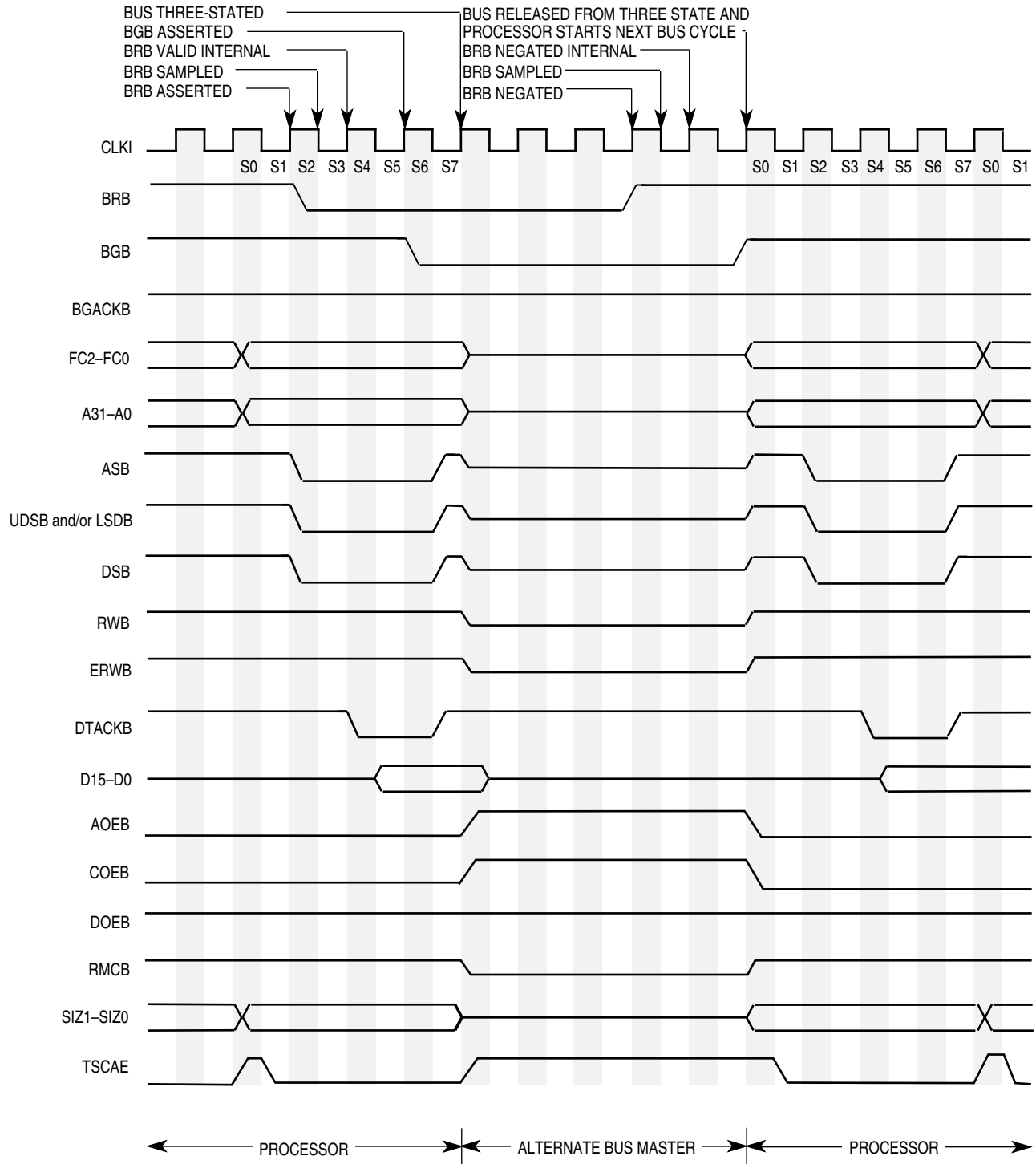
Figure 3-22. 2-Wire Bus Arbitration Timing Diagram—SCM68000 Active

Freescale Semiconductor, Inc.

BUS RELEASED FROM THREE STATE AND PROCESSOR STARTS NEXT BUS CYCLE
BRB NEGATED
BGB ASSERTED AND BUS THREE STATED
BRB VALID INTERNAL
BRB SAMPLED
BRB ASSERTED

Figure 3-23. 2-Wire Bus Arbitration Timing Diagram—Bus Inactive

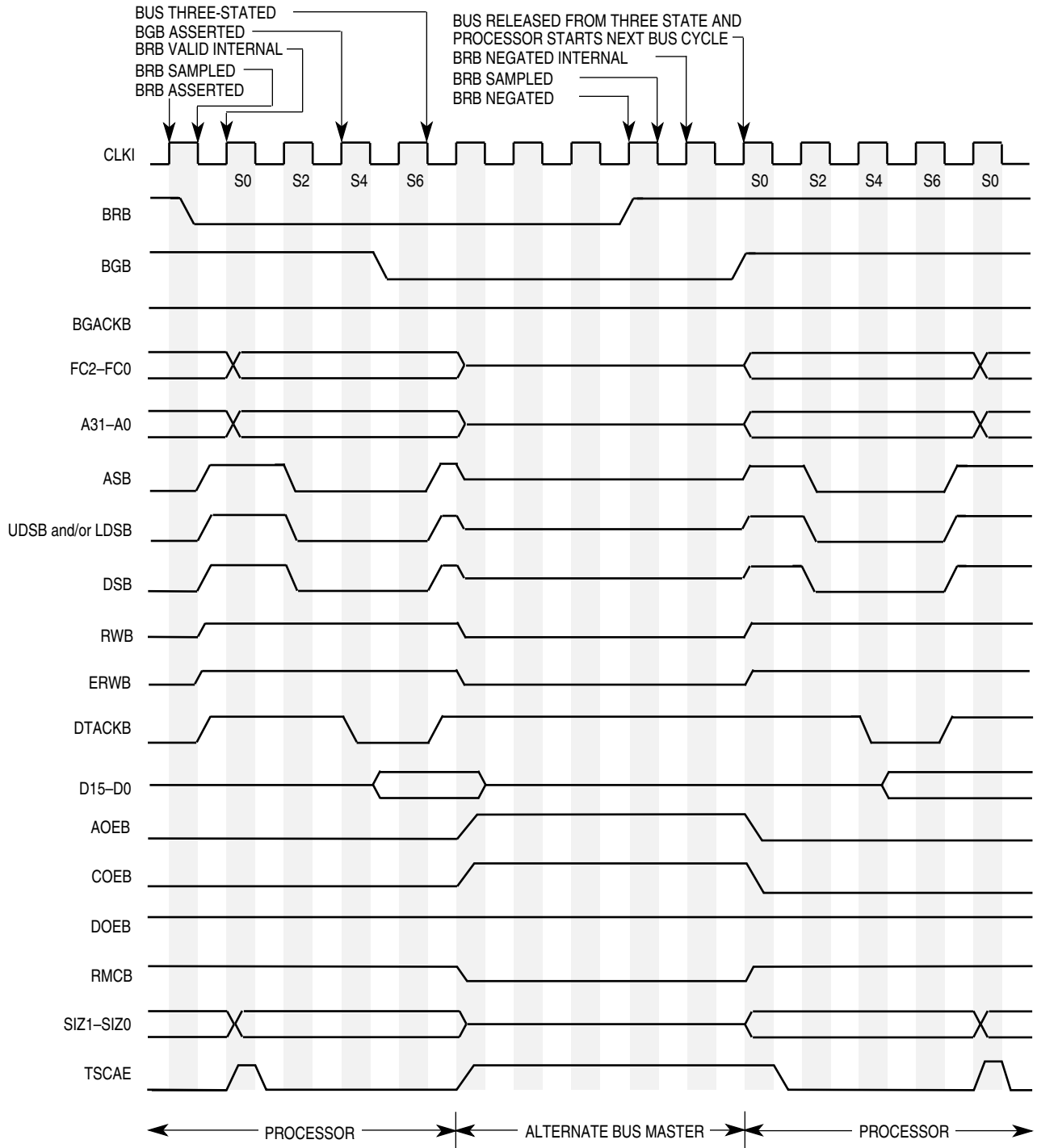**For More Information On This Product,**
Go to: www.freescale.com

Figure 3-24. 2-Wire Bus Arbitration Timing Diagram—Special Case

3.4 BUS ERROR AND HALT OPERATION

In a bus architecture that requires a handshake from an external device, such as the asynchronous bus used in the SCM68000, the handshake may not always occur. A bus error input is provided to terminate a bus cycle in error when the expected signal is not asserted. Different systems and different devices within the same system require different maximum response times. External circuitry can be provided to assert the bus error signal after the appropriate delay following the assertion of address strobe.

3.4.1 Bus Error Operation

A bus error is recognized when HALTIB is negated and BERRB is asserted, either alone or with DTACKB.

When the bus error condition is recognized, the current bus cycle is terminated in state 9 (S9) (only BERRB is asserted) or in state 7 (S7) (BERRB and DTACKB are asserted) for a read cycle or a write cycle. The bus cycle is terminated in state 11 (S11) for the read portion of a read-modify-write cycle. For the write portion of a read-modify-write cycle, the bus cycle is terminated in state 21 (S21) (only BERRB is asserted) or in state 19 (S19) (BERRB and DTACKB are asserted). As long as BERRB remains asserted, the data bus is in the high-impedance state. Figure 3-25 shows the timing for the normal bus error.

After the aborted bus cycle is terminated and BERRB is negated, the SCM68000 enters exception processing for the bus error exception. During the exception processing sequence, the following information is placed on the supervisor stack:

1. Status register
2. Program counter (two words, which may be up to five words past the instruction being executed)
3. Error information

The first two items are identical to the information stacked by any other exception. The SCM68000 stacks bus error information to help determine and to correct the error.

After the SCM68000 has placed the required information on the stack, the bus error exception vector is read from vector table entry 2 (offset $08) and placed in the program counter. The SCM68000 resumes execution at the address in the vector, which is the first instruction in the bus error handler routine.
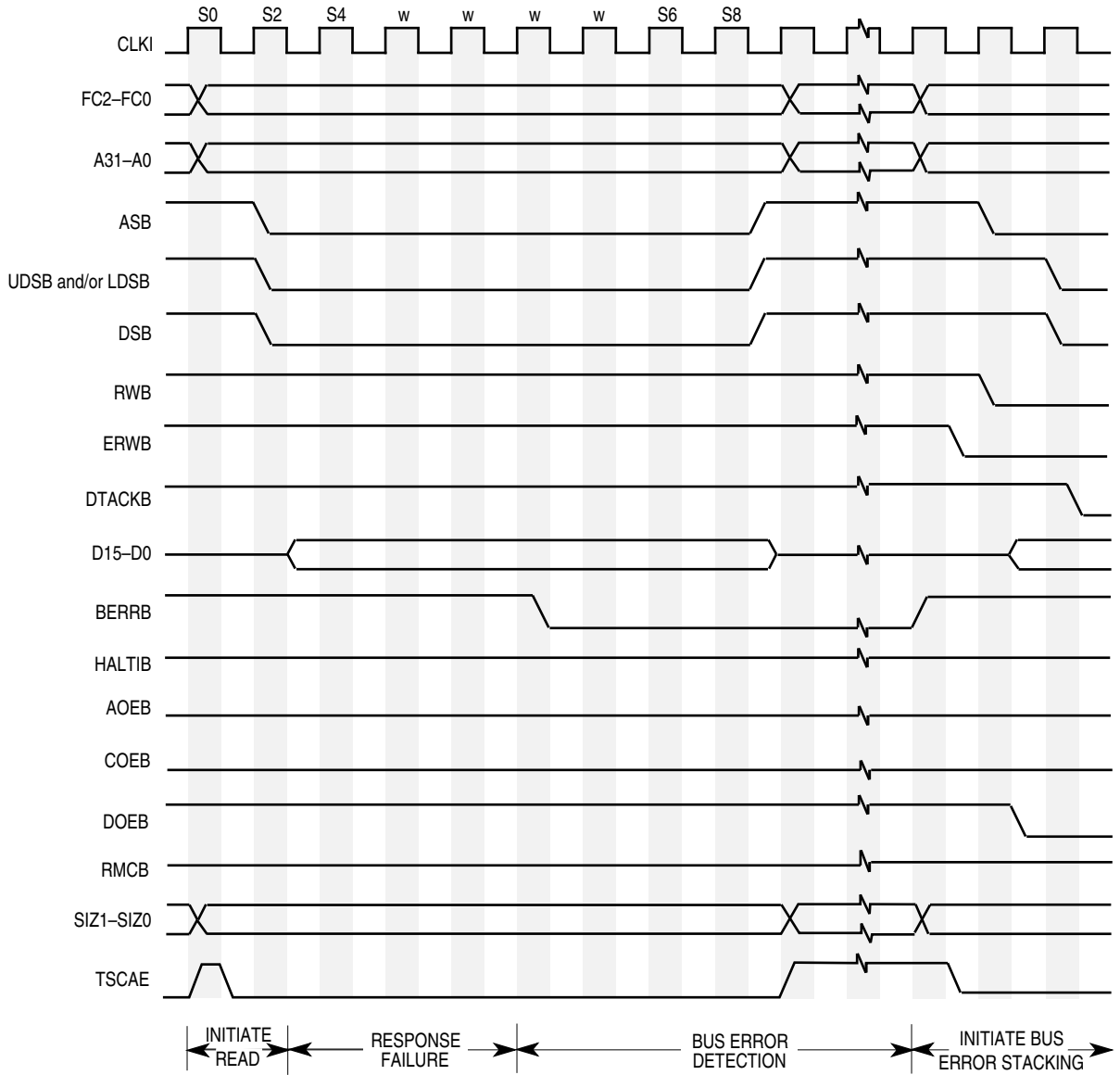
Figure 3-25. Bus Error Timing Diagram

Freescale Semiconductor, Inc.

3.4.2 Retrying the Bus Cycle

The assertion of BERRB during a bus cycle in which HALTIB is also asserted by an external device initiates a retry operation. Figure 3-26 is a timing diagram of the retry operation.

The SCM68000 terminates the bus cycle, then puts the data bus in the high-impedance state. The SCM68000 remains in this state until HALTIB is negated. Then the SCM68000 retries the preceding cycle using the same function codes, address, and data (for a write operation). BERRB should be negated at least one clock cycle before HALTIB is negated.

NOTE

To guarantee that the entire read-modify-write cycle runs correctly and that the write portion of the operation is performed without negating the address strobe, the SCM68000 does not retry a read-modify-write cycle. When BERRB is asserted during a read-modify-write operation, a bus error operation is performed whether or not HALTIB is asserted.

3.4.3 Halt Operation

HALTIB performs a halt/run/single-step operation similar to the halt operation of an MC68000. When HALTIB is asserted by an external device, the SCM68000 halts and remains halted as long as the signal remains asserted, as shown in Figure 3-27.

While the SCM68000 is halted, only the data bus is placed in the high-impedance state as shown in Table 2-1. Bus arbitration is performed as usual. Should a bus error occur while HALTIB is asserted, the SCM68000 performs the retry operation previously described.

The single-step mode is derived from correctly timed transitions of HALTIB. HALTIB is negated to allow the SCM68000 to begin a bus cycle, then asserted to enter the halt mode when the cycle completes. The single-step mode proceeds through a program one bus cycle at a time for debugging purposes. The halt operation and the hardware trace capability allow tracing of either bus cycles or instructions one at a time. These capabilities and a software debugging package provide total debugging flexibility.

NOTE

Execution of the RESET instruction while using the HALTIB signal in the single-step mode can cause the SCM68000 to reset. **4.3.1 Reset** has more detailed information about the RESET instruction.
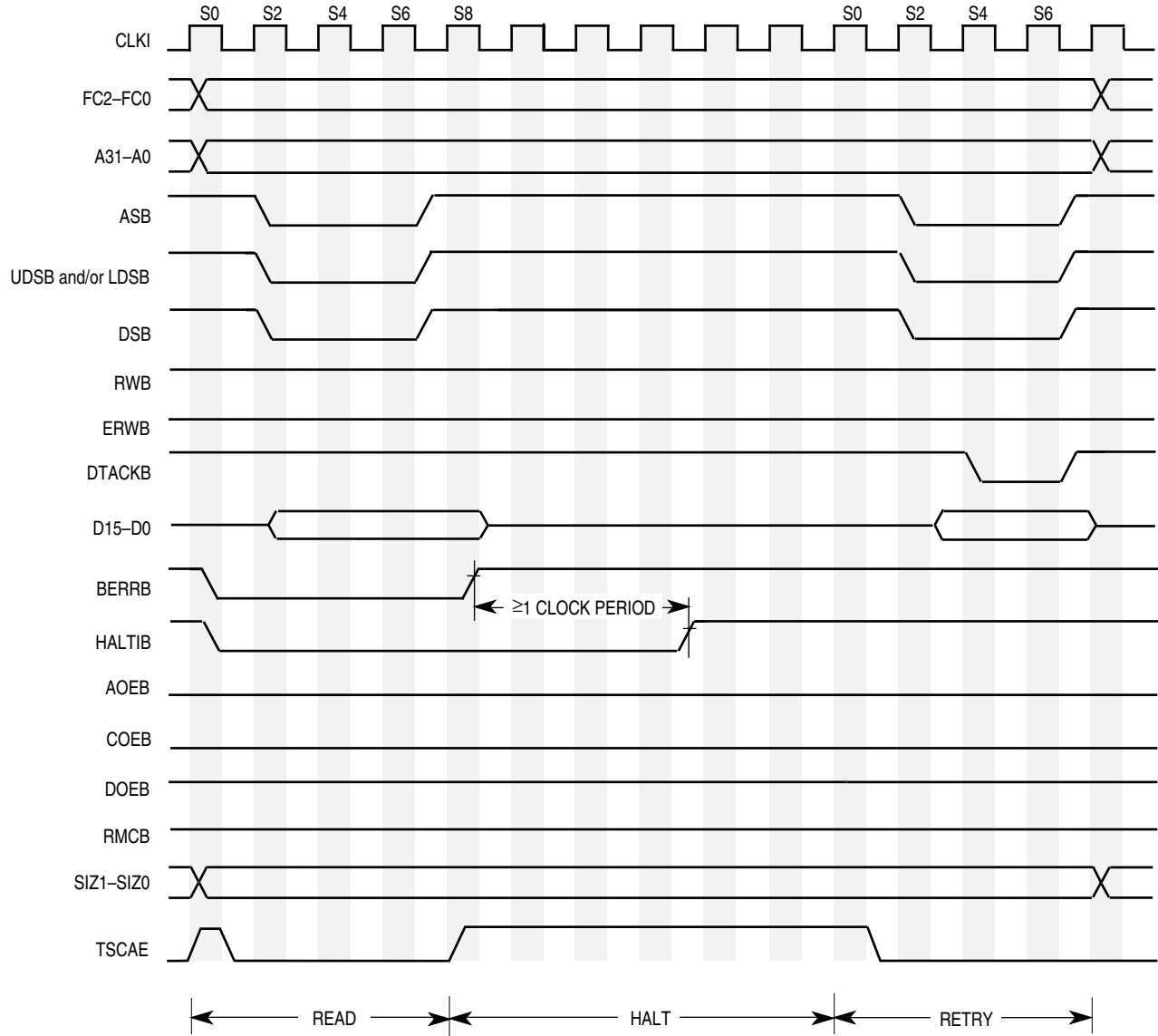
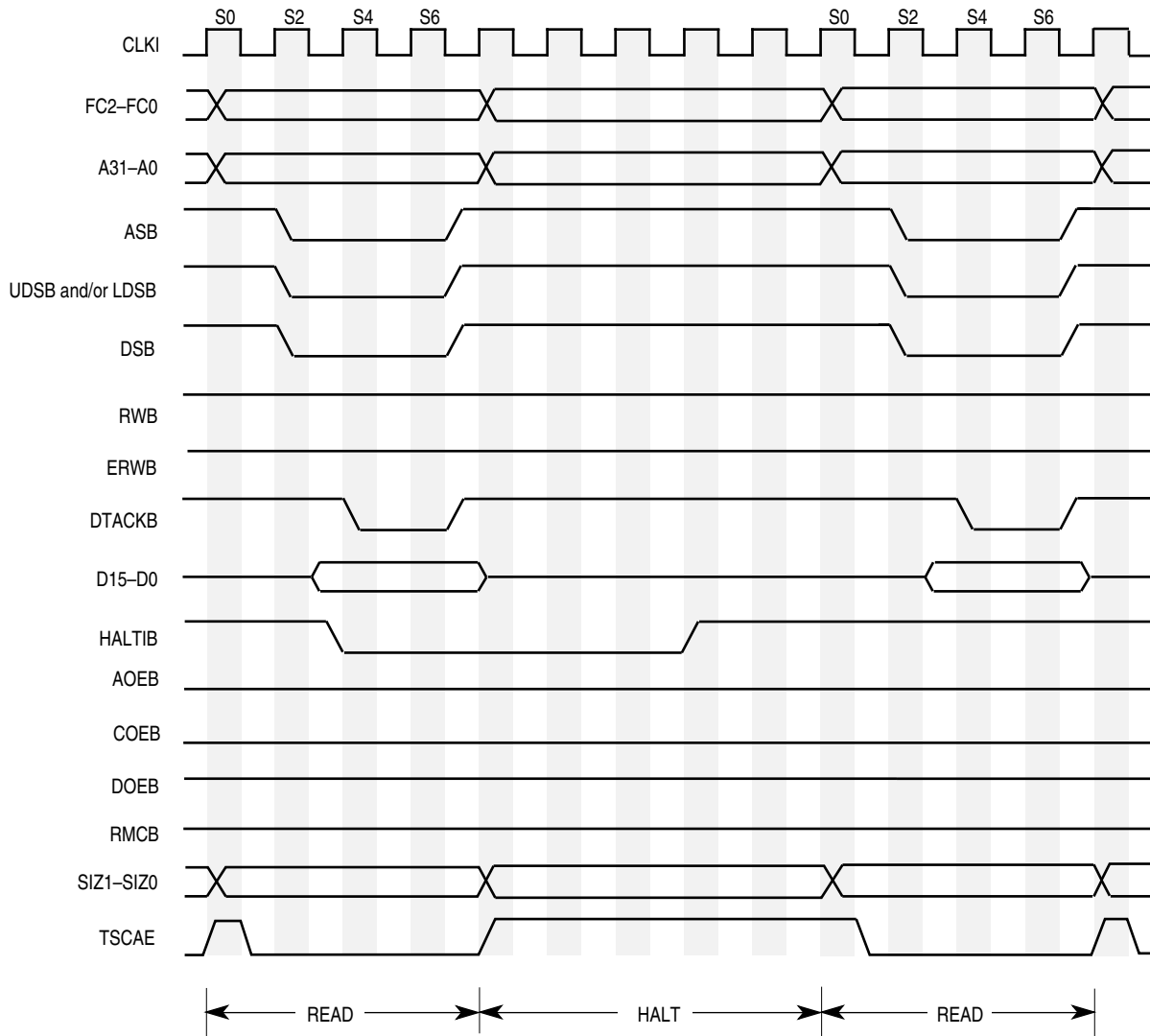**Figure 3-26. Retry Bus Cycle Timing Diagram**

Freescale Semiconductor, Inc.

Figure 3-27. Halt Operation Timing Diagram

EC000 CORE PROCESSOR USER'S MANUAL MOTOROLA
For More Information On This Product,
Go to: www.freescale.com

3.4.4 Double Bus Fault

When a bus error exception occurs, the SCM68000 begins exception processing by stacking information on the supervisor stack. If another bus error occurs during exception processing (i.e., before execution of another instruction begins) the SCM68000 halts and asserts HALTOB. This situation is a double bus fault. Only an external reset operation can restart a SCM68000 halted due to a double bus fault.

A retry operation does not initiate exception processing; a bus error during a retry operation does not cause a double bus fault. The SCM68000 can continue to retry a bus cycle indefinitely if external hardware requests.

A double bus fault occurs during a reset operation when a bus error occurs while the SCM68000 is reading the vector table (before the first instruction is executed). The reset operation is described in **4.3.1 Reset**.

3.5 ASYNCHRONOUS OPERATION

All asynchronous input signals to the SCM68000 are synchronized before being used internally. As shown in Figure 3-28, synchronization requires a maximum of one cycle of the system clock, assuming that the asynchronous input setup time (spec #47, defined in **Section 7 Electrical Characteristics**) has been met. The input asynchronous signal is sampled on the falling edge of the clock and is valid internally after the next rising edge. The asynchronous inputs are AVECB, RESETIB, HALTIB, DTACKB, BERRB, IPLB2–IPLB0, BRB, and BGACKB.
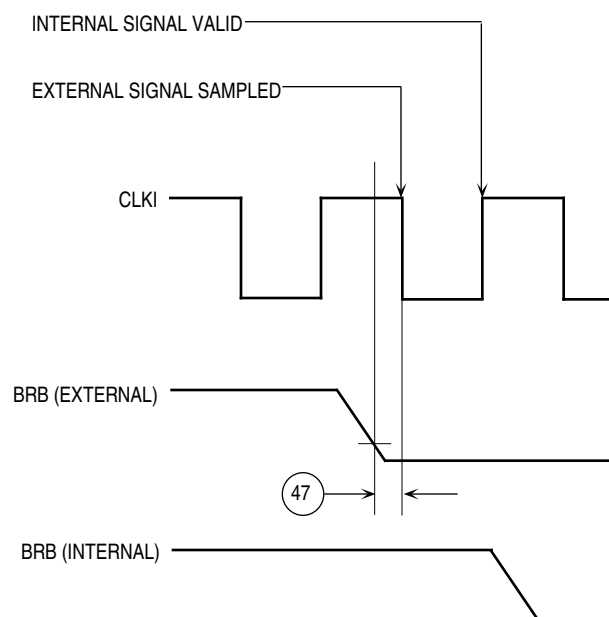

Figure 3-28. External Asynchronous Signal Synchronization

Freescale Semiconductor, Inc.

To achieve clock frequency independence at a system level, the bus can be operated in an asynchronous manner. Asynchronous bus operation uses the bus handshake signals to control the transfer of data. The handshake signals are ASB, UDSB, LDSB, DSB, DTACKB, BERRB, HALTIB, and AVECB. ASB indicates the start of the bus cycle, and UDSB, LDSB, and DSB signal valid data for a write cycle. After placing the requested data on the data bus (read cycle) or latching the data (write cycle), the slave device (memory or peripheral) asserts DTACKB to terminate the bus cycle. If no device responds or if the access is invalid, external control logic asserts BERRB, or BERRB and HALTIB, to abort or retry the cycle. Figure 3-29 shows the use of the bus handshake signals in a fully asynchronous read cycle. Figure 3-30 shows a fully asynchronous write cycle.
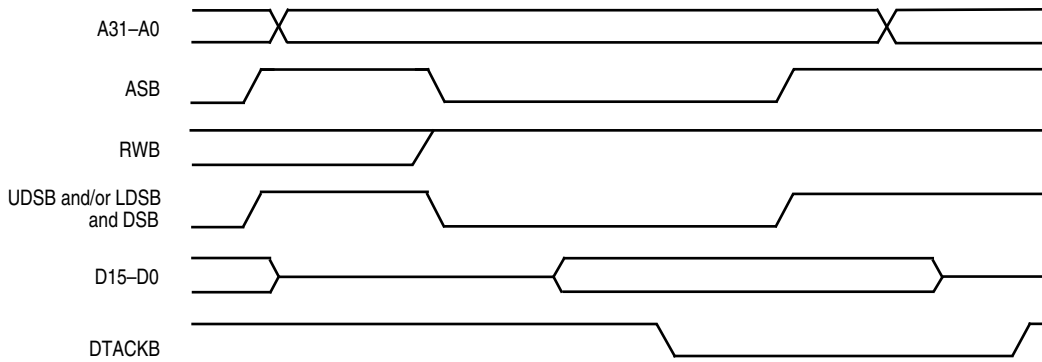
Figure 3-29. Fully Asynchronous Read Cycle
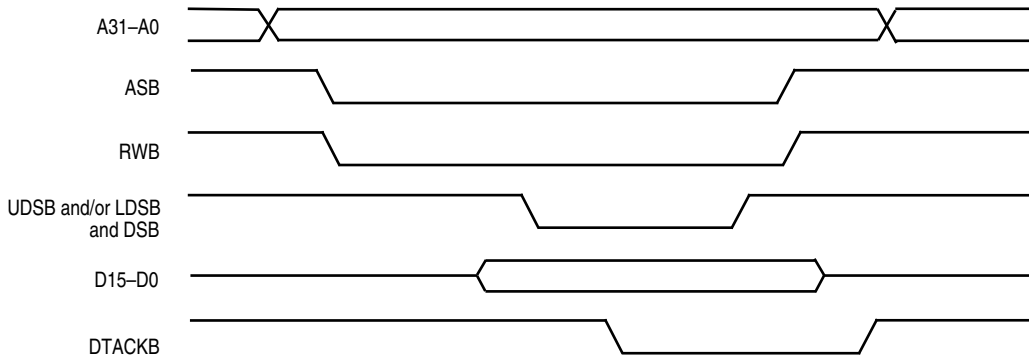
Figure 3-30. Fully Asynchronous Write Cycle

In the asynchronous mode, the accessed device operates independently of the frequency and phase of the system clock. For example, the MC68681 dual universal asynchronous receiver/transmitter (DUART) does not require any clock-related information from the bus master during a bus transfer. Asynchronous devices are designed to operate correctly with processors at any clock frequency when relevant timing requirements are observed.

EC000 CORE PROCESSOR USER'S MANUAL MOTOROLA
For More Information On This Product,
Go to: www.freescale.com

A device can use a clock at the same frequency as the system clock, but without a defined phase relationship to the system clock. This mode of operation is pseudo-asynchronous; it increases performance by observing timing parameters related to the system clock frequency without being completely synchronous with that clock. A common example of a pseudo-asynchronous device is a memory array designed to operate with the SCM68000 at a certain frequency but is not driven by the SCM68000 clock.

The designer of a fully asynchronous system can make no assumptions about address setup time, which could be used to improve performance. However, with the system clock frequency known, the slave device can be designed to decode the address bus before recognizing an address strobe. Parameter #11 (refer to **Section 7 Electrical Characteristics** for all parameters listed in this section) specifies the minimum time before address strobe during which the address is valid.

In a pseudo-asynchronous system, timing specifications allow DTACKB to be asserted for a read cycle (see Figure 3-31) before the data from a slave device is valid. The length of time that DTACKB may precede data is specified as parameter #31 in Figure 3-31. This parameter must be met to ensure the validity of the data latched into the SCM68000. No maximum time is specified from the assertion of ASB to the assertion of DTACKB. During this unlimited time, the SCM68000 inserts wait cycles in one-clock-period increments until DTACKB is recognized. Figure 3-31 shows the important timing parameters for a pseudo-asynchronous read cycle.
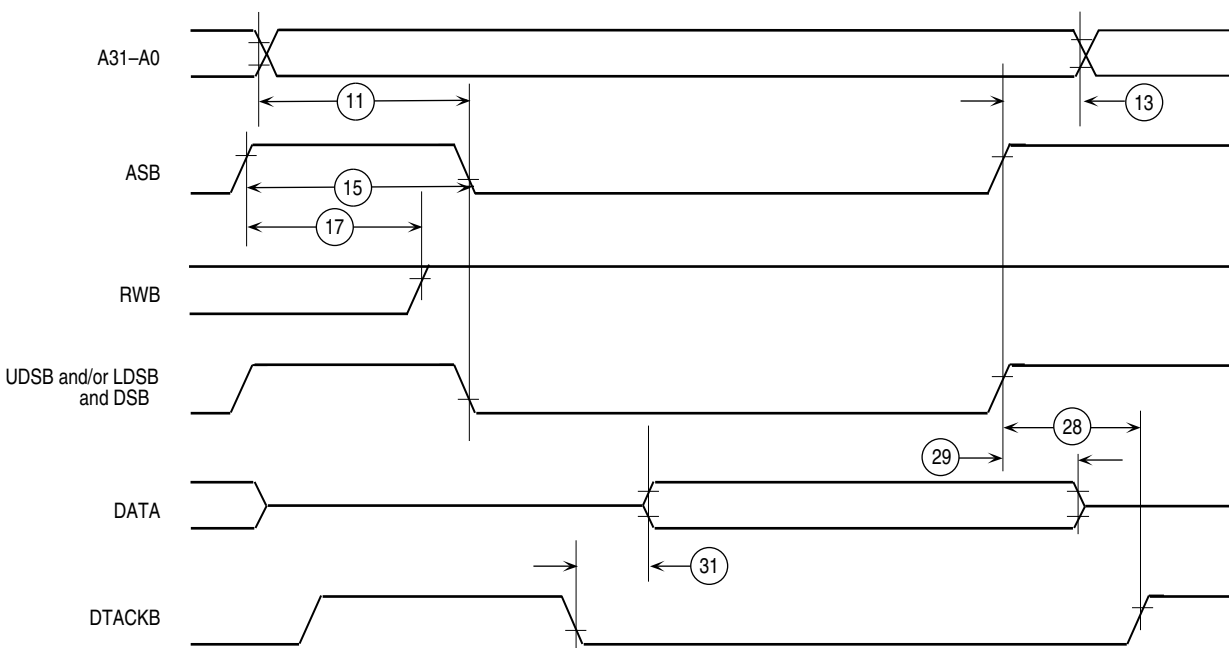

Figure 3-31. Pseudo-Asynchronous Read Cycle

During a write cycle (see Figure 3-32), after the SCM68000 asserts ASB but before driving the data bus, the SCM68000 drives RWB to a logic low. Parameter #55 specifies the mini-

mum time between the transition of RWB and the driving of the data bus, which is effectively the maximum turnoff time for any device driving the data bus.

After the SCM68000 places valid data on the bus, it asserts the data strobe signals. A data setup time, similar to the address setup time previously discussed, can be used to improve performance. Parameter #26 is the minimum time a slave device can accept valid data before recognizing a data strobe. The slave device asserts DTACKB after it accepts the data. Parameter #25 is the minimum time after negation of the strobes during which the valid data remains on the address bus. Parameter #28 is the maximum time between the negation of the strobes by the SCM68000 and the negation of DTACKB by the slave device. If DTACKB remains asserted past the time specified by parameter #28, the SCM68000 may recognize it as being asserted early in the next bus cycle and may terminate that cycle prematurely. Figure 3-32 shows the important timing parameters for a pseudo-asynchronous write cycle.
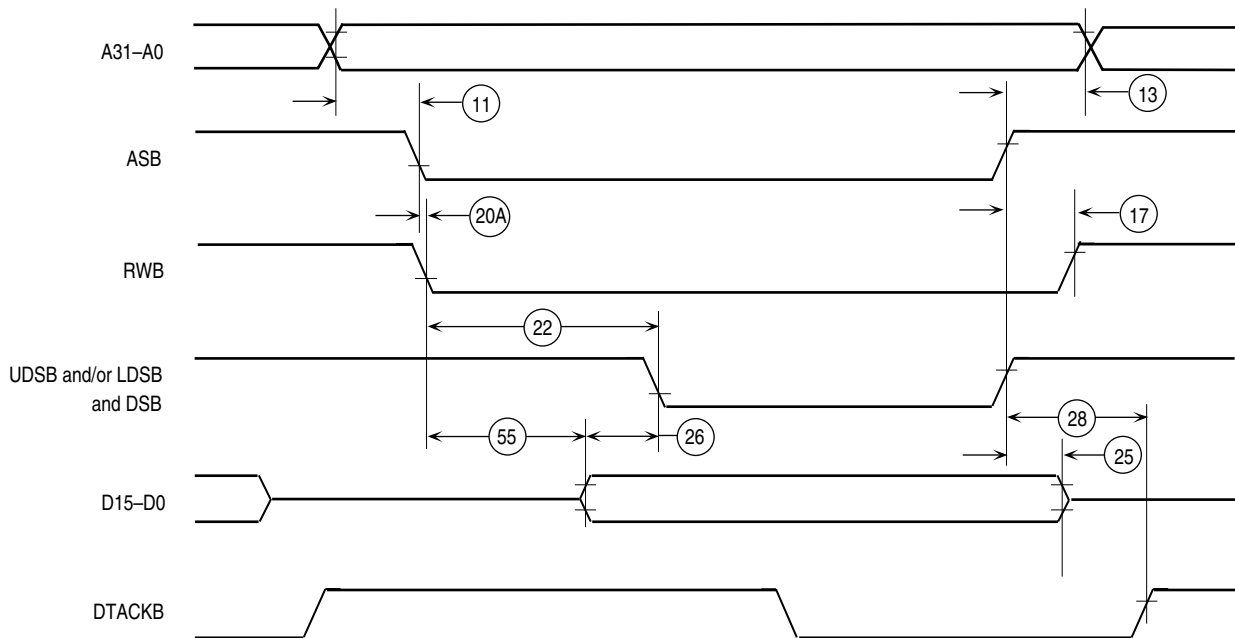
Figure 3-32. Pseudo-Asynchronous Write Cycle

3.6 SYNCHRONOUS OPERATION

In some systems, external devices use the system clock to generate DTACKB and other asynchronous input signals. This synchronous operation provides a closely coupled design with maximum performance, appropriate for frequently accessed parts of the system. For example, memory can operate in the synchronous mode, but peripheral devices operate asynchronously. For a synchronous device, the designer uses explicit timing information shown in **Section 7 Electrical Characteristics**. These specifications define the state of all bus signals relative to a specific state of the SCM68000 clock.

The standard SCM68000 bus cycle consists of four clock periods (eight bus cycle states) and, optionally, an integral number of clock cycles inserted as wait states. Wait states are

inserted as required to allow sufficient response time for the external device. The following state-by-state description of the bus cycle differs from those descriptions in **3.1.1 Read Cycle** and **3.1.2 Write Cycle** by including information about the important timing parameters that apply in the bus cycle states.

Figure 3-33 shows a synchronous read cycle and the important timing parameters that apply. The timing for a synchronous write cycle, including relevant timing parameters, is shown in Figure 3-34.
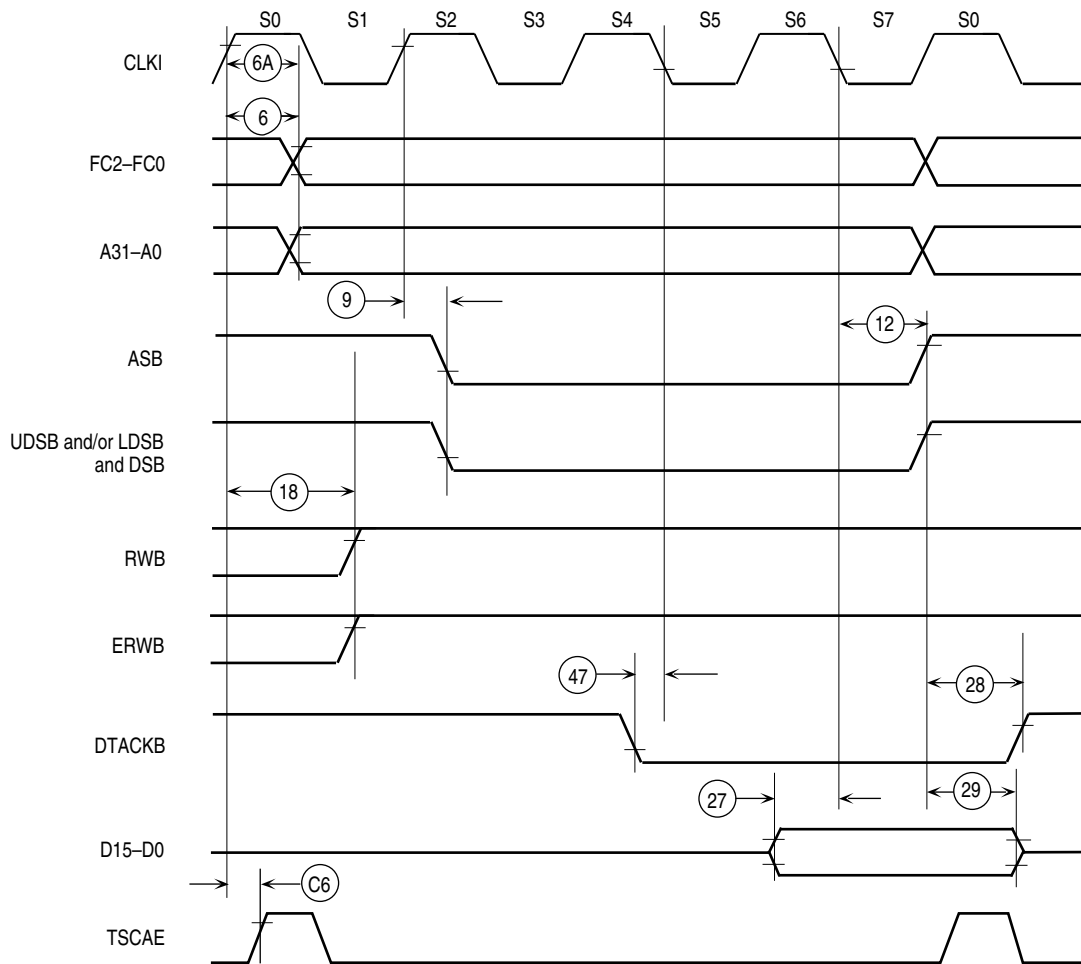
Figure 3-33. Synchronous Read Cycle

STATE 0

The bus cycle starts in S0, during which the clock is high. At the rising edge of S0, the function code for the access is driven externally. Parameter #6A defines the delay from this rising edge until the function codes are valid. The address of the accessed device is driven externally with an assertion delay defined by parameter #6. The RWB and ERWB signals are driven to logic high; parameter #18 defines the delay from the same rising edge to the transition of RWB. The minimum value for parameter #18 applies to a read
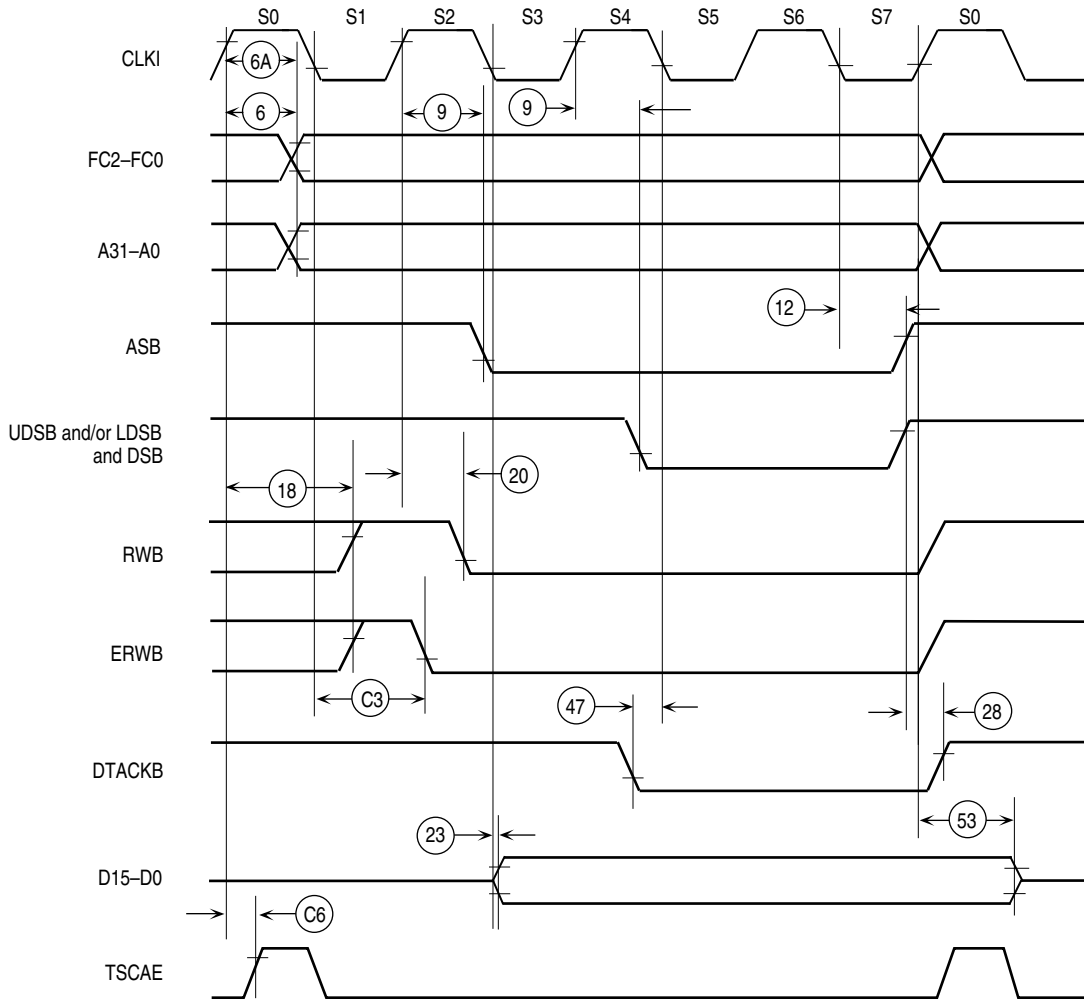
Figure 3-34. Synchronous Write Cycle

cycle preceded by a write cycle; this value is the maximum hold time for a logic low on RWB beyond the initiation of the read cycle. The TSCAE signal is driven to a logic high. Parameter #C6 defines the delay from the rising edge of the clock to the assertion of TSCAE.

STATE 1

Entering S1, a low period of the clock, TSCAE is driven to a logic low. During a write, the ERWB signal is driven to a logic low with an assertion delay defined by parameter #C3.

STATE 2

On the rising edge of S2, a high period of the clock, ASB is asserted. During a read cycle, UDSB, LDSB, and DSB are also asserted at this time. Parameter #9 defines the assertion delay for these signals. For a write cycle, the RWB signal is driven to a logic low with a delay defined by parameter #20.

STATE 3

On the falling edge of the clock entering S3 during a write cycle, the data bus is driven out of the high-impedance state with the data being written to the accessed device. Parameter #23 specifies the data assertion delay. In a read cycle, no signal is altered in S3.

STATE 4

Entering the high clock period of S4 during a write cycle, UDSB, LDSB, and DSB are asserted on the rising edge of the clock. As in S2 for a read cycle, parameter #9 defines the assertion delay from the rising edge of S4 for UDSB, LDSB, and DSB. In a read cycle, no signal is altered by the SCM68000 during S4.

Until the falling edge of the clock at the end of S4 (beginning of S5), no response from any external device except RESETIB is acknowledged by the SCM68000. If either DTACKB or BERRB is asserted before the falling edge of S4 and satisfies the input setup time defined by parameter #47, the SCM68000 enters S5 and the bus cycle continues. If either DTACKB or BERRB is asserted but without meeting the setup time defined by parameter #47, the SCM68000 may recognize the signal and continue the bus cycle; the result is unpredictable. If neither DTACKB nor BERRB is asserted before the next falling edge of the clock, the bus cycle remains in S4, and wait states (complete clock cycles) are inserted until one of the bus cycle termination conditions is met.

STATE 5

S5 is a low period of the clock, during which the SCM68000 does not alter any signal.

STATE 6

S6 is a high period of the clock, during which data for a read operation is set up relative to the falling edge (entering S7). Parameter #27 defines the minimum period by which the data must precede the falling edge. For a write operation, the SCM68000 changes no signal during S6.

STATE 7

On the falling edge of the clock entering S7, the SCM68000 latches data and negates ASB and UDSB, LDSB, and DSB during a read cycle. The hold time for these strobes from this falling edge is specified by parameter #12. The hold time for data relative to the negation of ASB and UDSB, LDSB, and DSB is specified by parameter #29. For a write cycle, only ASB and UDSB, LDSB, and DSB are negated; timing parameter #12 also applies.

During a write cycle, on the rising edge of the clock at the end of S7 (which may be the start of S0 for the next bus cycle), the SCM68000 also places the data bus in the high-impedance state and drives RWB and ERWB to a logic high. External logic circuitry should respond to the negation of the ASB and UDSB, LDSB, and DSB by negating DTACKB and/or BERRB. Parameter #28 is the hold time for DTACKB, and parameter #30 is the hold time for BERRB.

A key consideration when designing in a synchronous environment is the timing for the assertion of DTACKB and BERRB by an external device. To properly use external inputs, the SCM68000 must synchronize these signals to the internal clock. The SCM68000 must sample the external signal and determine whether to consider it high or low during the suc-

ceeding clock period. The external signal has no defined phase relationship to the CPU clock and may be changing at sampling time. Successful synchronization requires that the internal machine receives a valid logic level (not a metastable signal), whether the input is high, low, or in transition. Metastable signals propagating through synchronous machines can produce unpredictable operation.

Parameter #47 of **Section 7 Electrical Characteristics** is the asynchronous input setup time. Signals that meet parameter #47 are guaranteed to be recognized at the next falling edge of the system clock. However, signals that do not meet parameter #47 are not guaranteed to be recognized. In addition, if DTACKB is recognized on a falling edge, valid data is latched into the SCM68000 (during a read cycle) on the next falling edge, provided the data meets the setup time required (parameter #27). When parameter #27 has been met, parameter #31 may be ignored. If DTACKB is asserted with the required setup time before the falling edge of S4, no wait states are incurred, and the bus cycle runs at its maximum speed of four clock periods.

3.7 THE RELATIONSHIP OF DTACKB, BERRB, AND HALTIB

To properly control termination of a bus cycle for a retry or a bus error condition, DTACKB, BERRB, and HALTIB should meet the setup and hold time to the falling edge of the SCM68000 clock. Specification #48 (see **Section 7 Electrical Characteristics**), can be ignored when DTACKB, BERRB, and HALTIB are stable at the falling edge of the SCM68000 clock.

The possible bus cycle termination can be summarized as follows (case numbers refer to Table 3-1):

- Normal Termination—DTACKB is asserted. BERRB and HALTIB remain negated (case 1).

- Halt Termination—HALTIB is asserted coincident with or preceding DTACKB, and BERRB remains negated (case 2).

- Bus Error Termination—BERRB is asserted in lieu of, coincident with, or preceding DTACKB (case 3). HALTIB remains negated, and BERRB is negated coincident with or after DTACKB.

- Retry Termination—HALTIB and BERRB are asserted in lieu of, coincident with, or before DTACKB (cases 4, 5, and 6). BERRB is negated coincident with or after DTACKB. HALTIB must be held at least one cycle after BERRB.

Table 3-1 shows the details of the resulting bus cycle termination for various combinations of signal sequences.

Table 3-1. DTACKB, BERRB, and HALTIB Assertion Results

Case No.	Control Signal Input	Asserted on Rising Edge of State*		Result
		N	N+2	
1	DTACKB BERRB HALTIB	A NA NA	S X X	Normal cycle terminate and continue.
2	DTACKB BERRB HALTIB	A NA A/S	S X S	Normal cycle terminate and halt. Continue when HALTIB negated.
3	DTACKB BERRB HALTIB	X A NA	X S NA	Terminate and take bus error trap.
4	DTACKB BERRB HALTIB	NA A NA	X S A	Terminate and retry when HALTIB negated.
5	DTACKB BERRB HALTIB	X A A	X S S	Terminate and retry when HALTIB negated.
6	DTACKB BERRB HALTIB	NA NA A	X A S	Terminate and retry when HALTIB negated.

LEGEND:
  N — The number of the current even bus state (e.g., S4, S6, etc.)
  A — Signal asserted in this bus state
  NA — Signal not asserted in this bus state
  X — Don't care
  S — Signal asserted in preceding bus state and remains asserted in this state

*The DTACKB, BERRB, and HALTIB signals are subject to the setup and hold time (spec #47, defined in **Section 7 Electrical Characteristics**) before they are sampled on the falling edge of the previous state. "Asserted" in this table refers to the time when the signals are valid internally. See **3.5 Asynchronous Operation** for more details on external asynchronous signal synchronization.

The negation of BERRB and HALTIB under several conditions is shown in Table 3-2. DTACKB is assumed to be negated normally in all cases. For reliable operation, both DTACKB and BERRB should be negated when address strobe is negated.

Table 3-2 shows when BERRB and HALTIB should be negated with respect to when they were asserted to produce various results. The first column describes which case in Table 3-1 is being used for asserting the signals. The third column shows the current bus state and the fourth column shows the following bus state. The last column describes what will happen in the next bus cycle given the conditions described in the previous columns.

Table 3-2. BERRB and HALTIB Negation Results

Conditions of Termination in Table 3-1	Control Signal Input	Negated on Rising Edge of State*		Results—Next Cycle
		N	N+2	
Normal (cases 1 and 2)	BERRB HALTIB	• •		May lengthen next cycle.
Normal (cases 1 and 2)	BERRB HALTIB	•	•	May lengthen next cycle.
Normal (cases 1 and 2)	BERRB HALTIB	•	•	If next cycle is started, it will be terminated as a bus error.
Normal (cases 1 and 2)	BERRB HALTIB	• none		If next cycle is started, it will be terminated as a bus error.
Bus Error (case 3)	BERRB HALTIB	• none		Takes bus error trap.
Rerun (cases 4, 5, and 6)	BERRB HALTIB	• •		Illegal sequence; usually traps to vector number 0.
Rerun (cases 4, 5, and 6)	BERRB HALTIB	•	•	Illegal sequence; usually traps to vector number 0.
Rerun (cases 4, 5, and 6)	BERRB HALTIB	•	•	Reruns the bus cycle.

LEGEND:

    N —   The number of the current even bus state (e.g., S4, S6, etc.)

    • —   Signal is negated in this bus state.

 none —   Signal was not asserted.

*The BERRB and HALTIB signals are subject to the setup and hold time (spec #47, defined in **Section 7 Electrical Characteristics**) before they are sampled on the falling edge of the previous state. "Negated" in this table refers to the time when the signals are valid internally. See **3.5 Asynchronous Operation** for more details on the external asynchronous signal synchronization.

EXAMPLE A:

A system uses a watchdog timer to terminate accesses to unused address space. The timer asserts BERRB after timeout (case 3).

EXAMPLE B:

A system uses error detection on random-access memory (RAM) contents. The system designer may:

1. Delay DTACKB until the data is verified. If data is invalid, return BERRB and HALTIB simultaneously to retry the error cycle (case 5).

2. Delay DTACKB until the data is verified. If data is invalid, return BERRB at the same time as DTACKB to take a bus error trap (case 3).

SECTION 4
EXCEPTION PROCESSING

This section describes operations of the SCM68000 (EC000 core)[1] outside the normal processing associated with the execution of instructions. The functions of the bits in the supervisor portion of the status register are described: the supervisor/user bit, the trace enable bit, and the interrupt priority mask. Finally, the sequence of memory references and actions taken by the SCM68000 for exception conditions are described in detail.

The SCM68000 is always in one of three processing states: normal, exception, or halted. The normal processing state is associated with instruction execution; the memory references are to fetch instructions and operands and to store results. A special case of the normal state is the stopped state, resulting from the execution of a STOP instruction. In this state, no further memory references are made.

The exception processing state is associated with interrupts, trap instructions, tracing, and other exceptional conditions. The exception may be internally generated by an instruction or by an unusual condition arising during the execution of an instruction. Externally, exception processing can be forced by an interrupt, a bus error, or a reset. Exception processing provides an efficient context switch so that the SCM68000 can handle unusual conditions.

The halted processing state is an indication of catastrophic hardware failure. For example, if during the exception processing of a bus error another bus error occurs, the SCM68000 assumes the system is unusable and halts. Only an external reset can restart the halted SCM68000. Note that the stopped state is not the same as the halted state.

4.1 PRIVILEGE MODES

The SCM68000 operates in one of two levels of privilege: the supervisor mode or the user mode. The privilege mode determines which operations are legal. The mode is optionally used by an external memory management device to control and translate accesses. The mode is also used to choose between the supervisor stack pointer (SSP) and the user stack pointer (USP) in instruction references.

The privilege mode is a mechanism for providing security in a computer system. Programs should access only their own code and data areas and should be restricted from accessing information that they do not need and must not modify. The operating system executes in the supervisor mode, allowing it to access all resources required to perform the overhead tasks for the user mode programs. Most programs execute in user mode, in which the accesses are controlled and the effects on other parts of the system are limited.

[1.] The SCM68000 is the name of the Verilog model for the EC000 Core. The remainder of this section will refer to the part as only the SCM68000

4.1.1 Supervisor Mode

The supervisor mode has the higher level of privilege. The mode of the SCM68000 is determined by the S-bit of the status register; if the S-bit is set, the SCM68000 is in the supervisor mode. All instructions can be executed in the supervisor mode. The bus cycles generated by instructions executed in the supervisor mode are classified as supervisor references. While the SCM68000 is in the supervisor mode, those instructions that use either the system stack pointer implicitly or address register seven explicitly access the SSP.

4.1.2 User Mode

The user mode has the lower level of privilege. If the S-bit of the status register is clear, the SCM68000 is executing instructions in the user mode.

Most instructions execute identically in either mode. However, some instructions having important system effects are designated privileged. For example, user programs are not permitted to execute the STOP instruction or the RESET instruction. To ensure that a user program cannot enter the supervisor mode except in a controlled manner, the instructions that modify the entire status register are privileged. To aid in debugging system software, the move to user stack pointer (MOVE to USP) and move from user stack pointer (MOVE from USP) instructions are privileged.

The bus cycles generated by an instruction executed in user mode are classified as user references. Classifying a bus cycle as a user reference allows an external memory management device to control access to protected portions of the address space. While the SCM68000 is in the user mode, those instructions that use either the system stack pointer implicitly or address register seven explicitly access the USP.

4.1.3 Privilege Mode Changes

The transition from supervisor to user mode can be accomplished by any of four instructions: return from exception (RTE), move to status register (MOVE to SR), AND immediate to status register (ANDI to SR), and exclusive OR immediate to status register (EORI to SR). The RTE instruction fetches the new status register and program counter from the supervisor stack and loads each into its respective register. Next, it begins the instruction fetch at the new program counter address in the privilege mode determined by the S-bit of the new contents of the status register.

Once the SCM68000 is in the user mode and is executing instructions, only exception processing can change the privilege mode. During exception processing, the current state of the S-bit of the status register is saved, and the S-bit is set, putting the SCM68000 in the supervisor mode. Therefore, when instruction execution resumes at the address specified to process the exception, the SCM68000 is in the supervisor privilege mode.

The MOVE to SR, ANDI to SR, and EORI to SR instructions fetch all operands in the supervisor mode, perform the appropriate update to the status register, and then fetch the next instruction at the next sequential program counter address in the privilege mode determined by the new S-bit. The instruction following the MOVE/ANDI/EORI SR instruction will be fetched twice, once from the old FC space and again from the new FC space (even if the S-

bit was not modified). External memory management hardware should not treat the access in the old FC space as an error.

4.1.4 Reference Classification

When the SCM68000 makes a reference, it classifies the reference according to the encoding of the three function code output lines. This classification allows external translation of addresses, control of access, and differentiation of special SCM68000 states, such as CPU space (used by interrupt acknowledge cycles). Table 4-1 lists the classification of references.

Table 4-1. Reference Classification

Function Code Output			Address Space
FC2	**FC1**	**FC0**	
0	0	0	(Reserved)*
0	0	1	User Data
0	1	0	User Program
0	1	1	(Undefined)*
1	0	0	(Reserved)*
1	0	1	Supervisor Data
1	1	0	Supervisor Program
1	1	1	CPU Space

* Address space 3 is reserved for user definition, while 0 and 4 are reserved for future use by Motorola.

4.1.5 CPU Space Cycle

A CPU space cycle, indicated when the function codes are all high, is a special cycle. Bits A19–A16 of the address bus identify sixteen types of CPU space cycles. The interrupt acknowledge cycle, in which A19–A16 are high, is currently the only defined CPU space cycle for the SCM68000. Other configurations of A19–A16 are reserved by Motorola to define other types of CPU cycles used in other M68000 Family microprocessors. Figure 4-1 shows the encoding of CPU space addresses.
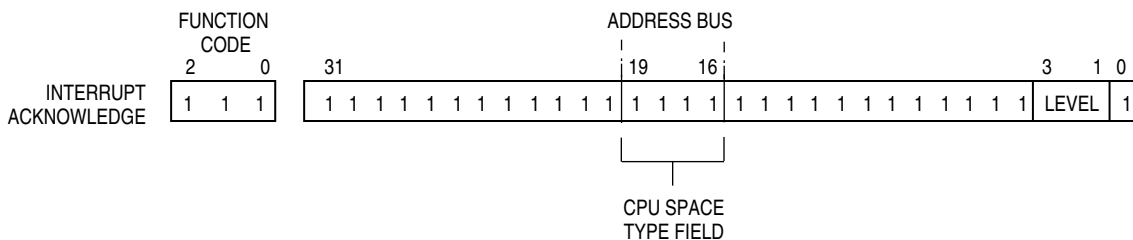
Figure 4-1. CPU Space Address Encoding

4.1.5.1 INTERRUPT ACKNOWLEDGE CYCLE. The interrupt acknowledge cycle places the level of the interrupt being acknowledged on address bits A3–A1 and drives all other address lines high. For a vectored interrupt, the interrupt acknowledge cycle reads a vector number when the interrupting device places a vector number on the data bus and asserts DTACKB to acknowledge the cycle. The timing diagram for a vectored interrupt is shown in Figure 4-2.

Figure 4-2. Interrupt Acknowledge Cycle Timing Diagram (Sheet 1 of 3)
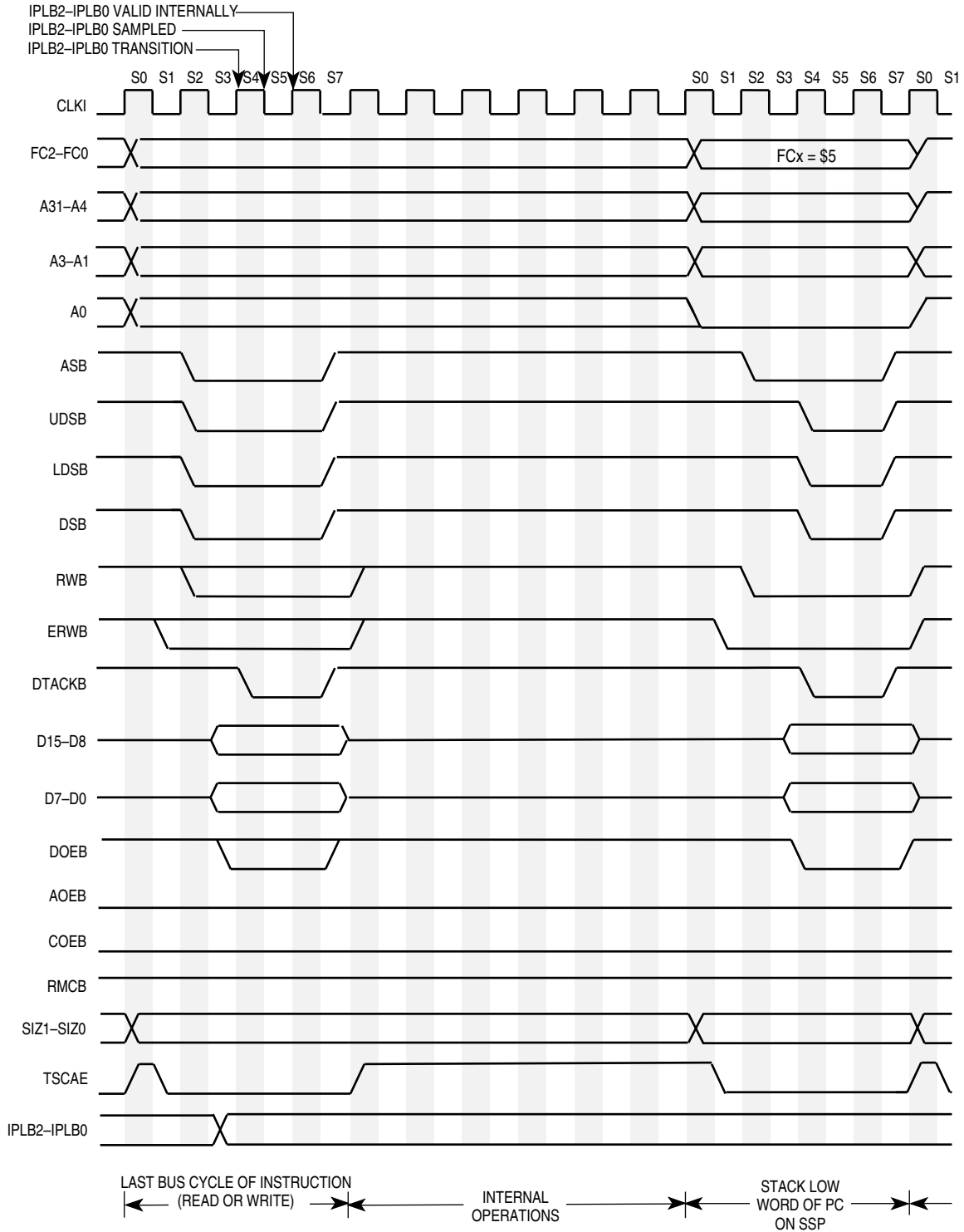
Figure 4-2. Interrupt Acknowledge Cycle Timing Diagram (Sheet 2 of 3)
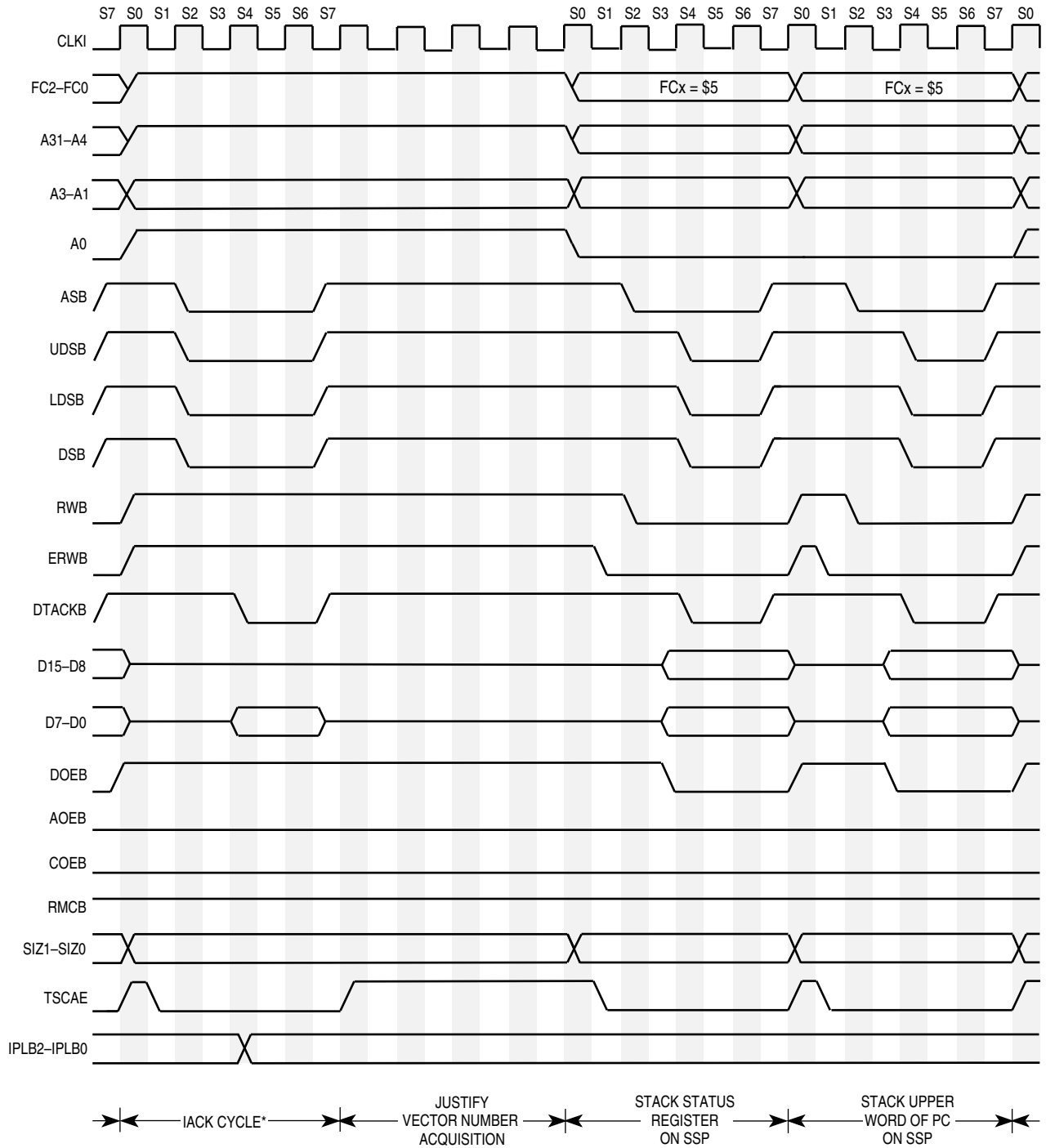
*During an IACK cycle, although a vector number is one byte, both data strobes are asserted due to the microcode used for exception processing. The processor does not recognize anything on D15–D8 at this time.

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

Freescale Semiconductor, Inc.

Figure 4-2. Interrupt Acknowledge Cycle Timing Diagram (Sheet 3 of 3)

For More Information On This Product,
Go to: www.freescale.com
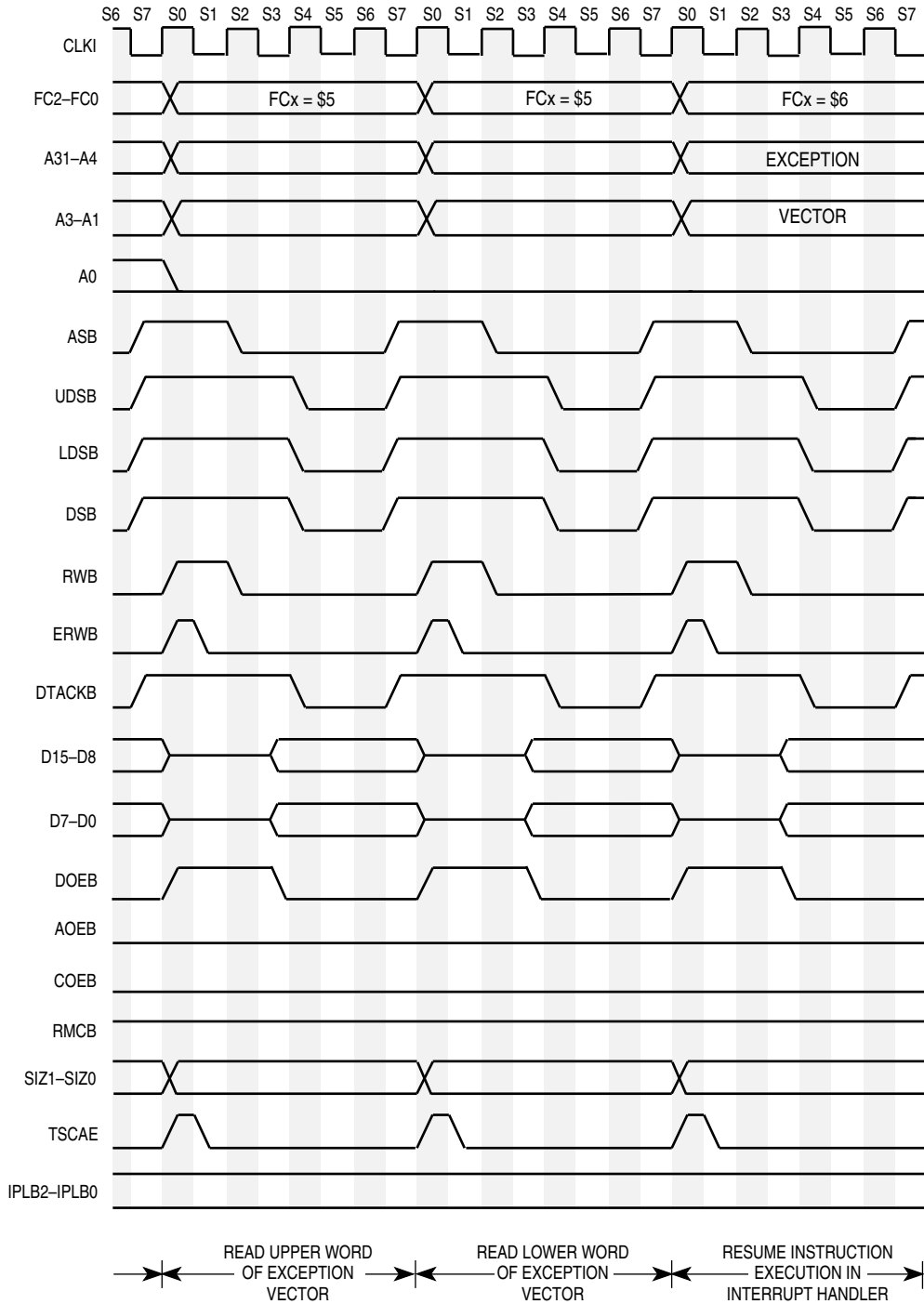
Although the timing diagram in Figure 4-2 is for the 16-bit mode, the following list describes the sequence of events executed by the processor for a vectored and autovectored interrupt in either mode.

1. Make an internal copy of the current status register.

2. In the status register, set the S bit, clear the T bit, and replace the interrupt mask with the level of the interrupt that was recognized. No bus activity occurs during the six clocks required to complete steps 1 and 2.

3. Stack the lower word of the program counter on the supervisor stack.

4. Run an interrupt acknowledge bus cycle for vector number acquisition. This step takes four clock cycles with no wait states. For an autovectored interrupt, this step takes ten to eighteen clock cycles.

5. Justify the vector number for vector acquisition. No bus activity occurs during the four clock periods that are required for this step.

6. Stack the status register that was saved in step 1 on the supervisor stack.

7. Stack the upper word of the program counter on the supervisor stack.

8. Read the upper word of the exception vector.

9. Read the lower word of the exception vector.

10. Fetch the first word of the first instruction of the interrupt handler routine.

11. Continue fetching instructions and executing as normal until the interrupt handling routine is complete.

4.1.5.2 AUTOVECTORED INTERRUPT ACKNOWLEDGE CYCLE. An interrupt acknowledge cycle can be autovectored if AVECB is asserted instead of DTACKB in state 4 (S4). For an autovectored interrupt, the vector number is internally generated to be $18 pl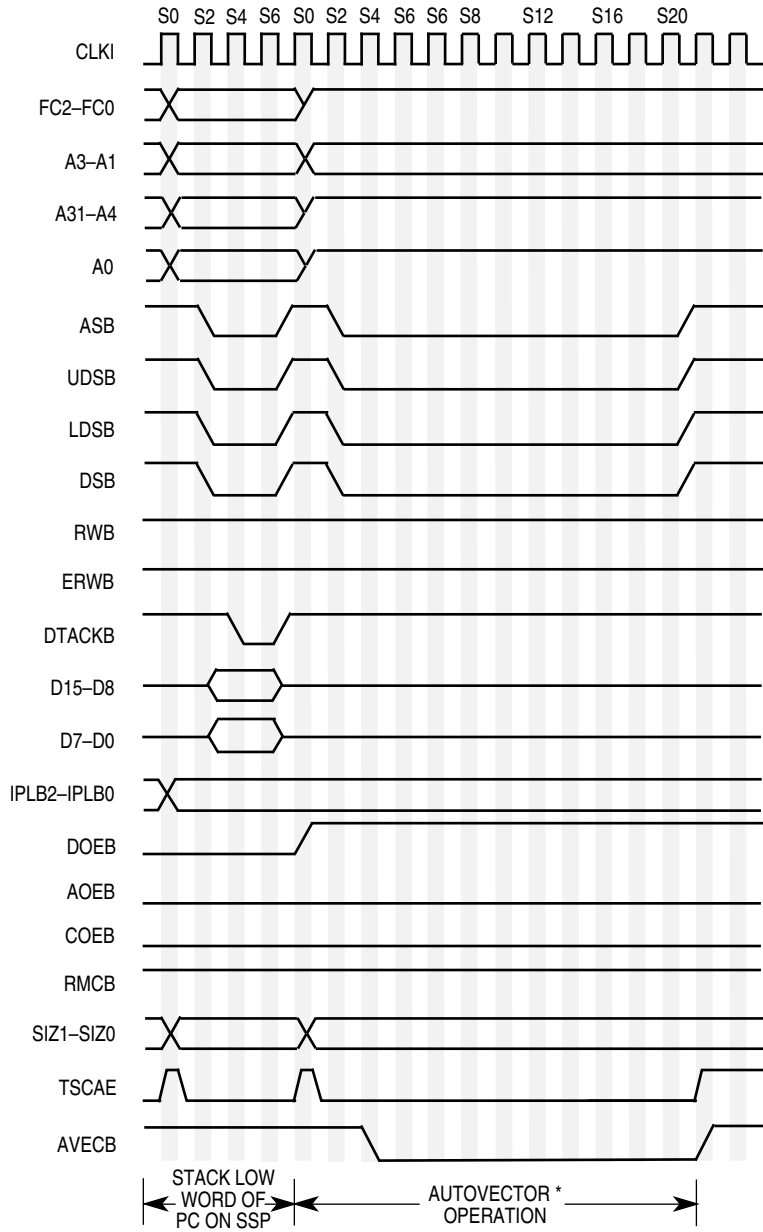us the interrupt level. The autovector capability provides vectors for each of the six maskable interrupt levels and for the nonmaskable interrupt level. The timing diagram for an autovectored interrupt acknowledge cycle is shown in Figure 4-3.

After recognizing AVECB, the processor waits until the test clock (TESTCLK) signal is low. Figure 4-4 shows the best-case timing of an autovectored interrupt acknowledge cycle, while Figure 4-5 shows the worst-case timing. The cycle length is entirely dependent on the relationship of the assertion of AVECB to the test clock.

When AVECB is recognized on the falling edge of S4 no extra wait states are inserted. The only wait states inserted are those required to synchronize AVECB with the test clock. The synchronization delay is an integral number of system clock cycles within the following extremes:

1. Best Case—the assertion of AVECB is recognized on the falling edge of the system clock that occurs three clock cycles before TESTCLK rises (or three clock cycles after TESTCLK falls).

2. Worst Case—the assertion of AVECB is recognized on the falling edge of the system clock that occurs two clock cycles before TESTCLK rises (or four clock cycles after TESTCLK falls).

Freescale Semiconductor, Inc.

S0 S2 S4 S6 S0 S2 S4 S6 S6 S8   S12   S16   S20

CLKI
FC2–FC0
A3–A1
A31–A4
A0
ASB
UDSB
LDSB
DSB
RWB
ERWB
DTACKB
D15–D8
D7–D0
IPLB2–IPLB0
DOEB
AOEB
COEB
RMCB
SIZ1–SIZ0
TSCAE
AVECB

STACK LOW
WORD OF
PC ON SSP

AUTOVECTOR *
OPERATION

NOTE: Although both UDSB and LDSB are asserted, no data is read from the bus during the
autovector cycle. The vector number is generated internally.
* An autovector operation will take between 10 and 18 clock cycles. See the best
and worst case examples on the following pages.

Figure 4-3. Autovector Operation Timing Diagram

The bus cycle ends in S7 when the SCM68000 negates the address and data strobes, and
the test clock goes low. The AVECB signal must be removed within one clock cycle after the
negation of address strobe.

Data transfer acknowledge (DTACKB) must not be asserted while AVECB is asserted. The
state machine in the processor looks for DTACKB to identify an asynchronous bus cycle and

EC000 CORE PROCESSOR USER'S MANUAL MOTOROLA
For More Information On This Product,
Go to: www.freescale.com

Figure 4-4. Autovector Operation Timing Diagram—Best Case

for AVECB to identify an autovectored interrupt acknowledge cycle. If both signals are asserted, the operation of the state machine is unpredictable.
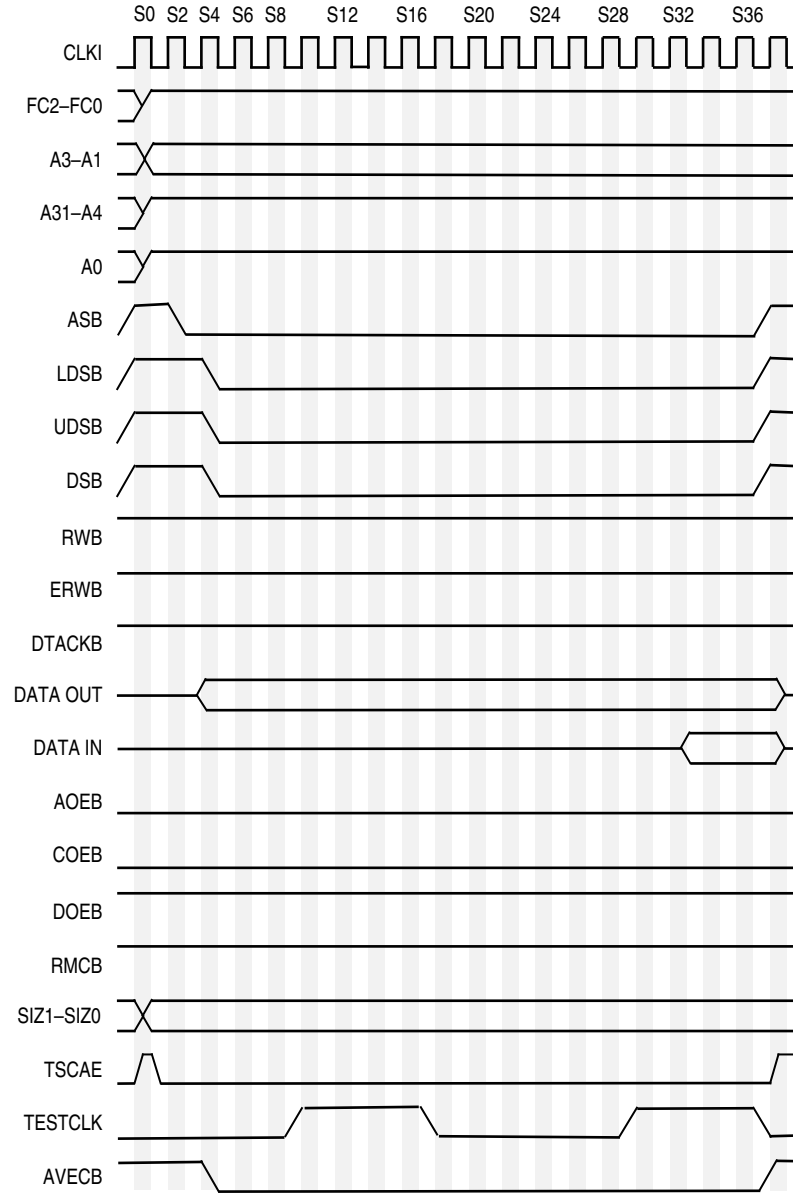
MOTOROLA   **EC000 CORE PROCESSOR USER'S MANUAL**   4-9
For More Information On This Product,
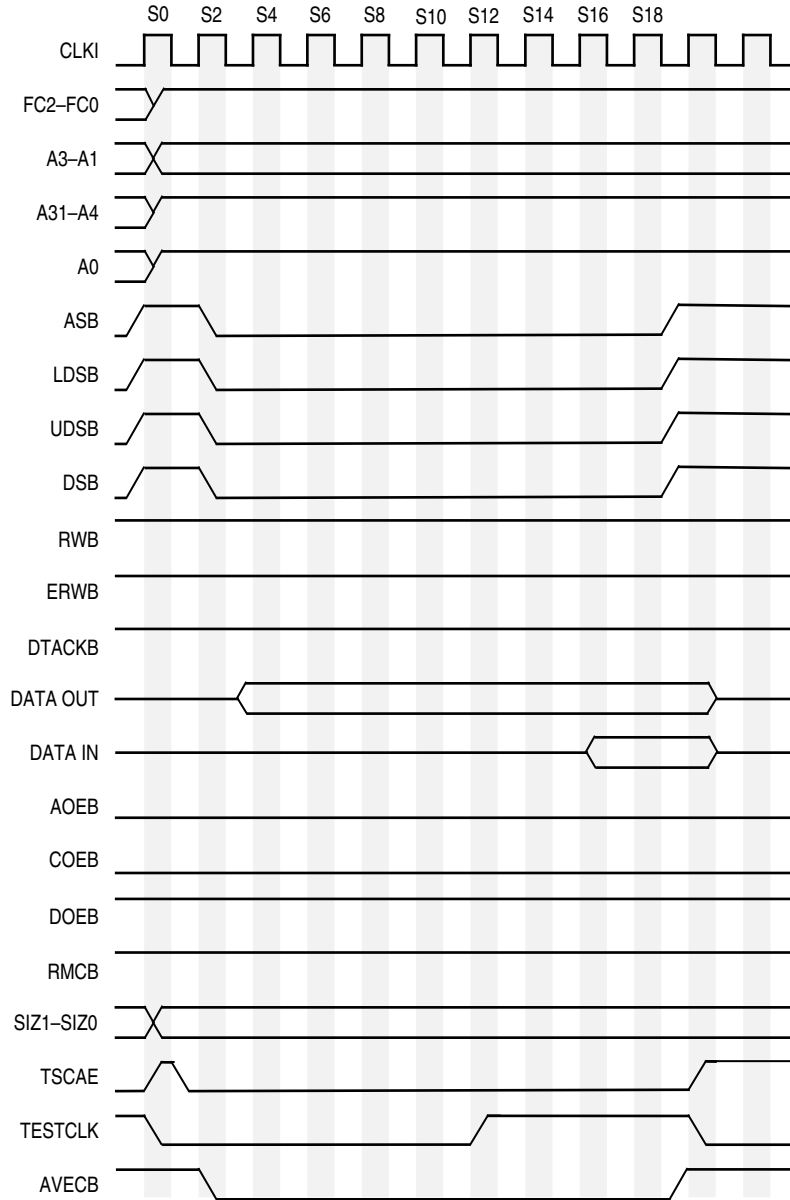**Go to: www.freescale.com**

NOTE: Although both UDSB and LDSB are asserted, no data is read from the bus during the autovector cycle. The vector number is generated internally.

Figure 4-5. Autovector Operation Timing Diagram—Worst Case

4.2 EXCEPTION PROCESSING DESCRIPTION

The processing of an exception occurs in four steps, with variations for different exception causes:

1. Make a temporary copy of the status register and set the status register for exception processing.

2. Obtain the exception vector.

3. Save the current SCM68000 context.

4. Obtain a new context and resume instruction processing.

4.2.1 Exception Vectors

An exception vector is a memory location from which the SCM68000 fetches the address of a routine to handle an exception. Each exception type requires a handler routine and a unique vector. All exception vectors are two words in length (see Figure 4-6) and reside in the supervisor data space, except for the reset vector, which is four words long and resides in the supervisor program space.

	EVEN BYTE (A0 = 0)	ODD BYTE (A0 = 1)	
WORD 0	NEW PROGRAM COUNTER (HIGH)		A1 = 0
WORD 1	NEW PROGRAM COUNTER (LOW)		A1 = 1
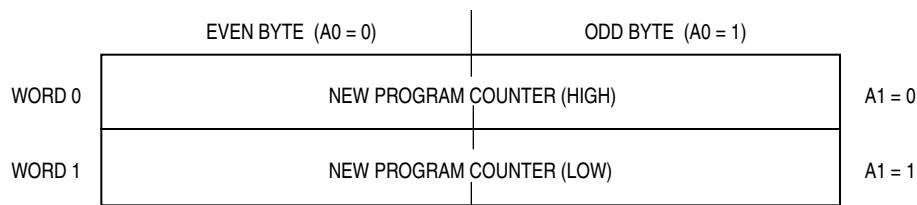
Figure 4-6. Exception Vector Format

A vector number is an 8-bit number that is multiplied by four to obtain the address of an exception vector. The SCM68000 forms the vector address by left-shifting the vector number two bit positions and zero-filling the upper-order bits to obtain a 32-bit long-word vector address (see Figure 4-7).

A31	A10	A9	A8	A7	A6	A5	A4	A3	A2	A1	A0
ALL ZEROES		V7	V6	V5	V4	V3	V2	V1	V0	0	0
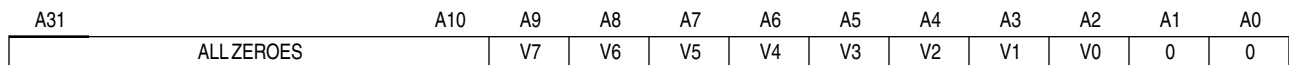
Figure 4-7. Address Translated from 8-Bit Vector Number

The vector numbers can be found in the vector table (see Table 4-2) which is 512 words long (1024 bytes), starting at address 0 and proceeding through address 1023 (decimal). The vector table provides 255 unique vectors, some of which are reserved for trap and other system function vectors. Of the 255 vectors, 192 are reserved for user interrupt vectors. However, the first 64 entries are not protected, so user interrupt vectors may overlap at the discretion of the system designer.

Table 4-2. Exception Vector Assignment

Vectors Numbers		Address		Space[5]	Assignment
Hex	Decimal	Dec	Hex		
0	0	0	000	SP	Reset: Initial SSP[2]
1	1	4	004	SP	Reset: Initial PC[2]
2	2	8	008	SD	Bus Error
3	3	12	00C	SD	Address Error
4	4	16	010	SD	Illegal Instruction
5	5	20	014	SD	Divide-by-Zero
6	6	24	018	SD	CHK Instruction
7	7	28	01C	SD	TRAPV Instruction
8	8	32	020	SD	Privilege Violation
9	9	36	024	SD	Trace
A	10	40	028	SD	Line 1010 Emulator
B	11	44	02C	SD	Line 1111 Emulator
C	12[1]	48	030	SD	(Unassigned, Reserved)
D	13[1]	52	034	SD	(Unassigned, Reserved)
E	14[1]	56	038	SD	(Unassigned, Reserved)
F	15	60	03C	SD	Uninitialized Interrupt Vector
10–17	16–23[1]	64	040	SD	(Unassigned, Reserved)
		92	05C		
18	24	96	060	SD	Spurious Interrupt[3]
19	25	100	064	SD	Level 1 Interrupt Autovector
1A	26	104	068	SD	Level 2 Interrupt Autovector
1B	27	108	06C	SD	Level 3 Interrupt Autovector
1C	28	112	070	SD	Level 4 Interrupt Autovector
1D	29	116	074	SD	Level 5 Interrupt Autovector
1E	30	120	078	SD	Level 6 Interrupt Autovector
1F	31	124	07C	SD	Level 7 Interrupt Autovector
20–2F	32–47	128	080	SD	TRAP Instruction Vectors[4]
		188	0BC		
30–3F	48–63[1]	192	0C0	SD	(Unassigned, Reserved)
		255	0FF		
40–FF	64–255	256	100	SD	User Interrupt Vectors
		1020	3FC		

NOTES:

1. Vector numbers 12–14, 16–23, and 48–63 are reserved for future enhancements by Motorola. No user peripheral devices should be assigned these numbers.

2. Reset vector (0) requires four words, unlike the other vectors which only require two words, and is located in the supervisor program space.

3. The spurious interrupt vector is taken when there is a bus error indication during interrupt processing.

4. TRAP #n uses vector number 32+ n (decimal).

5. SP denotes supervisor program space, and SD denotes supervisor data space.

4.2.2 Kinds of Exceptions

Exceptions are generated internally or externally, depending on the reason for the exception. The external exceptions are generated by interrupts, bus errors, and a reset. The interrupts are requests from peripheral devices for SCM68000 action. For interrupt requests, the peripheral must provide an 8-bit vector number on data bus lines D7-D0 (see Figure 4-8). The bus error and reset inputs are used for access control and SCM68000 restart.

D15		D8	D7	D6	D5	D4	D3	D2	D1	D0
IGNORED			V7	V6	V5	V4	V3	V2	V1	V0
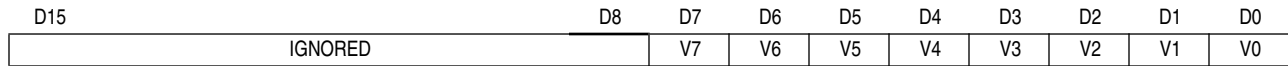
Where V7 is the MSB of the vector number and v0 is the LSB of the vector number.

Figure 4-8. Interrupt Vector Number Format

The internal exceptions are generated by instructions, address errors, or tracing. The trap (TRAP), trap on overflow (TRAPV), check register against bounds (CHK), and divide (DIV) instructions can generate exceptions as part of their instruction execution. In addition, illegal instructions, word access to odd addresses, and privilege violations initiate exceptions. Tracing is similar to a very high priority interrupt which is internally generated following each instruction.

4.2.3 Multiple Exceptions

These paragraphs describe the processing that occurs when multiple exceptions arise simultaneously. Exceptions can be grouped by their occurrence and priority. Group 0 exceptions are reset, bus error, and address error. These exceptions cause the instruction currently being executed to abort and the exception processing to commence within two clock cycles. Group 1 exceptions are trace and interrupt, privilege violations, and illegal instructions. Trace and interrupt exceptions allow the current instruction to execute to completion, but pre-empt the execution of the next instruction by forcing exception processing to occur. A privilege-violating instruction or an illegal instruction is detected when it is the next instruction to be executed. Group 2 exceptions occur as part of the normal processing of instructions. TRAP, TRAPV, CHK, and divide-by-zero exceptions are in this group. For these exceptions, normal execution of an instruction may lead to exception processing.

Group 0 exceptions have the highest priority and group 2 exceptions have the lowest priority. Within group 0, reset has the highest priority, followed by address error and then bus error. Within group 1, trace has priority over external interrupts, which in turn takes priority over illegal instruction and privilege violation. Since only one instruction can be executed at a time, no priority relationship applies within group 2.

The priority relationship between two exceptions determines which is taken, or taken first, if the conditions for both arise simultaneously. Therefore, if a bus error occurs during a TRAP instruction, the bus error takes precedence, and TRAP instruction processing is aborted. In another example, if an interrupt request occurs during the execution of an instruction while the T-bit is asserted, the trace exception has priority and is processed first. Before instruction execution resumes, however, the interrupt exception is also processed, and instruction

processing finally commences in the interrupt handler routine. A summary of exception grouping and priority is given in Table 4-3.

Table 4-3. Exception Grouping and Priority

Group	Exception	Processing
0	Reset Address Error Bus Error	Exception Processing Begins as Soon as the Bus Cycle Is Terminated
1	Trace Interrupt Illegal Privilege	Exception Processing Begins Before the Next Instruction
2	TRAP, TRAPV CHK Divide-by-Zero	Exception Processing Is Started by Normal Instruction Execution

As an example, consider trap, trace, and interrupt exceptions that occurred simultaneously and are pending. The exception processing for the trap occurs first, followed immediately by exception processing for the trace, and then for the interrupt. When the SCM68000 resumes normal instruction execution, it is in the interrupt handler, which returns to the trace handler, which returns to the trap execution handler. The reset exception handler is always executed first because it clears all other exceptions.

4.2.4 Exception Stack Frames

Exception processing saves the most volatile portion of the current SCM68000 context on the top of the supervisor stack. This context is organized in a format called the exception stack frame. Although this information varies with type of exception, it always includes the status register and program counter of the SCM68000 when the exception occurred.

The amount and type of information saved on the stack are determined by the exception type. Exceptions are grouped by type according to priority of the exception.

Of the group 0 exceptions, the reset exception does not stack any information. The information stacked by a bus error or address error exception is described in **4.3.7 Bus Error** and is shown in Figure 4-14.

The groups 1 and 2 exception stack frame is shown in Figure 4-9. Only the program counter and status register are saved. The program counter points to the next instruction to be executed after exception processing.

4.2.5 Exception Processing Sequence

In the first step of exception processing, an internal copy is made of the status register. After the copy is made, the S-bit of the status register is set, putting the SCM68000 into the supervisor mode. Also, the T-bit is cleared, which allows the exception handler to execute unhindered by tracing. For the reset and interrupt exceptions, the interrupt priority mask is also updated appropriately.

In the second step, the vector number of the exception is determined. For interrupts, the vector number is obtained by an SCM68000 bus cycle classified as an interrupt acknowledge
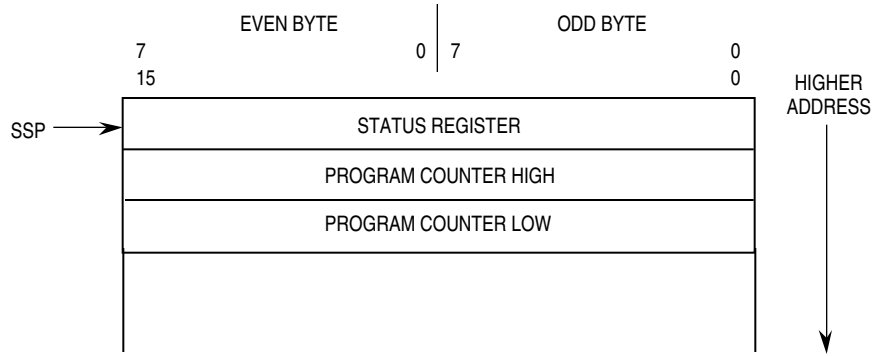
```
            EVEN BYTE              ODD BYTE
        7                 0   7                 0
        15                                      0    HIGHER
                                                     ADDRESS
  SSP →         STATUS REGISTER
             PROGRAM COUNTER HIGH
             PROGRAM COUNTER LOW

```

Figure 4-9. Groups 1 and 2 Exception Stack Frame

cycle. For all other exceptions, internal logic provides the vector number. This vector number is then used to calculate the address of the exception vector.

The third step, except for the reset exception, is to save the current SCM68000 status. (The reset exception does not save the context and skips this step.) The current program counter value and the saved copy of the status register are stacked using the SSP. The stacked program counter value usually points to the next unexecuted instruction. However, for bus error and address error, the value stacked for the program counter is unpredictable and may be incremented from the address of the instruction that caused the error. Group 1 and 2 exceptions use a short format exception stack frame. Additional information defining the current context is stacked for the bus error and address error exceptions.

The last step is the same for all exceptions. The new program counter value is fetched from the exception vector. The SCM68000 then resumes instruction execution at the address provided by the exception vector, and normal instruction decoding and execution is started.

4.3 PROCESSING OF SPECIFIC EXCEPTIONS

The exceptions are classified according to their sources, and each type is processed differently. The following paragraphs describe in detail the types of exceptions and the processing of each type.

4.3.1 Reset

The reset exception corresponds to the highest exception level. The processing of the reset exception is performed for system initiation and recovery from catastrophic failure. If the SCM68000 is currently executing a bus cycle, it will start processing at state 5 (S5) immediately after the internal reset signal is valid. The bus cycle will end at state 7 (S7) and the current instruction being executed will be canceled. The SCM68000 is forced into the supervisor state, and the trace state is forced off. The interrupt priority mask is set to level seven. The vector number is internally generated to reference the reset exception vector at location zero in the supervisor program space. Because no assumptions can be made about the validity of register contents, in particular the SSP, neither the program counter nor the status register is saved. The address in the first two words of the reset exception vector is fetched as the initial SSP, and the address in the last two words of the reset exception vector is

fetched as the initial program counter. Finally, instruction execution is started at the address in the program counter. The initial program counter should point to the power-up/restart code.

The RESET instruction does not cause a reset exception; it asserts the RESETOB signal to reset external devices, which allows the software to reset the system to a known state and continue processing with the next instruction.

4.3.1.1 RESET OPERATION. The SCM68000 has an input reset signal (RESETIB) and an output reset signal (RESETOB). In many cases, these two signals are connected to form just one bi-directional RESETB signal. One way of combining these signals using cells from the Motorola standard cell library is illustrated in Figure 4-10. The SCM68000 and it's external circuitry can be reset in three ways: asserting RESETIB and HALTIB simultaneously, asserting only RESETIB, and using the RESET instruction. The methods to reset the SCM68000 and it's external devices are described in the following paragraphs.
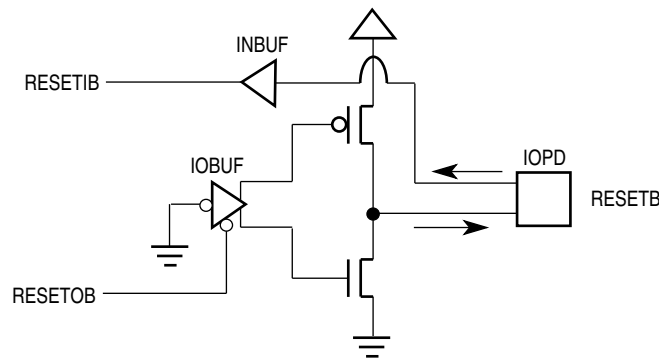
Figure 4-10. Reset Circuit

4.3.1.1.1 Reset Using RESETIB and HALTIB. For the initial reset, RESETIB and HALTIB must be asserted by an external device for at least 132 clock periods. Resetting the SCM68000 initializes the internal state. The SCM68000 reads the reset vector table entry (address $00000) and loads the contents into the SSP. Next, the SCM68000 loads the contents of address $00004 (vector table entry 1) into the program counter. Then the SCM68000 initializes the status registers by setting the supervisor state, clearing the trace mode bit, and setting the interrupt mask level to seven. No other register is affected by the reset sequence. The internal circuitry may cause the SCM68000 to take up to four clock periods to recognize that a reset is taking place. The timings for the intial reset operation are shown in Figure 4-11.

Subsequent resets can be accomplished by asserting RESETIB and HALTIB simultaneously for ten or more clock periods.

4.3.1.1.2 Reset Instruction. The RESET instruction causes the SCM68000 to assert RESETOB for 124 clock periods to reset the external devices of the system. The internal state of the SCM68000 is not affected. Neither the status register nor any of the internal
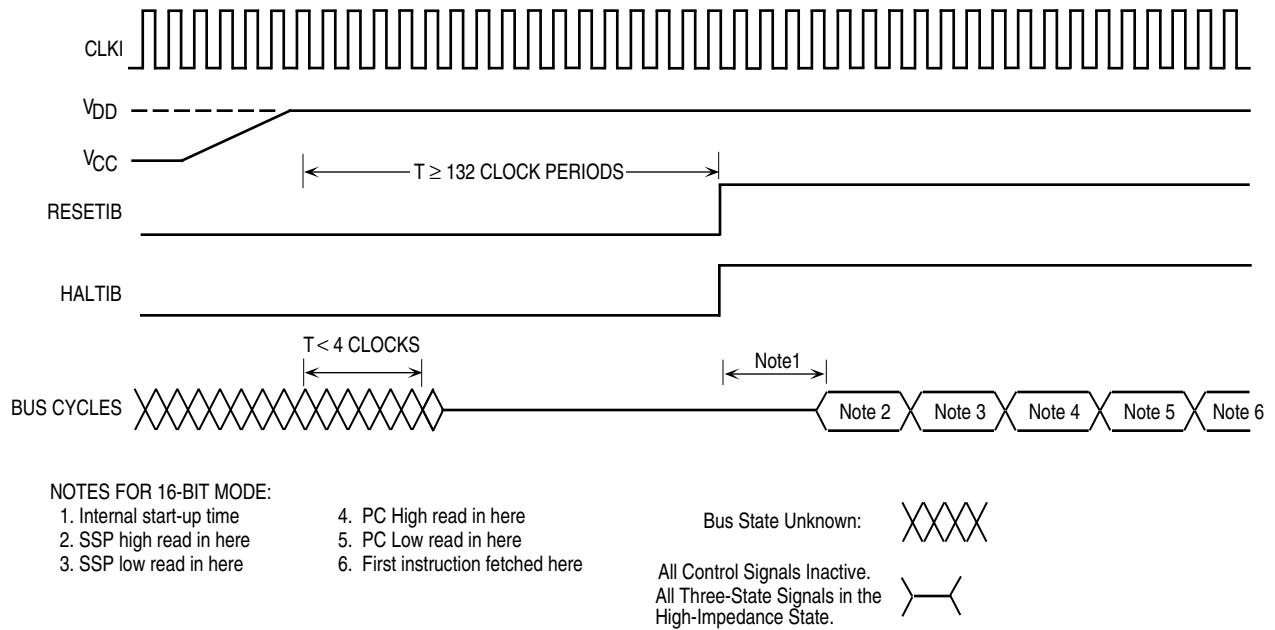
Figure 4-11. Reset Operation Timing Diagram

registers are affected by a RESET instruction. Figure 4-12 shows a timing diagram for RESETOB.

NOTE

The user must ensure that all external devices are reset at the completion of the RESET instruction. The RESETOB signal is negated on the rising edge of S0 of the next bus cycle.

If the RESETIB and RESETOB signals are connected as shown in Figure 4-10, the HALTIB signal should not be asserted during the RESET instruction. When the SCM68000 recognizes that both the HALTIB and RESETIB signals are asserted, it will begin the reset exception and negate RESETOB which will cause RESETIB to be negated. Since RESETIB would not be asserted for at least ten clock periods with HALTIB, a partial reset will occur and may drive the SCM68000 into an unknown state.

4.3.1.1.3 Reset Using Only RESETIB. The SCM68000 must initially be reset using the RESETIB and HALTIB signals as previously described. However, subsequent resets can also be accomplished by asserting only the RESETIB signal for a minimum of 132 clock periods.

Because an assertion of the RESETIB signal is ignored while the RESETOB signal is asserted, the two signals can be connected as shown in Figure 4-10. Since the core may be executing a RESET instruction at the time it is being reset using only the RESETIB signal, the RESETIB signal should be asserted for a minimum of 132 clock periods. This will ensure that the RESETOB signal has been negated and the RESETIB signal will be recognized.
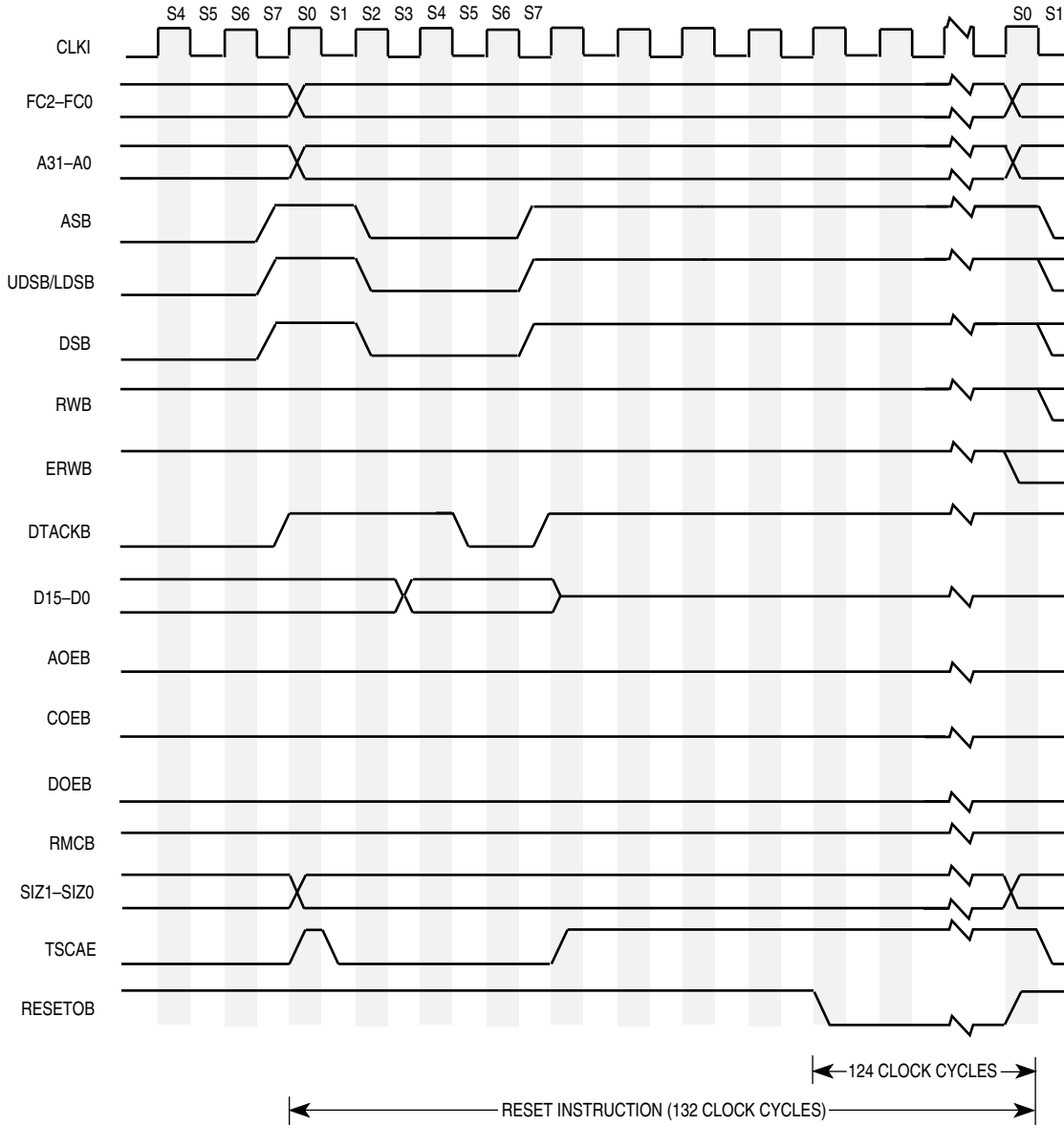
Figure 4-12. RESETOB Timing Diagram

4.3.1.2 INITIALIZING THE SCM68000 FOR SIMULATION. To simulate the SCM68000 properly, the initialization procedure differs slightly from the actual part. This difference is necessary to correctly start the test clock (TESTCLK). Both HALTIB and RESETIB must be asserted for at least 10 clock cycles. The TEST signal must be asserted for at least two clock cycles while HALTIB and RESETIB are asserted and must be negated at least two clock cycles before HALTIB and RESETIB are negated. Figure 4-13 shows the timing for initializing the SCM68000 for simulation.

If the SCM68000 has been properly reset during simulation, the TESTCLK signal will pulse with a period equal to ten SCM68000 clock periods. The period will consist of six SCM68000 clock periods in the low position and four SCM68000 clock periods in the high position.
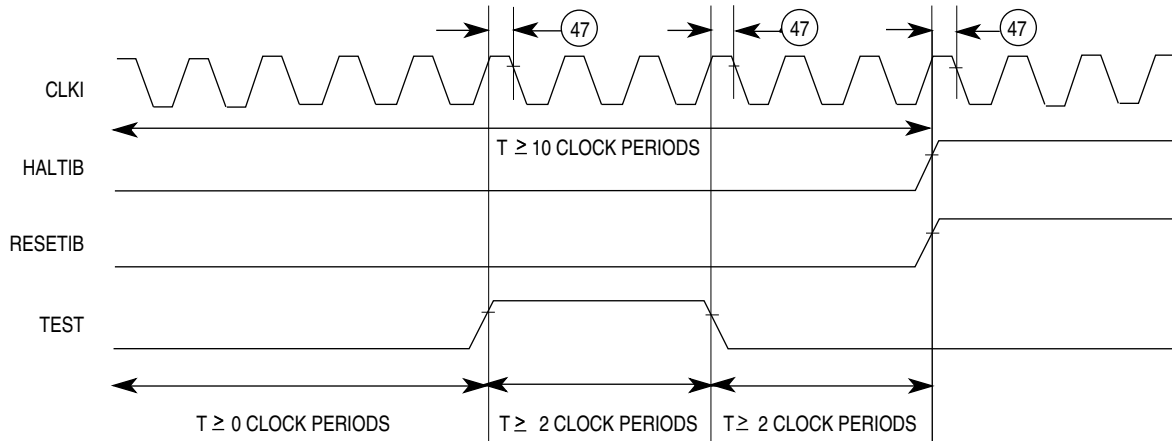
Figure 4-13. Initialization of the SCM68000 for Simulation Timing Diagram

4.3.2 Interrupts

Seven levels of interrupt priorities are provided, numbered 1–7. Devices can be chained externally within interrupt priority levels, allowing an unlimited number of peripheral devices to interrupt the SCM68000. The status register contains a 3-bit mask indicating the current interrupt priority, and interrupts are inhibited for all priority levels less than or equal to the current priority.

An interrupt request is made to the SCM68000 by encoding the interrupt request levels 1–7 on the three interrupt request lines (IPLB2–IPLB0); all negated lines indicate no interrupt request. Interrupt requests arriving at the SCM68000 do not force immediate exception processing, but the requests are made pending. Pending interrupts are detected between instruction executions. If the priority of the pending interrupt is lower than or equal to the current SCM68000 priority, execution continues with the next instruction, and the requesting interrupt is postponed until the priority of the pending interrupt becomes greater than the current SCM68000 priority.

If the priority of the pending interrupt is greater than the current SCM68000 priority, the exception processing sequence for the requesting interrupt is started. A copy of the status register is saved; the privilege mode is set to supervisor mode; tracing is suppressed; and the SCM68000 priority level is set to the level of the interrupt being acknowledged. The SCM68000 fetches the vector number from the interrupting device by executing an interrupt acknowledge cycle, which displays the level number of the interrupt being acknowledged on the address bus. If external logic requests an automatic vector, the SCM68000 internally generates a vector number corresponding to the interrupt level number. If external logic indicates a bus error, the interrupt is considered spurious, and the generated vector number references the spurious interrupt vector. The SCM68000 then proceeds with the usual exception processing, saving the program counter and status register on the supervisor stack. The saved value of the program counter is the address of the instruction that would have been executed had the interrupt not been taken. The appropriate interrupt vector is fetched and loaded into the program counter, and normal instruction execution commences in the interrupt handling routine.

Interrupt requests should be maintained on the interrupt control signals IPLB2–IPLB0 until the interrupt acknowledge bus cycle is initiated to guarantee that the interrupt will be recognized.

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

Level seven interrupts are edge triggered by a transition from a lower priority request to the level seven request, as opposed to interrupt levels one through six, which are level sensitive. Therefore, if IPLB2-IPLB0 remain at level seven, the SCM68000 will only recognize one level seven interrupt since only one transition from a lower level request to a level seven request occurred. For the processor to recognize a level seven interrupt followed by another level seven interrupt, one of the two following sequences must occur:

1. The interrupt request level on the interrupt control pins changes from a lower request level to level seven and remains at level seven until the interrupt acknowledge bus cycle begins. Later, the interrupt request level returns to a lower interrupt request level and then back to level seven, causing a second transition on the interrupt control lines.

2. The interrupt request level on the interrupt control pins changes from a lower request level to level seven and remains at level seven. If the interrupt handling routine for the level seven interrupt lowers the interrupt mask level, a second level seven interrupt will be recognized even though no transition has occurred on the interrupt control pins. After the level seven interrupt handling routine completes, the SCM68000 will compare the interrupt mask level to the interrupt request level on IPLB2-IPLB0. Since the interrupt mask level will be lower than the requested level, the interrupt mask will be set back to level seven. The level seven request on IPLB2-IPLB0 must be held until the second interrupt acknowledge bus cycle has begun to insure that the interrupt is recognized.

For more information on SCM68000 interrupts, see the application note *A Discussion of Interrupts for the MC68000* (AN1012).

4.3.2.2 UNINITIALIZED INTERRUPT. Under normal conditions, an interrupting device provides the SCM68000 with an interrupt vector number and asserts data transfer acknowledge (DTACKB), or asserts autovector (AVECB), or bus error (BERRB) during an interrupt acknowledge cycle by the SCM68000. If the interrupting M68000 Family peripheral has not been initialized, it will provide the uninitialized interrupt vector number ($0F). This response conforms to a uniform way to recover from a programming error.

4.3.2.3 SPURIOUS INTERRUPT. During the interrupt acknowledge cycle, if no device responds by asserting DTACKB or AVECB, BERRB should be asserted to terminate the vector acquisition. The SCM68000 separates the processing of this error from bus error by forming a short format exception stack and fetching the spurious interrupt vector instead of the bus error vector. The SCM68000 then proceeds with the usual exception processing.

4.3.3 Instruction Traps

Traps are exceptions caused by instructions. Traps occur when the SCM68000 recognizes an abnormal condition during instruction execution or when an instruction that normally causes exception processing is executed.

Exception processing for traps is straightforward. The status register is copied; the supervisor mode is entered; and tracing is turned off. The vector number is internally generated; for the TRAP instruction, part of the vector number comes from the instruction itself. The program counter and the copy of the status register are saved on the supervisor stack. The saved value of the program counter is the address of the instruction following the instruction that generated the trap. Finally, instruction execution commences at the address in the exception vector.

Some instructions are used specifically to generate traps. The TRAP instruction always forces an exception and is useful for implementing system calls for user programs. The TRAPV and CHK instructions force an exception if the user program detects a run-time error, which may be an arithmetic overflow or a subscript out of bounds.

A signed divide (DIVS) or unsigned divide (DIVU) instruction forces an exception if a division operation is attempted with a divisor of zero.

4.3.4 Illegal and Unimplemented Instructions

Illegal instruction is the term used to refer to any of the word bit patterns that do not match a legal SCM68000 instruction. If such an instruction is fetched, an illegal instruction exception occurs. Motorola reserves the right to define instructions using the opcodes of any of the illegal instructions. Three bit patterns always force an illegal instruction trap: $4AFA, $4AFB, and $4AFC. Two of the patterns, $4AFA and $4AFB, are reserved for Motorola system products. The third pattern, $4AFC, is reserved for customer use (as the take illegal instruction trap (ILLEGAL) instruction).

Word patterns with bits 15–12 equaling 1010 or 1111 are distinguished as unimplemented instructions, and separate exception vectors are assigned to these patterns to permit efficient emulation. These vectors allow the operating system to emulate unimplemented instructions in software.

Exception processing for illegal instructions is similar to that of traps. After the instruction is fetched and decoding is attempted, the SCM68000 determines that execution of an illegal instruction is being attempted and starts exception processing. The exception stack frame for group 2 (see Table 4-3 for more information on exception groups) is then pushed on the supervisor stack, and the illegal instruction vector is fetched.

4.3.5 Privilege Violations

To provide system security, various instructions are privileged. An attempt to execute one of the privileged instructions while in the user mode causes an exception. The privileged instructions can be found in Chapter 6 of the *M68000 Family Programmer's Reference Manual* (M68000PM/AD).

Exception processing for privilege violations is similar to that of illegal instructions. After the instruction is fetched and decoded and the SCM68000 determines that a privilege violation is being attempted, the SCM68000 starts exception processing. The status register is copied; the supervisor mode is entered; and tracing is turned off. The vector number is generated to reference the privilege violation vector, and the current program counter and the copy of the status register are saved on the supervisor stack. Finally, instruction execution commences at the address in the privilege violation exception vector.

4.3.6 Tracing

To aid in program development, the M68000 Family includes a facility to allow tracing following each instruction. When tracing is enabled, an exception is forced after each instruction is executed. Thus, a debugging program can monitor the execution of the program under test.

The trace function is controlled by the T-bit in the supervisor portion of the status register. If the T-bit is cleared (off), tracing is disabled and instruction execution proceeds normally. If the T-bit is set (on) at the beginning of the execution of an instruction, a trace exception is generated after the instruction is completed. If the instruction is not executed because an interrupt is taken or because the instruction is illegal or privileged, the trace exception does not occur. The trace exception also does not occur if the instruction is aborted by the reset, bus error, or address error exceptions. If the instruction is executed and an interrupt is pending upon completion, the trace exception is processed before the interrupt exception. During the execution of the instruction, if an exception is forced by that instruction, the exception processing for the instruction exception occurs before that of the trace exception.

As an extreme illustration of these rules, consider the arrival of an interrupt during the execution of a TRAP instruction while tracing is enabled. First, the trap exception is processed, then the trace exception, and finally, the interrupt exception. Instruction execution resumes in the interrupt handler routine.

After the execution of the instruction is complete and before the start of the next instruction, exception processing for a trace begins. A copy is made of the status register. The transition to supervisor mode is made, and the T-bit of the status register is turned off, disabling further tracing. The vector number is generated to reference the trace exception vector, and the current program counter and the copy of the status register are saved on the supervisor stack. The saved value of the program counter is the address of the next instruction. Instruction execution commences at the address contained in the trace exception vector.

4.3.7 Bus Error

A bus error exception is requested by external logic. The current bus cycle is terminated. The current SCM68000 activity, whether instruction or exception processing, is terminated, and the SCM68000 immediately begins exception processing.

Exception processing for a bus error follows the usual sequence of steps. The status register is copied, the supervisor mode is entered, and tracing is turned off. The vector number is generated to refer to the bus error vector. Since the SCM68000 is fetching the instruction or an operand when the error occurs, the context of the SCM68000 is more detailed. To save

more of this context, additional information is saved on the supervisor stack. The program counter and the copy of the status register are saved. The value saved for the program counter is advanced 2–10 bytes beyond the address of the first word of the instruction that made the reference causing the bus error. If the bus error occurred during the fetch of the next instruction, the saved program counter has a value in the vicinity of the current instruction, even if the current instruction is a branch, a jump, or a return instruction. In addition to the usual information, the SCM68000 saves its internal copy of the first word of the instruction being processed and the address being accessed by the aborted bus cycle. Specific information about the access is also saved: type of access (read or write), SCM68000 activity (processing an instruction), and function code outputs when the bus error occurred. The SCM68000 is processing an instruction if it is in the normal state or processing a group 2 exception; the SCM68000 is not processing an instruction if it is processing a group 0 or a group 1 exception (see Table 4-3 for more information on exception groups). Figure 4-14 illustrates how this information is organized on the supervisor stack. If a bus error occurs during the last step of exception processing, while either reading the exception vector or fetching the instruction, the value of the program counter is the address of the exception vector. Although this information is not generally sufficient to fully recover from the bus error, it does allow software diagnosis. Finally, the SCM68000 commences instruction processing at the address in the vector. It is the responsibility of the error handler routine to clean up the stack and determine where to continue execution.

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
LOWER ADDRESS												R/W	I/N	FUNCTION CODE		
ACCESS ADDRESS								HIGH								
								LOW								
							INSTRUCTION REGISTER									
							STATUS REGISTER									
PROGRAM COUNTER								HIGH								
								LOW								
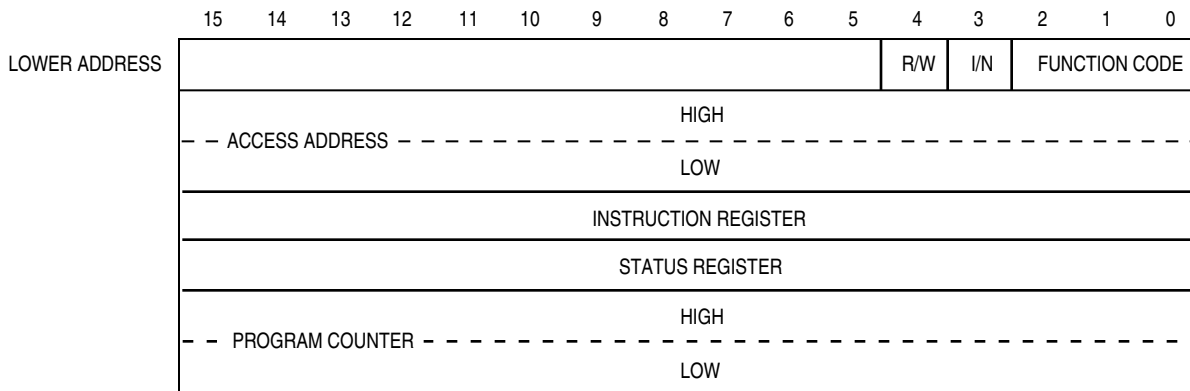
R/W (Read/Write): Write = 0, Read = 1. I/N (Instruction/Not): Instruction = 0, Not = 1

Figure 4-14. Supervisor Stack Order for Bus or Address Error Exception

If a bus error or address error occurs during the exception processing for a bus error, an address error, or a reset, the SCM68000 halts and all processing ceases. This halt simplifies the detection of a catastrophic system failure, since the SCM68000 removes itself from the system to protect memory contents from erroneous accesses. Only an external reset operation can restart a halted SCM68000.

4.3.8 Address Error

An address error exception occurs when the SCM68000 attempts to access a word or long-word operand or an instruction at an odd address. An address error is similar to an internally generated bus error. The bus cycle is aborted, and the SCM68000 ceases current process-

ing and begins exception processing. The exception processing sequence is the same as that of a bus error, including the information to be stacked, except the vector number refers to the address error vector. Likewise, if an address error occurs during the exception processing for a bus error, address error, or reset, the SCM68000 is halted.

SECTION 5
8-BIT INSTRUCTION EXECUTION TIMES

This section contains listings of the instruction execution times in terms of external clock (CLKI) periods for the SCM68000 (EC000 core)[1] in 8-bit mode. In this data, it is assumed that both memory read and write cycles consist of four clock periods. A longer memory cycle causes the generation of wait states that must be added to the total instruction times.

The number of bus read and write cycles for each instruction is also included with the timing data. This data is shown as

$$n(r/w)$$

where:

   n is the total number of clock periods
   r is the number of read cycles
   w is the number of write cycles

For example, a timing number shown as 18(3/1) means that 18 clock periods are required to execute the instruction. Of the 18 clock periods, 12 are used for the three read cycles (four periods per cycle). Four additional clock periods are used for the single write cycle, for a total of 16 clock periods. The bus is idle for two clock periods during which the processor completes the internal operations required for the instruction.

NOTE

The total number of clock periods (n) includes instruction fetch and all applicable operand fetches and stores.

5.1 OPERAND EFFECTIVE ADDRESS CALCULATION TIMES

Table 5-1 lists the numbers of clock periods required to compute the effective addresses for instructions. The totals include fetching any extension words, computing the address, and fetching the memory operand. The total number of clock periods, the number of read cycles, and the number of write cycles (zero for all effective address calculations) are shown in the previously described format.

---

[1] The SCM68000 is the name of the Verilog model for the EC000 core. The remainder of this section will refer to the part as only the SCM68000.

Table 5-1. Effective Address Calculation Times

Addressing Mode		Byte	Word	Long
Register				
Dn	Data Register Direct	**0**(0/0)	**0**(0/0)	**0**(0/0)
An	Address Register Direct	**0**(0/0)	**0**(0/0)	**0**(0/0)
Memory				
(An)	Address Register Indirect	**4**(1/0)	**8**(2/0)	**16**(4/0)
(An)+	Address Register Indirect with Postincrement	**4**(1/0)	**8**(2/0)	**16**(4/0)
−(An)	Address Register Indirect with Predecrement	**6**(1/0)	**10**(2/0)	**18**(4/0)
(d_{16}, An)	Address Register Indirect with Displacement	**12**(3/0)	**16**(4/0)	**24**(6/0)
$(d_8, An, Xn)^*$	Address Register Indirect with Index	**14**(3/0)	**18**(4/0)	**26**(6/0)
(xxx).W	Absolute Short	**12**(3/0)	**16**(4/0)	**24**(6/0)
(xxx).L	Absolute Long	**20**(5/0)	**24**(6/0)	**32**(8/0)
(d_{16}, PC)	Program Counter Indirect with Displacement	**12**(3/0)	**16**(3/0)	**24**(6/0)
$(d_8, PC, Xn)^*$	Program Counter Indirect with Index	**14**(3/0)	**18**(4/0)	**26**(6/0)
#<data>	Immediate	**8**(2/0)	**8**(2/0)	**16**(4/0)

*The size of the index register (Xn) does not affect execution time.

5.2 MOVE INSTRUCTION EXECUTION TIMES

Table 5-2, Table 5-3, and Table 5-4 list the numbers of clock periods required for the move instructions. The totals include instruction fetch, operand reads, and operand writes. The total number of clock periods, the number of read cycles, and the number of write cycles are shown in the previously described format.

Table 5-2. Move Byte Instruction Execution Times

Source	Destination								
	Dn	An	(An)	(An)+	−(An)	(d_{16}, An)	$(d_8, An, Xn)^*$	(xxx).W	(xxx).L
Dn	**8**(2/0)	**8**(2/0)	**12**(2/1)	**12**(2/1)	**12**(2/1)	**20**(4/1)	**22**(4/1)	**20**(4/1)	**28**(6/1)
(An)	**12**(3/0)	**12**(3/0)	**16**(3/1)	**16**(3/1)	**16**(3/1)	**24**(5/1)	**26**(5/1)	**24**(5/1)	**32**(7/1)
(An)+	**12**(3/0)	**12**(3/0)	**16**(3/1)	**16**(3/1)	**16**(3/1)	**24**(5/1)	**26**(5/1)	**24**(5/1)	**32**(7/1)
−(An)	**14**(3/0)	**14**(3/0)	**18**(3/1)	**18**(3/1)	**18**(3/1)	**26**(5/1)	**28**(5/1)	**26**(5/1)	**34**(7/1)
(d_{16}, An)	**20**(5/0)	**20**(5/0)	**24**(5/1)	**24**(5/1)	**24**(5/1)	**32**(7/1)	**34**(7/1)	**32**(7/1)	**40**(9/1)
$(d_8, An, Xn)^*$	**22**(5/0)	**22**(5/0)	**26**(5/1)	**26**(5/1)	**26**(5/1)	**34**(7/1)	**36**(7/1)	**34**(7/1)	**42**(9/1)
(xxx).W	**20**(5/0)	**20**(5/0)	**24**(5/1)	**24**(5/1)	**24**(5/1)	**32**(7/1)	**34**(7/1)	**32**(7/1)	**40**(9/1)
(xxx).L	**28**(7/0)	**28**(7/0)	**32**(7/1)	**32**(7/1)	**32**(7/1)	**40**(9/1)	**42**(9/1)	**40**(9/1)	**48**(11/1)
(d_{16}, PC)	**20**(5/0)	**20**(5/0)	**24**(5/1)	**24**(5/1)	**24**(5/1)	**32**(7/1)	**34**(7/1)	**32**(7/1)	**40**(9/1)
$(d_8, PC, Xn)^*$	**22**(5/0)	**22**(5/0)	**26**(5/1)	**26**(5/1)	**26**(5/1)	**34**(7/1)	**36**(7/1)	**34**(7/1)	**42**(9/1)
#<data>	**16**(4/0)	**16**(4/0)	**20**(4/1)	**20**(4/1)	**20**(4/1)	**28**(6/1)	**30**(6/1)	**28**(6/1)	**36**(8/1)

*The size of the index register (Xn) does not affect execution time.

Table 5-3. Move Word Instruction Execution Times

Source	Destination								
	Dn	**An**	**(An)**	**(An)+**	**–(An)**	**(d$_{16}$, An)**	**(d$_8$, An, Xn)***	**(xxx).W**	**(xxx).L**
Dn	**8**(2/0)	**8**(2/0)	**16**(2/2)	**16**(2/2)	**16**(2/2)	**24**(4/2)	**26**(4/2)	**24**(4/2)	**32**(6/2)
An	**8**(2/0)	**8**(2/0)	**16**(2/2)	**16**(2/2)	**16**(2/2)	**24**(4/2)	**26**(4/2)	**24**(4/2)	**32**(6/2)
(An)	**16**(4/0)	**16**(4/0)	**24**(4/2)	**24**(4/2)	**24**(4/2)	**32**(6/2)	**34**(6/2)	**32**(6/2)	**40**(8/2)
(An)+	**16**(4/0)	**16**(4/0)	**24**(4/2)	**24**(4/2)	**24**(4/2)	**32**(6/2)	**34**(6/2)	**32**(6/2)	**40**(8/2)
–(An)	**18**(4/0)	**18**(4/0)	**26**(4/2)	**26**(4/2)	**26**(4/2)	**34**(6/2)	**32**(6/2)	**34**(6/2)	**42**(8/2)
(d$_{16}$, An)	**24**(6/0)	**24**(6/0)	**32**(6/2)	**32**(6/2)	**32**(6/2)	**40**(8/2)	**42**(8/2)	**40**(8/2)	**48**(10/2)
(d$_8$, An, Xn)*	**26**(6/0)	**26**(6/0)	**34**(6/2)	**34**(6/2)	**34**(6/2)	**42**(8/2)	**44**(8/2)	**42**(8/2)	**50**(10/2)
(xxx).W	**24**(6/0)	**24**(6/0)	**32**(6/2)	**32**(6/2)	**32**(6/2)	**40**(8/2)	**42**(8/2)	**40**(8/2)	**48**(10/2)
(xxx).L	**32**(8/0)	**32**(8/0)	**40**(8/2)	**40**(8/2)	**40**(8/2)	**48**(10/2)	**50**(10/2)	**48**(10/2)	**56**(12/2)
(d$_{16}$, PC)	**24**(6/0)	**24**(6/0)	**32**(6/2)	**32**(6/2)	**32**(6/2)	**40**(8/2)	**42**(8/2)	**40**(8/2)	**48**(10/2)
(d$_8$, PC, Xn)*	**26**(6/0)	**26**(6/0)	**34**(6/2)	**34**(6/2)	**34**(6/2)	**42**(8/2)	**44**(8/2)	**42**(8/2)	**50**(10/2)
#<data>	**16**(4/0)	**16**(4/0)	**24**(4/2)	**24**(4/2)	**24**(4/2)	**32**(6/2)	**34**(6/2)	**32**(6/2)	**40**(8/2)

*The size of the index register (Xn) does not affect execution time.

Table 5-4. Move Long Instruction Execution Times

Source	Destination								
	Dn	**An**	**(An)**	**(An)+**	**–(An)**	**(d$_{16}$, An)**	**(d$_8$, An, Xn)***	**(xxx).W**	**(xxx).L**
Dn	**8**(2/0)	**8**(2/0)	**24**(2/4)	**24**(2/4)	**24**(2/4)	**32**(4/4)	**34**(4/4)	**32**(4/4)	**40**(6/4)
An	**8**(2/0)	**8**(2/0)	**24**(2/4)	**24**(2/4)	**24**(2/4)	**32**(4/4)	**34**(4/4)	**32**(4/4)	**40**(6/4)
(An)	**24**(6/0)	**24**(6/0)	**40**(6/4)	**40**(6/4)	**40**(6/4)	**48**(8/4)	**50**(8/4)	**48**(8/4)	**56**(10/4)
(An)+	**24**(6/0)	**24**(6/0)	**40**(6/4)	**40**(6/4)	**40**(6/4)	**48**(8/4)	**50**(8/4)	**48**(8/4)	**56**(10/4)
–(An)	**26**(6/0)	**26**(6/0)	**42**(6/4)	**42**(6/4)	**42**(6/4)	**50**(8/4)	**52**(8/4)	**50**(8/4)	**58**(10/4)
(d$_{16}$, An)	**32**(8/0)	**32**(8/0)	**48**(8/4)	**48**(8/4)	**48**(8/4)	**56**(10/4)	**58**(10/4)	**56**(10/4)	**64**(12/4)
(d$_8$, An, Xn)*	**34**(8/0)	**34**(8/0)	**50**(8/4)	**50**(8/4)	**50**(8/4)	**58**(10/4)	**60**(10/4)	**58**(10/4)	**66**(12/4)
(xxx).W	**32**(8/0)	**32**(8/0)	**48**(8/4)	**48**(8/4)	**48**(8/4)	**56**(10/4)	**58**(10/4)	**56**(10/4)	**64**(12/4)
(xxx).L	**40**(10/0)	**40**(10/0)	**56**(10/4)	**56**(10/4)	**56**(10/4)	**64**(12/4)	**66**(12/4)	**64**(12/4)	**72**(14/4)
(d$_{16}$, PC)	**32**(8/0)	**32**(8/0)	**48**(8/4)	**48**(8/4)	**48**(8/4)	**56**(10/4)	**58**(10/4)	**56**(10/4)	**64**(12/4)
(d$_8$, PC, Xn)*	**34**(8/0)	**34**(8/0)	**50**(8/4)	**50**(8/4)	**50**(8/4)	**58**(10/4)	**60**(10/4)	**58**(10/4)	**66**(12/4)
#<data>	**24**(6/0)	**24**(6/0)	**40**(6/4)	**40**(6/4)	**40**(6/4)	**48**(8/4)	**50**(8/4)	**48**(8/4)	**56**(10/4)

*The size of the index register (Xn) does not affect execution time.

5.3 STANDARD INSTRUCTION EXECUTION TIMES

The numbers of clock periods listed in Table 5-5 indicate the times required to perform the operations, store the results, and read the next instruction. The total number of clock periods, the number of read cycles, and the number of write cycles are shown in the previously described format. The number of clock periods, the number of read cycles, and the number of write cycles, respectively, must be added to those of the effective address calculation where indicated by a plus sign (+).

In Table 5-5, the following notation applies:

    An — Address register operand
    Dn — Data register operand
    ea — An operand specified by an effective address
    M — Memory effective address operand

Table 5-5. Standard Instruction Execution Times

Instruction	Size	op<ea>, An	op<ea>, Dn	op Dn, <M>
ADD/ADDA	Byte Word Long	— **12**(2/0)+ **10**(2/0)+**	**8**(2/0)+ **8**(2/0)+ **10**(2/0)+**	**12**(2/1)+ **16**(2/2)+ **24**(2/4)+
AND	Byte Word Long	— — —	**8**(2/0)+ **8**(2/0)+ **10**(2/0)+**	**12**(2/1)+ **16**(2/2)+ **24**(2/4)+
CMP/CMPA	Byte Word Long	— **10**(2/0)+ **10**(2/0)+	**8**(2/0)+ **8**(2/0)+ **10**(2/0)+	— — —
DIVS DIVU	Word Word	— —	**162**(2/0)+* **144**(2/0)+*	— —
EOR	Byte Word Long	— — —	**8**(2/0)+*** **8**(2/0)+*** **12**(2/0)+***	**12**(2/1)+ **16**(2/2)+ **24**(2/4)+
MULS MULU	Word Word	— —	**74**(2/0)+* **74**(2/0)+*	— —
OR	Byte Word Long	— — —	**8**(2/0)+ **8**(2/0)+ **10**(2/0)+**	**12**(2/1)+ **16**(2/2)+ **24**(2/4)+
SUB	Byte Word Long	**12**(2/0)+ **10**(2/0)+**	**8**(2/0)+ **8**(2/0)+ **10**(2/0)+**	**12**(2/1)+ **16**(2/2)+ **24**(2/4)+

+Add effective address calculation time.

*Indicates maximum base value added to word effective address time.

**The base time of 10 clock periods is increased to 12 if the effective address mode is register direct or immediate (effective address time should also be added).

***Only available effective address mode is data register direct.

MULS, MULU—The multiply algorithm requires 42+2n clocks where n is defined as:
   MULS: n = tag the <ea> with a zero as the MSB; n is the resultant number of 10 or 01 patterns in the 17-bit source; i.e., worst case happens when the source is $5555.
   MULU: n = the number of ones in the <ea>

5.4 IMMEDIATE INSTRUCTION EXECUTION TIMES

The numbers of clock periods listed in Table 5-6 include the times to fetch immediate operands, perform the operations, store the results, and read the next operation. The total number of clock periods, the number of read cycles, and the number of write cycles are shown in the previously described format. The number of clock periods, the number of read cycles, and the number of write cycles, respectively, must be added to those of the effective address calculation where indicated by a plus sign (+).

In Table 5-6, the following notation applies:

   # — Immediate operand
   Dn — Data register operand
   An — Address register operand
   M — Memory operand

Table 5-6. Immediate Instruction Execution Times

Instruction	Size	op #, Dn	op #, An	op #, M
ADDI	Byte Word Long	**16**(4/0) **16**(4/0) **28**(6/0)	— — —	**20**(4/1)+ **24**(4/2)+ **40**(6/4)+
ADDQ	Byte Word Long	**8**(2/0) **8**(2/0) **12**(2/0)	— **12**(2/0) **12**(2/0)	**12**(2/1)+ **16**(2/2)+ **24**(2/4)+
ANDI	Byte Word Long	**16**(4/0) **16**(4/0) **28**(6/0)	— — —	**20**(4/1)+ **24**(4/2)+ **40**(6/4)+
CMPI	Byte Word Long	**16**(4/0) **16**(4/0) **26**(6/0)	— — —	**16**(4/0) **16**(4/0) **24**(6/0)
EORI	Byte Word Long	**16**(4/0) **16**(4/0) **28**(6/0)	— — —	**20**(4/1)+ **24**(4/2)+ **40**(6/4)+
MOVEQ	Long	**8**(2/0)	—	—
ORI	Byte Word Long	**16**(4/0) **16**(4/0) **28**(6/0)	— — —	**20**(4/1)+ **24**(4/2)+ **40**(6/4)+
SUBI	Byte Word Long	**16**(4/0) **16**(4/0) **28**(6/0)	— — —	**12**(2/1)+ **16**(2/2)+ **24**(2/4)+
SUBQ	Byte Word Long	**8**(2/0) **8**(2/0) **12**(2/0)	— **12**(2/0) **12**(2/0)	**20**(4/1)+ **24**(4/2)+ **40**(6/4)+

+ Add effective address calculation time.

5.5 SINGLE OPERAND INSTRUCTION EXECUTION TIMES

Table 5-7 lists the timing data for the single operand instructions. The total number of clock periods, the number of read cycles, and the number of write cycles are shown in the previously described format. The number of clock periods, the number of read cycles, and the number of write cycles, respectively, must be added to those of the effective address calculation where indicated by a plus sign (+).

Table 5-7. Single Operand Instruction Execution Times

Instruction	Size	Register	Memory
CLR	Byte Word Long	**8**(2/0) **8**(2/0) **10**(2/0)	**12**(2/1)+ **16**(2/2)+ **24**(2/4)+
NBCD	Byte	**10**(2/0)	**12**(2/1)+
NEG	Byte Word Long	**8**(2/0) **8**(2/0) **10**(2/0)	**12**(2/1)+ **16**(2/2)+ **24**(2/4)+
NEGX	Byte Word Long	**8**(2/0) **8**(2/0) **10**(2/0)	**12**(2/1)+ **16**(2/2)+ **24**(2/4)+
NOT	Byte Word Long	**8**(2/0) **8**(2/0) **10**(2/0)	**12**(2/1)+ **16**(2/2)+ **24**(2/4)+
Scc	Byte, False Byte, True	**8**(2/0) **10**(2/0)	**12**(2/1)+ **12**(2/1)+
TAS	Byte	**8**(2/0)	**14**(2/1)+
TST	Byte Word Long	**8**(2/0) **8**(2/0) **8**(2/0)	**8**(2/0)+ **8**(2/0)+ **8**(2/0)+

+ Add effective address calculation time.

5.6 SHIFT/ROTATE INSTRUCTION EXECUTION TIMES

Table 5-8 lists the timing data for the shift and rotate instructions. The total number of clock periods, the number of read cycles, and the number of write cycles are shown in the previously described format. The number of clock periods, the number of read cycles, and the number of write cycles, respectively, must be added to those of the effective address calculation where indicated by a plus sign (+).

Table 5-8. Shift/Rotate Instruction Execution Times

Instruction	Size	Register	Memory
ASR, ASL	Byte Word Long	**10+2n**(2/0) **10+2n**(2/0) **12+n2**(2/0)	— **16**(2/2)+ —
LSR, LSL	Byte Word Long	**10+2n**(2/0) **10+2n**(2/0) **12+n2**(2/0)	— **16**(2/2)+ —
ROR, ROL	Byte Word Long	**10+2n**(2/0) **10+2n**(2/0) **12+n2**(2/0)	— **16**(2/2)+ —
ROXR, ROXL	Byte Word Long	**10+2n**(2/0) **10+2n**(2/0) **12+n2**(2/0)	— **16**(2/2)+ —

+ Add effective address calculation time for word operands.

n is the shift count.

5.7 BIT MANIPULATION INSTRUCTION EXECUTION TIMESS

Table 5-9 lists the timing data for the bit manipulation instructions. The total number of clock periods, the number of read cycles, and the number of write cycles are shown in the previously described format. The number of clock periods, the number of read cycles, and the number of write cycles, respectively, must be added to those of the effective address calculation where indicated by a plus sign (+).

Table 5-9. Bit Manipulation Instruction Execution Times

Instruction	Size	Dynamic		Static	
		Register	Memory	Register	Memory
BCHG	Byte Long	— **12**(2/0)*	**12**(2/1)+ —	— **20**(4/0)*	**20**(4/1)+ —
BCLR	Byte Long	— **14**(2/0)*	**12**(2/1)+ —	— **22**(4/0)*	**20**(4/1)+ —
BSET	Byte Long	— **12**(2/0)*	**12**(2/1)+ —	— **20**(4/0)*	**20**(4/1)+ —
BTST	Byte Long	— **10**(2/0)	**8**(2/0)+ —	— **18**(4/0)	**16**(4/0)+ —

+ Add effective address calculation time.

* Indicates maximum value; data addressing mode only.

5.8 CONDITIONAL INSTRUCTION EXECUTION TIMES

Table 5-10 lists the timing data for the conditional instructions. The total number of clock periods, the number of read cycles, and the number of write cycles are shown in the previously described format. The number of clock periods, the number of read cycles, and the number of write cycles, respectively, must be added to those of the effective address calculation where indicated by a plus sign (+).

Table 5-10. Conditional Instruction Execution Times

Instruction	Displacement	Trap or Branch Taken	Trap or Branch Not Taken
Bcc	Byte Word	**18**(4/0) **18**(4/0)	**12**(2/0) **20**(4/0)
BRA	Byte Word	**18**(4/0) **18**(4/0)	— —
BSR	Byte Word	**34**(4/4) **34**(4/4)	— —
DBcc	cc True cc False	— **18**(4/0)	**20**(4/0) **26**(6/0)
CHK	—	**68**(8/6)+*	**14**(2/0)+
TRAPV	—	**66**(10/6)	**8**(2/0)

+ Add effective address calculation time for word operand.

* Indicates maximum base value.

5.9 JMP, JSR, LEA, PEA, AND MOVEM INSTRUCTION EXECUTION TIMES

Table 5-11 lists the timing data for the jump (JMP), jump to subroutine (JSR), load effective address (LEA), push effective address (PEA), and move multiple registers (MOVEM) instructions. The total number of clock periods, the number of read cycles, and the number of write cycles are shown in the previously described format.

Table 5-11. JMP, JSR, LEA, PEA, and MOVEM Instruction Execution Times

Instruction	Size	(An)	(An)+	–(An)	(d_{16},An)	$(d_8,An,Xn)+$	(xxx).W	(xxx).L	$(d_{16}$ PC)	$(d_8,$ PC, Xn)*
JMP	—	**16**(4/0)	—	—	**18**(4/0)	**22**(4/0)	**18**(4/0)	**24**(6/0)	**18**(4/0)	**22**(4/0)
JSR	—	**32**(4/4)	—	—	**34**(4/4)	**38**(4/4)	**34**(4/4)	**40**(6/4)	**34**(4/4)	**32**(4/4)
LEA	—	**8**(2/0)	—	—	**16**(4/0)	**20**(4/0)	**16**(4/0)	**24**(6/0)	**16**(4/0)	**20**(4/0)
PEA	—	**24**(2/4)	—	—	**32**(4/4)	**36**(4/4)	**32**(4/4)	**40**(6/4)	**32**(4/4)	**36**(4/4)
MOVEM M → R	Word	**24+8n**(6+2n/0)	**24+8n**(6+2n/0)	—	**32+8n**(8+2n/0)	**34+8n**(8+2n/0)	**32+8n**(10+n/0)	**40+8n**(10+2n/0)	**32+8n**(8+2n/0)	**34+8n**(8+2n/0)
	Long	**24+16n**(6+4n/0)	**24+16n**(6+4n/0)	—	**32+16n**(8+4n/0)	**34+16n**(8+4n/0)	**32+16n**(8+4n/0)	**40+16n**(8+4n/0)	**32+16n**(8+4n/0)	**34+16n**(8+4n/0)
MOVEM R → M	Word	**16+8n**(4/2n)	—	**16+8n**(4/2n)	**24+8n**(6/2n)	**26+8n**(6/2n)	**24+8n**(6/2n)	**32+8n**(8/2n)	—	—
	Long	**16+16n**(4/4n)	—	**16+16n**(4/4n)	**24+16n**(6/4n)	**26+16n**	**24+16n**(8/4n)	**32+16n**(6/4n)	—	—

*The size of the index register (Xn) does not affect the instruction's execution time.

 n is the number of registers to move.

5.10 MULTIPRECISION INSTRUCTION EXECUTION TIMES

Table 5-12 lists the timing data for multiprecision instructions. The numbers of clock periods include the times to fetch both operands, perform the operations, store the results, and read the next instructions. The total number of clock periods, the number of read cycles, and the number of write cycles are shown in the previously described format.

The following notation applies in Table 5-12:

Dn — Data register operand

M — Memory operand

Table 5-12. Multiprecision Instruction Execution Times

Instruction	Size	op Dn, Dn	op M, M
ADDX	Byte Word Long	**8**(2/0) **8**(2/0) **12**(2/0)	**22**(4/1) **50**(6/2) **58**(10/4)
CMPM	Byte Word Long	— — —	**16**(4/0) **24**(6/0) **40**(10/0)
SUBX	Byte Word Long	**8**(2/0) **8**(2/0) **12**(2/0)	**22**(4/1) **50**(6/2) **58**(10/4)
ABCD	Byte	**10**(2/0)	**20**(4/1)
SBCD	Byte	**10**(2/0)	**20**(4/1)

5.11 MISCELLANEAOUS INSTRUCTION EXECUTION TIMES

Table 5-13 and Table 5-14 list the timing data for miscellaneous instructions. The total number of clock periods, the number of read cycles, and the number of write cycles are shown in the previously described format. The number of clock periods, the number of read cycles, and the number of write cycles, respectively, must be added to those of the effective address calculation where indicated by a plus sign (+).

Table 5-13. Miscellaneous Instruction Execution Times

Instruction	Register	Memory
ANDI to CCR	**32**(6/0)	—
ANDI to SR	**32**(6/0)	—
EORI to CCR	**32**(6/0)	—
EORI to SR	**32**(6/0)	—
EXG	**10**(2/0)	—
EXT	**8**(2/0)	—
LINK	**32**(4/4)	—
MOVE to CCR	**18**(4/0)	**18**(4/0)+
MOVE to SR	**18**(4/0)	**18**(4/0)+
MOVE from SR	**10**(2/0)	**16**(2/2)+
MOVE to USP	**8**(2/0)	—
MOVE from USP	**8**(2/0)	—
NOP	**8**(2/0)	—
ORI to CCR	**32**(6/0)	—
ORI to SR	**32**(6/0)	—
RESET	**136**(2/0)	—
RTE	**40**(10/0)	—
RTR	**40**(10/0)	—
RTS	**32**(8/0)	—
STOP	**4**(0/0)	—
SWAP	**8**(2/0)	—
UNLK	**24**(6/0)	—

+ Add effective address calculation time for word operand.

Table 5-14. Move Peripheral Instruction Execution Times

Instruction	Size	Register → Memory	Memory → Register
MOVEP	Word Long	**24**(4/2) **32**(4/4)	**24**(6/0) **32**(8/0)

+ Add effective address calculation time.

5.12 EXCEPTION PROCESSING EXECUTION TIMES

Table 5-15 lists the timing data for exception processing. The numbers of clock periods include the times for all stacking, the vector fetch, and the fetch of the first instruction of the handler routine. The total number of clock periods, the number of read cycles, and the number of write cycles are shown in the previously described format. The number of clock periods, the number of read cycles, and the number of write cycles, respectively, must be added to those of the effective address calculation where indicated by a plus sign (+).

Table 5-15. Exception Processing Execution Times

Exception	Periods
Address Error	**94**(8/14)
Bus Error	**94**(8/14)
CHK Instruction	**68**(8/6)+
Divide-by-Zero	**66**(8/6)+
Interrupt	**72**(9/6)*
Illegal Instruction	**62**(8/6)
Privilege Violation	**62**(8/6)
RESET**	**64**(12/0)
Trace	**62**(8/6)
TRAP Instruction	**62**(8/6)
TRAPV Instruction	**66**(10/6)

+Add effective address calculation time.

*The interrupt acknowledge cycle is assumed to take four clock periods.

**Indicates the time from when RESET and HALT are first sampled as negated to when instruction execution starts.

EC000 CORE PROCESSOR USER'S MANUAL   MOTOROLA
For More Information On This Product,
Go to: www.freescale.com

SECTION 6
16-BIT INSTRUCTION EXECUTION TIMES

This section contains listings of the instruction execution times in terms of external clock (CLKI) periods for the SCM68000 (EC000 core)[1] in 16-bit mode. In this data, it is assumed that both memory read and write cycles consist of four clock periods. A longer memory cycle causes the generation of wait states that must be added to the total instruction times.

The number of bus read and write cycles for each instruction is also included with the timing data. This data is shown as

$$n(r/w)$$

where:

　　n is the total number of clock periods
　　r is the number of read cycles
　　w is the number of write cycles

For example, a timing number shown as 18(3/1) means that the total number of clock periods is 18. Of the 18 clock periods, 12 are used for the three read cycles (four periods per cycle). Four additional clock periods are used for the single write cycle, for a total of 16 clock periods. The bus is idle for two clock periods during which the processor completes the internal operations required for the instruction.

NOTE

The total number of clock periods (n) includes instruction fetch and all applicable operand fetches and stores.

6.1 OPERAND EFFECTIVE ADDRESS CALCULATION TIMES

Table 6-1 lists the numbers of clock periods required to compute the effective addresses for instructions. The totals include fetching any extension words, computing the address, and fetching the memory operand. The total number of clock periods, the number of read cycles, and the number of write cycles (zero for all effective address calculations) are shown in the previously described format.

---

[1] The SCM68000 is the name of the Verilog model for the EC000 core. The remainder of this section will refer to the part as only the SCM68000.

Table 6-1. Effective Address Calculation Times

Addressing Mode		Byte, Word	Long
Register			
Dn	Data Register Direct	**0**(0/0)	**0**(0/0)
An	Address Register Direct	**0**(0/0)	**0**(0/0)
Memory			
(An)	Address Register Indirect	**4**(1/0)	**8**(2/0)
(An)+	Address Register Indirect with Postincrement	**4**(1/0)	**8**(2/0)
–(An)	Address Register Indirect with Predecrement	**6**(1/0)	**10**(2/0)
(d_{16}, An)	Address Register Indirect with Displacement	**8**(2/0)	**12**(3/0)
(d_8, An, Xn)*	Address Register Indirect with Index	**10**(2/0)	**14**(3/0)
(xxx).W	Absolute Short	**8**(2/0)	**12**(3/0)
(xxx).L	Absolute Long	**12**(3/0)	**16**(4/0)
(d_8, PC)	Program Counter Indirect with Displacement	**8**(2/0)	**12**(3/0)
(d_{16}, PC, Xn)*	Program Counter Indirect with Index	**10**(2/0)	**14**(3/0)
#<data>	Immediate	**4**(1/0)	**8**(2/0)

*The size of the index register (Xn) does not affect execution time.

6.2 MOVE INSTRUCTION EXECUTION TIMES

Table 6-2 and Table 6-3 list the numbers of clock periods required for the move instructions. The totals include instruction fetch, operand reads, and operand writes. The total number of clock periods, the number of read cycles, and the number of write cycles are shown in the previously described format.

Table 6-2. Move Byte and Word Instruction Execution Times

Source	Destination								
	Dn	An	(An)	(An)+	–(An)	(d_{16}, An)	(d_8, An, Xn)*	(xxx).W	(xxx).L
Dn	**4**(1/0)	**4**(1/0)	**8**(1/1)	**8**(1/1)	**8**(1/1)	**12**(2/1)	**14**(2/1)	**12**(2/1)	**16**(3/1)
An**	**4**(1/0)	**4**(1/0)	**8**(1/1)	**8**(1/1)	**8**(1/1)	**12**(2/1)	**14**(2/1)	**12**(2/1)	**16**(3/1)
(An)	**8**(2/0)	**8**(2/0)	**12**(2/1)	**12**(2/1)	**12**(2/1)	**16**(3/1)	**18**(3/1)	**16**(3/1)	**20**(4/1)
(An)+	**8**(2/0)	**8**(2/0)	**12**(2/1)	**12**(2/1)	**12**(2/1)	**16**(3/1)	**18**(3/1)	**16**(3/1)	**20**(4/1)
–(An)	**10**(2/0)	**10**(2/0)	**14**(2/1)	**14**(2/1)	**14**(2/1)	**18**(3/1)	**20**(3/1)	**18**(3/1)	**22**(4/1)
(d_{16}, An)	**12**(3/0)	**12**(3/0)	**16**(3/1)	**16**(3/1)	**16**(3/1)	**20**(4/1)	**22**(4/1)	**20**(4/1)	**24**(5/1)
(d_8, An, Xn)*	**14**(3/0)	**14**(3/0)	**18**(3/1)	**18**(3/1)	**18**(3/1)	**22**(4/1)	**24**(4/1)	**22**(4/1)	**26**(5/1)
(xxx).W	**12**(3/0)	**12**(3/0)	**16**(3/1)	**16**(3/1)	**16**(3/1)	**20**(4/1)	**22**(4/1)	**20**(4/1)	**24**(5/1)
(xxx).L	**16**(4/0)	**16**(4/0)	**20**(4/1)	**20**(4/1)	**20**(4/1)	**24**(5/1)	**26**(5/1)	**24**(5/1)	**28**(6/1)
(d_{16}, PC)	**12**(3/0)	**12**(3/0)	**16**(3/1)	**16**(3/1)	**16**(3/1)	**20**(4/1)	**22**(4/1)	**20**(4/1)	**24**(5/1)
(d_8, PC, Xn)*	**14**(3/0)	**14**(3/0)	**18**(3/1)	**18**(3/1)	**18**(3/1)	**22**(4/1)	**24**(4/1)	**22**(4/1)	**26**(5/1)
#<data>	**8**(2/0)	**8**(2/0)	**12**(2/1)	**12**(2/1)	**12**(2/1)	**16**(3/1)	**18**(3/1)	**16**(3/1)	**20**(4/1)

*The size of the index register (Xn) does not affect execution time.

**Only word operands; byte operands not allowed.

Table 6-3. Move Long Instruction Execution Times

Source	Destination								
	Dn	**An**	**(An)**	**(An)+**	**–(An)**	**(d₁₆, An)**	**(d₈, An, Xn)***	**(xxx).W**	**(xxx).L**
Dn An (An)	**4**(1/0) **4**(1/0) **12**(3/0)	**4**(1/0) **4**(1/0) **12**(3/0)	**12**(1/2) **12**(1/2) **20**(3/2)	**12**(1/2) **12**(1/2) **20**(3/2)	**12**(1/2) **12**(1/2) **20**(3/2)	**16**(2/2) **16**(2/2) **24**(4/2)	**18**(2/2) **18**(2/2) **26**(4/2)	**16**(2/2) **16**(2/2) **24**(4/2)	**20**(3/2) **20**(3/2) **28**(5/2)
(An)+ –(An) (d₁₆, An)	**12**(3/0) **14**(3/0) **16**(4/0)	**12**(3/0) **14**(3/0) **16**(4/0)	**20**(3/2) **22**(3/2) **24**(4/2)	**20**(3/2) **22**(3/2) **24**(4/2)	**20**(3/2) **22**(3/2) **24**(4/2)	**24**(4/2) **26**(4/2) **28**(5/2)	**26**(4/2) **28**(4/2) **30**(5/2)	**24**(4/2) **26**(4/2) **28**(5/2)	**28**(5/2) **30**(5/2) **32**(6/2)
(d₈, An, Xn)* (xxx).W (xxx).L	**18**(4/0) **16**(4/0) **20**(5/0)	**18**(4/0) **16**(4/0) **20**(5/0)	**26**(4/2) **24**(4/2) **28**(5/2)	**26**(4/2) **24**(4/2) **28**(5/2)	**26**(4/2) **24**(4/2) **28**(5/2)	**30**(5/2) **28**(5/2) **32**(6/2)	**32**(5/2) **30**(5/2) **34**(6/2)	**30**(5/2) **28**(5/2) **32**(6/2)	**34**(6/2) **32**(6/2) **36**(7/2)
(d, PC) (d, PC, Xn)* #<data>	**16**(4/0) **18**(4/0) **12**(3/0)	**16**(4/0) **18**(4/0) **12**(3/0)	**24**(4/2) **26**(4/2) **20**(3/2)	**24**(4/2) **26**(4/2) **20**(3/2)	**24**(4/2) **26**(4/2) **20**(3/2)	**28**(5/2) **30**(5/2) **24**(4/2)	**30**(5/2) **32**(5/2) **26**(4/2)	**28**(5/2) **30**(5/2) **24**(4/2)	**32**(5/2) **34**(6/2) **28**(5/2)

*The size of the index register (Xn) does not affect execution time.

6.3 STANDARD INSTRUCTION EXECUTION TIMES

The numbers of clock periods listed in Table 6-4 indicate the times required to perform the operations, store the results, and read the next instruction. The total number of clock periods, the number of read cycles, and the number of write cycles are shown in the previously described format. The number of clock periods, the number of read cycles, and the number of write cycles, respectively, must be added to those of the effective address calculation where indicated by a plus sign (+).

In Table 6-4, the following notation applies:

    An — Address register operand
    Dn — Data register operand
     ea — An operand specified by an effective address
     M — Memory effective address operand

Table 6-4. Standard Instruction Execution Times

Instruction	Size	op<ea>, An†	op<ea>, Dn	op Dn, <M>
ADD/ADDA	Byte, Word Long	**8**(1/0)+ **6**(1/0)+**	**4**(1/0)+ **6**(1/0)+**	**8**(1/1)+ **12**(1/2)+
AND	Byte, Word Long	— —	**4**(1/0)+ **6**(1/0)+**	**8**(1/1)+ **12**(1/2)+
CMP/CMPA	Byte, Word Long	**6**(1/0)+ **6**(1/0)+	**4**(1/0)+ **6**(1/0)+	— —
DIVS DIVU	Word Word	— —	**158**(1/0)+* **140**(1/0)+*	— —
EOR	Byte, Word Long	— —	**4**(1/0)*** **8**(1/0)***	**8**(1/1)+ **12**(1/2)+
MULS MULU	Word Word	— —	**70**(1/0)+* **70**(1/0)+*	— —
OR	Byte, Word Long	— —	**4**(1/0)+ **6**(1/0)+**	**8**(1/1)+ **12**(1/2)+
SUB	Byte, Word Long	**8**(1/0)+ **6**(1/0)+**	**4**(1/0)+ **6**(1/0)+**	**8**(1/1)+ **12**(1/2)+

+Add effective address calculation time.

†Word or long only.

*Indicates maximum basic value added to word effective address time.

**The base time of six clock periods is increased to eight if the effective address mode is register direct or immediate (effective address time should also be added).

***Only available effective address mode is data register direct.

MULS, MULU—The multiply algorithm requires 38+2n clocks where n is defined as:
    MULU: n = the number of ones in the <ea>
    MULS: n=concatenate the <ea> with a zero as the LSB; n is the resultant number of 10 or 01
      patterns in the 17-bit source; i.e., worst case happens when the source is $5555.

6.4 IMMEDIATE INSTRUCTION EXECUTION TIMES

The numbers of clock periods listed in Table 6-5 include the times to fetch immediate operands, perform the operations, store the results, and read the next operation. The total number of clock periods, the number of read cycles, and the number of write cycles are shown in the previously described format. The number of clock periods, the number of read cycles, and the number of write cycles, respectively, must be added to those of the effective address calculation where indicated by a plus sign (+).

In Table 6-5, the following notation applies:

    # — Immediate operand
  Dn — Data register operand
  An — Address register operand
   M — Memory operand

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

Table 6-5. Immediate Instruction Execution Times

Instruction	Size	op #, Dn	op #, An	op #, M
ADDI	Byte, Word Long	**8**(2/0) **16**(3/0)	— —	**12**(2/1)+ **20**(3/2)+
ADDQ	Byte, Word Long	**4**(1/0) **8**(1/0)	**4**(1/0)* **8**(1/0)	**8**(1/1)+ **12**(1/2)+
ANDI	Byte, Word Long	**8**(2/0) **14**(3/0)	— —	**12**(2/1)+ **20**(3/2)+
CMPI	Byte, Word Long	**8**(2/0) **14**(3/0)	— —	**8**(2/0)+ **12**(3/0)+
EORI	Byte, Word Long	**8**(2/0) **16**(3/0)	— —	**12**(2/1)+ **20**(3/2)+
MOVEQ	Long	**4**(1/0)	—	—
ORI	Byte, Word Long	**8**(2/0) **16**(3/0)	— —	**12**(2/1)+ **20**(3/2)+
SUBI	Byte, Word Long	**8**(2/0) **16**(3/0)	— —	**12**(2/1)+ **20**(3/2)+
SUBQ	Byte, Word Long	**4**(1/0) **8**(1/0)	**8**(1/0)* **8**(1/0)	**8**(1/1)+ **12**(1/2)+

6.5 SINGLE OPERAND INSTRUCTION EXECUTION TIMES

Table 6-6 lists the timing data for the single operand instructions. The total number of clock periods, the number of read cycles, and the number of write cycles are shown in the previously described format. The number of clock periods, the number of read cycles, and the number of write cycles, respectively, must be added to those of the effective address calculation where indicated by a plus sign (+).

Table 6-6. Single Operand Instruction Execution Times

Instruction	Size	Register	Memory
CLR	Byte, Word Long	**4**(1/0) **6**(1/0)	**8**(1/1)+ **12**(1/2)+
NBCD	Byte	**6**(1/0)	**8**(1/1)+
NEG	Byte, Word Long	**4**(1/0) **6**(1/0)	**8**(1/1)+ **12**(1/2)+
NEGX	Byte, Word Long	**4**(1/0) **6**(1/0)	**8**(1/1)+ **12**(1/2)+
NOT	Byte, Word Long	**4**(1/0) **6**(1/0)	**8**(1/1)+ **12**(1/2)+
Scc	Byte, False Byte, True	**4**(1/0) **6**(1/0)	**8**(1/1)+ **8**(1/1)+
TAS	Byte	**4**(1/0)	**14**(2/1)+
TST	Byte, Word Long	**4**(1/0) **4**(1/0)	**4**(1/0)+ **4**(1/0)+

+ Add effective address calculation time.

6.6 SHIFT/ROTATE INSTRUCTION EXECUTION TIMES

Table 6-7 lists the timing data for the shift and rotate instructions. The total number of clock periods, the number of read cycles, and the number of write cycles are shown in the previously described format. The number of clock periods, the number of read cycles, and the number of write cycles, respectively, must be added to those of the effective address calculation where indicated by a plus sign (+).

Table 6-7. Shift/Rotate Instruction Execution Times

Instruction	Size	Register	Memory
ASR, ASL	Byte, Word Long	**6+2n**(1/0) **8+2n**(1/0)	**8**(1/1)+ —
LSR, LSL	Byte, Word Long	**6+2n**(1/0) **8+2n**(1/0)	**8**(1/1)+ —
ROR, ROL	Byte, Word Long	**6+2n**(1/0) **8+2n**(1/0)	**8**(1/1)+ —
ROXR, ROXL	Byte, Word Long	**6+2n**(1/0) **8+2n**(1/0)	**8**(1/1)+ —

\+ Add effective address calculation time for word operands.

n is the shift count.

6.7 BIT MANIPULATION INSTRUCTION EXECUTION TIMES

Table 6-8 lists the timing data for the bit manipulation instructions. The total number of clock periods, the number of read cycles, and the number of write cycles are shown in the previously described format. The number of clock periods, the number of read cycles, and the number of write cycles, respectively, must be added to those of the effective address calculation where indicated by a plus sign (+).

Table 6-8. Bit Manipulation Instruction Execution Times

Instruction	Size	Dynamic		Static	
		Register	Memory	Register	Memory
BCHG	Byte Long	— **8**(1/0)*	**8**(1/1)+ —	— **12**(2/0)*	**12**(2/1)+ —
BCLR	Byte Long	— **10**(1/0)*	**8**(1/1)+ —	— **14**(2/0)*	**12**(2/1)+ —
BSET	Byte Long	— **8**(1/0)*	**8**(1/1)+ —	— **12**(2/0)*	**12**(2/1)+ —
BTST	Byte Long	— **6**(1/0)	**4**(1/0)+ —	— **10**(2/0)	**8**(2/0)+ —

\+ Add effective address calculation time.

* Indicates maximum value—data addressing mode only.

6.8 CONDITIONAL INSTRUCTION EXECUTION TIMES

Table 6-9 lists the timing data for the conditional instructions. The total number of clock periods, the number of read cycles, and the number of write cycles are shown in the previously described format.

Table 6-9. Conditional Instruction Execution Times

Instruction	Displacement	Branch Taken	Branch Not Taken
Bcc	Byte Word	**10**(2/0) **10**(2/0)	**8**(1/0) **12**(2/0)
BRA	Byte Word	**10**(2/0) **10**(2/0)	— —
BSR	Byte Word	**18**(2/2) **18**(2/2)	— —
CHK (No Trap)	—	**10**(1/0)+	—
DBcc	cc true cc false	— **10**(2/0)	**12**(2/0) **14**(3/0)
TRAPV	—	**4**(1/0)	—

6.9 JMP, JSR, LEA, PEA, AND MOVEM INSTRUCTION EXECUTION TIMES

Table 6-10 lists the timing data for the jump (JMP), jump to subroutine (JSR), load effective address (LEA), push effective address (PEA), and move multiple registers (MOVEM) instructions. The total number of clock periods, the number of read cycles, and the number of write cycles are shown in the previously described format.

Table 6-10. JMP, JSR, LEA, PEA, and MOVEM Instruction Execution Times

Instruction	Size	(An)	(An)+	–(An)	(d_{16},An)	$(d_8,An,Xn)+$	(xxx).W	(xxx).L	$(d_{16} PC)$	(d_8, PC, Xn)*
JMP	—	**8**(2/0)	—	—	**10**(2/0)	**14**(3/0)	**10**(2/0)	**12**(3/0)	**10**(2/0)	**14**(3/0)
JSR	—	**16**(2/2)	—	—	**18**(2/2)	**22**(2/2)	**18**(2/2)	**20**(3/2)	**18**(2/2)	**22**(2/2)
LEA	—	**4**(1/0)	—	—	**8**(2/0)	**12**(2/0)	**8**(2/0)	**12**(3/0)	**8**(2/0)	**12**(2/0)
PEA	—	**12**(1/2)	—	—	**16**(2/2)	**20**(2/2)	**16**(2/2)	**20**(3/2)	**16**(2/2)	**20**(2/2)
MOVEM $M \rightarrow R$	Word	**12+4n** (3+n/0)	**12+4n** (3+n/0)	—	**16+4n** (4+n/0)	**18+4n** (4+n/0)	**16+4n** (4+n/0)	**20+4n** (5+n/0)	**16+4n** (4n/0)	**18+4n** (4+n/0)
	Long	**12+8n** (3+2n/0)	**12+8n** (3+n/0)	—	**16+8n** (4+2n/0)	**18+8n** (4+2n/0)	**16+8n** (4+2n/0)	**20+8n** (5+2n/0)	**16+8n** (4+2n/0)	**18+8n** (4+2n/0)
MOVEM $R \rightarrow M$	Word	**8+4n** (2/n)	—	**8+4n** (2/n)	**12+4n** (3/n)	**14+4n** (3/n)	**12+4n** (3/n)	**16+4n** (4/n)	—	—
	Long	**8+8n** (2/2n)	—	**8+8n** (2/2n)	**12+8n** (3/2n)	**14+8n** (3/2n)	**12+8n** (3/2n)	**16+8n** (4/2n)	—	—

n is the number of registers to move.

*The size of the index register (Xn) does not affect the instruction's execution time.

6.10 MULTIPRECISION INSTRUCTION EXECUTION TIMES

Table 6-11 lists the timing data for multiprecision instructions. The number of clock periods includes the time to fetch both operands, perform the operations, store the results, and read the next instructions. The total number of clock periods, the number of read cycles, and the number of write cycles are shown in the previously described format.

The following notation applies in Table 6-11:

Dn — Data register operand
M — Memory operand

Table 6-11. Multiprecision Instruction Execution Times

Instruction	Size	op Dn, Dn	op M, M
ADDX	Byte, Word Long	**4**(1/0) **8**(1/0)	**18**(3/1) **30**(5/2)
CMPM	Byte, Word Long	— —	**12**(3/0) **20**(5/0)
SUBX	Byte, Word Long	**4**(1/0) **8**(1/0)	**18**(3/1) **30**(5/2)
ABCD	Byte	**6**(1/0)	**18**(3/1)
SBCD	Byte	**6**(1/0)	**18**(3/1)

6.11 MISCELLANEOUS INSTRUCTION EXECUTION TIMES

Table 6-12 and Table 6-13 list the timing data for miscellaneous instructions. The total number of clock periods, the number of read cycles, and the number of write cycles are shown in the previously described format. The number of clock periods, the number of read cycles, and the number of write cycles, respectively, must be added to those of the effective address calculation where indicated by a plus sign (+).

Table 6-12. Miscellaneous Instruction Execution Times

Instruction	Size	Register	Memory
ANDI to CCR	Byte	**20**(3/0)	—
ANDI to SR	Word	**20**(3/0)	—
EORI to CCR	Byte	**20**(3/0)	—
EORI to SR	Word	**20**(3/0)	—
ORI to CCR	Byte	**20**(3/0)	—
ORI to SR	Word	**20**(3/0)	—
MOVE from SR	Word	**6**(1/0)	**8**(1/1)+
MOVE to CCR	Word	**12**(1/0)	**12**(1/0)+
MOVE to SR	Word	**12**(2/0)	**12**(2/0)+
EXG	Long	**6**(1/0)	—
EXT	Word, Long	**4**(1/0)	—
LINK	Word	**16**(2/2)	—
MOVE from USP	Long	**4**(1/0)	—
MOVE to USP	Long	**4**(1/0)	—
NOP	—	**4**(1/0)	—
RESET	—	**132**(1/0)	—
RTE	—	**20**(5/0)	—
RTR	—	**20**(2/0)	—
RTS	—	**16**(4/0)	—
STOP	—	**4**(0/0)	—
SWAP	—	**4**(1/0)	—
UNLK	—	**12**(3/0)	—

Table 6-12. Miscellaneous Instruction Execution Times

+ Add effective address calculation time.

Table 6-13. Move Peripheral Instruction Execution Times

Instruction	Size	Register → Memory	Memory → Register
MOVEP	Word	**16**(2/2)	**16**(4/0)
	Long	**24**(2/4)	**24**(6/0)

6.12 EXCEPTION PROCESSING EXECUTION TIMES

Table 6-14 lists the timing data for exception processing. The numbers of clock periods include the times for all stacking, the vector fetch, and the fetch of the first instruction of the handler routine. The total number of clock periods, the number of read cycles, and the number of write cycles are shown in the previously described format. The number of clock periods, the number of read cycles, and the number of write cycles, respectively, must be added to those of the effective address calculation where indicated by a plus sign (+).

Table 6-14. Exception Processing Execution Times

Exception	Periods
Address Error	**50**(4/7)
Bus Error	**50**(4/7)
CHK Instruction	**40**(4/3)+
Divide-by-Zero	**38**(4/3)+
Illegal Instruction	**34**(4/3)
Interrupt	**44**(5/3)*
Privilege Violation	**34**(4/3)
RESET**	**40**(6/0)
Trace	**34**(4/3)
TRAP Instruction	**34**(4/3)
TRAPV Instruction	**34**(5/3)

+Add effective address calculation time.
*The interrupt acknowledge cycle is assumed to take four clock periods.
**Indicates the time from when RESET and HALT are first sampled as negated to when instruction execution starts.

EC000 CORE PROCESSOR USER'S MANUAL

SECTION 7
ELECTRICAL CHARACTERISTICS

This section provides information on the maximum ratings for the SCM68000 (EC000 core)[1].

7.1 MAXIMUM RATINGS

Rating	Symbol	Value	Unit
Supply Voltage	V_{CC}	–0.3 to 7.0	V
Input Voltage	V_{in}	–0.3 to 7.0	V
Maximum Operating Temperature Range	T_A	T_L to T_H –40 to100	°C
Storage Temperature	T_{stg}	–55 to 150	°C

7.2 CMOS CONSIDERATIONS

Because the basic CMOS cell is composed of two complementary transistors, a virtual semi-conductor controlled rectifier (SCR) may be formed when an input exceeds the supply volt-age. The SCR that is formed by this high input causes the device to become latched in a mode that may result in excessive current drain and eventual destruction of the device. Although the SCM68000 is implemented with input protection diodes, care should be exer-cised to ensure that the maximum input voltage specification is not exceeded. Some sys-tems may require that the CMOS circuitry be isolated from voltage transients; other may require additional circuitry.

7.3 POWER CONSUMPTION

Characteristic	3.3 V		5 V		Unit
	Typical	Max	Typical	Max	
Operating Current @16 MHz	15	30	25	50	mA
Standby Current	TBD	TBD	TBD	TBD	mA

7.4 AC ELECTRICAL SPECIFICATION DEFINITIONS

The AC specifications presented consist of output delays, input setup and hold times, and signal skew times. All signals are specified relative to an appropriate edge of the clock and possibly to one or more other signals.

[1.] The SCM68000 is the name of the Verilog model for the EC000 core. The remainder of this section will refer to the part as only the SCM68000.

7.5 AC ELECTRICAL SPECIFICATIONS—CLOCK TIMING (see Figure 7-1)

Num	Characteristic	3.3 V		5.0 V		Unit
		Min	Max	Min	Max	
	Frequency of Operation	—	20.0	—	20.0	MHz
1	Cycle Time	50	—	50	—	ns
2,3	Clock Pulse Width	25	—	25	—	ns
4,5	Clock Rise and Fall Times	—	4	—	4	ns
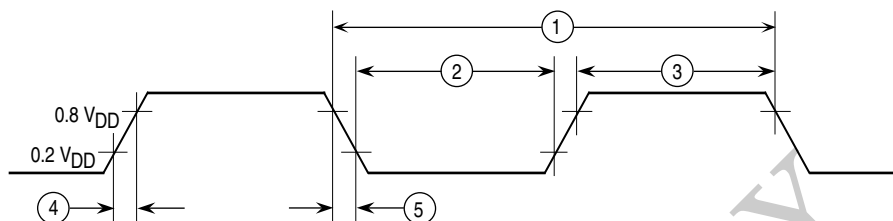
NOTE: Timing measurements are referenced to and from a low voltage of 0.2 VDD and a high voltage of 0.8 VDD, unless otherwise noted. The voltage swing through this range should start outside and pass through the range such that the rise or fall will be linear between 0.2 VDD and 0.8 VDD.

Figure 7-1. Clock Input Timing Diagram

7.6 AC ELECTRICAL SPECIFICATIONS—READ AND WRITE CYCLES

(Frequency = 0 to 20 MHz; GND = 0 V; $T_A = T_L$ to T_H; see Figure 7-2 and Figure 7-3)

Num	Characteristic	3.3 V		5.0 V		Unit
		Min	Max	Min	Max	
6	Clock High to Address Valid	—	20	—	18	ns
6A	Clock High to FC Valid	0	20	0	18	ns
7	Clock High to Address, Data Bus High Impedance (Maximum) (Write)	—	25	—	20	ns
8	Clock High to Address, FC Invalid (Minimum)	0	—	0	—	ns
9	Clock High to ASB, DSB Asserted	1	20	1	18	ns
11[1]	Address Valid to ASB, DSB Asserted (Read)/ASB Asserted (Write)	40	—	40	—	ns
11A[1]	FC Valid to ASB, DSB Asserted (Read)/ASB Asserted (Write)	40	—	40	—	ns
12	Clock Low to ASB, DSB Negated	1	20	1	18	ns
13[1]	ASB, DSB Negated to Address, FC Invalid	10	—	10	—	ns
14[1,5]	ASB and DSB Width Asserted (Read)/ASB Width Asserted (Write)	100	—	100	—	ns
14A[6]	DSB Width Asserted (Write)	50	—	50	—	ns
15[1]	ASB, DSB Width Negated (Read)/ASB Width Negated (Write)	50	—	50	—	ns
17[1,7]	ASB, DSB Negated to RWB Invalid	5	—	5	—	ns
18	Clock High to RWB High (Read)	0	20	0	18	ns
20	Clock High to RWB Low (Write)	0	20	0	18	ns
20A	ASB Asserted to RWB Low (Write)	—	10	—	10	ns
21[1]	Address Valid to RWB Low (Write)	0	—	0	—	ns
21A[1]	FC Valid to RWB Low (Write)	25	—	25	—	ns
22[1]	RWB Low to DSB Asserted (Write)	10	—	10	—	ns
23	Clock Low to Data-Out Valid (Write)	—	20	—	18	ns

Num	Characteristic	3.3 V		5.0 V		Unit
		Min	Max	Min	Max	
25[1]	ASB, DSB Negated to Data-Out Invalid (Write)	10	—	10	—	ns
26[1]	Data-Out Valid to DSB Asserted (Write)	10	—	10	—	ns
27[4]	Data-In Valid to Clock Low (Setup Time on Read)	5	—	5	—	ns
28[1]	ASB, DSB Negated to DTACKB Negated (Asynchronous Hold)	0	95	0	95	ns
28A[1]	Clock High to DTACKB Negated	—	95	—	95	ns
29	ASB, DSB Negated to Data-In Invalid (Hold Time on Read)	0	—	0	—	ns
29A	ASB, DSB Negated to Data-In High Impedance (Read)	—	75	—	75	ns
30	ASB, DSB Negated to BERRB Negated	0	—	0	—	ns
31[1,4]	DTACKB Asserted to Data-In Valid (Setup Time on Read)	—	42	—	42	ns
32	HALTIB and RESETIB Input Transition Time	0	100	0	100	ns
47[4]	Asynchronous Input Setup Time	5	—	5	—	ns
48[1,2]	BERRB Asserted to DTACKB Asserted	10	—	10	—	ns
53	Data-Out Hold from Clock High (Write)	0	—	0	—	ns
55	RWB Asserted to Data Bus Impedance Change (Write)	0	—	0	—	ns
56[3]	HALTIB, RESETIB Pulse Width	10	—	10	—	clks

NOTES:

1. Actual value depends on clock period.

2. If #47 is satisfied for both DTACKB and BERRB, #48 may be ignored. In the absence of DTACKB, BERRB is an asynchronous input using the asynchronous input setup time (#47).

3. For power-up, the MC68000 must be held in the reset state for 132 clock cycles to allow stablization of on-chip circuitry. After the system is powered up, #56 refers to the minimum pulse width required to reset the processor.

4. If the asynchronous input setup time (#47) requirement is satisfied for DTACKB, the DTACKB asserted to data setup time (#31) requirement can be ignored. The data must only satisfy the data-in to clock low setup time (#27) for the following clock cycle.

5. ASB and DSB will be asserted from the rising edge of state 2 (S2) until the falling edge of state 7 (S7).

6. During a write, DSB will be asserted in state 4 (S4) and negated in S7.

7. With consecutive bus cycles, ASB and DSB are negated from the falling edge of S7 to the rising edge of S2.
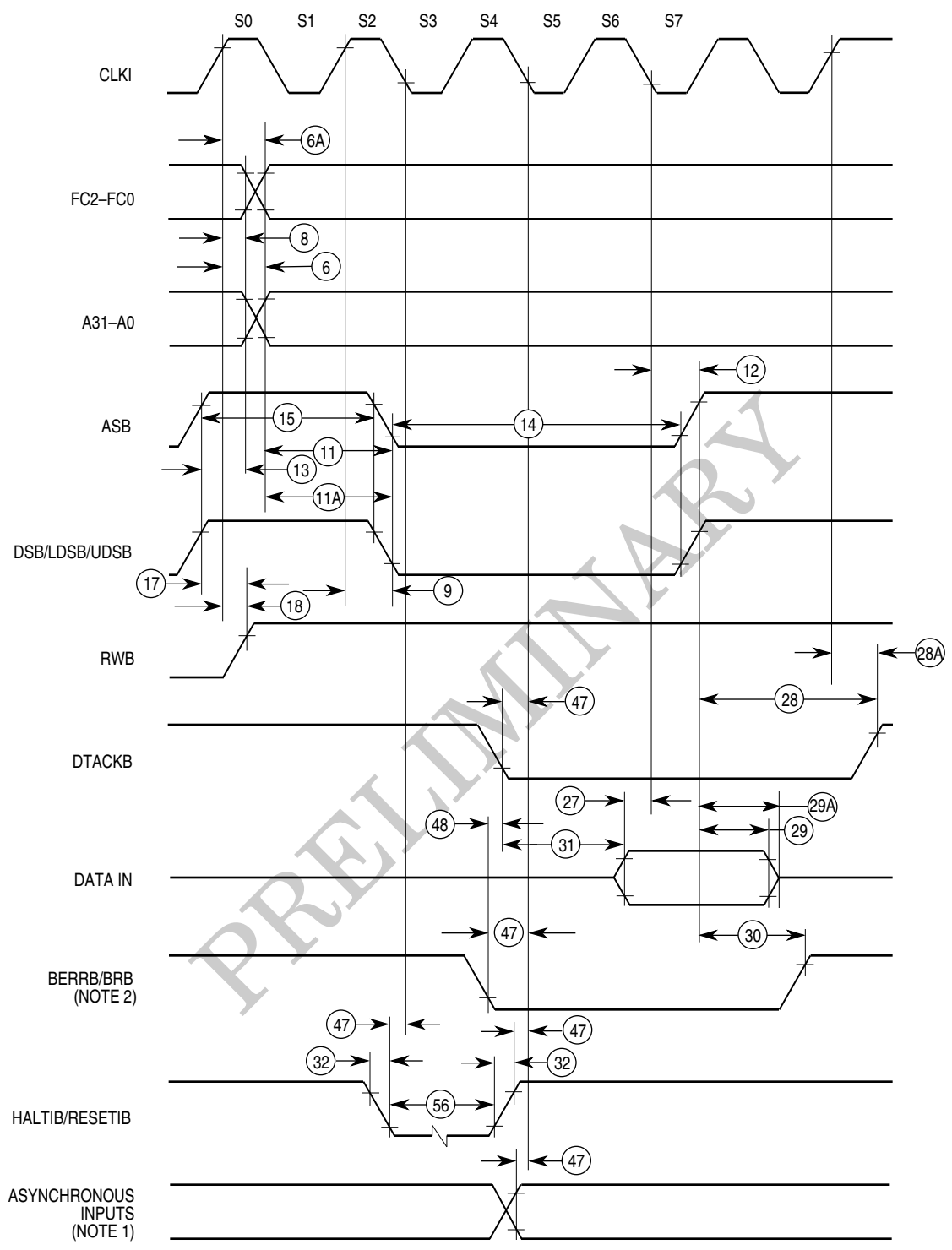
NOTES:
1. Setup time for the asynchronous inputs IPLB2–IPLB0 (spec #47) guarantees their recognition at the next falling edge of the clock.
2. BRB need fall at this time only to ensure being recognized at the end of the bus cycle.
3. Timing measurements are referenced to and from a low voltage of 0.2 VDD and a high voltage of 0.8 VDD, unless otherwise noted. The voltage swing through this range should start outside and pass through the range such that the rise or fall is linear between 0.2 VDD and 0.8 VDD.

Figure 7-2. Read Cycle Timing Diagram
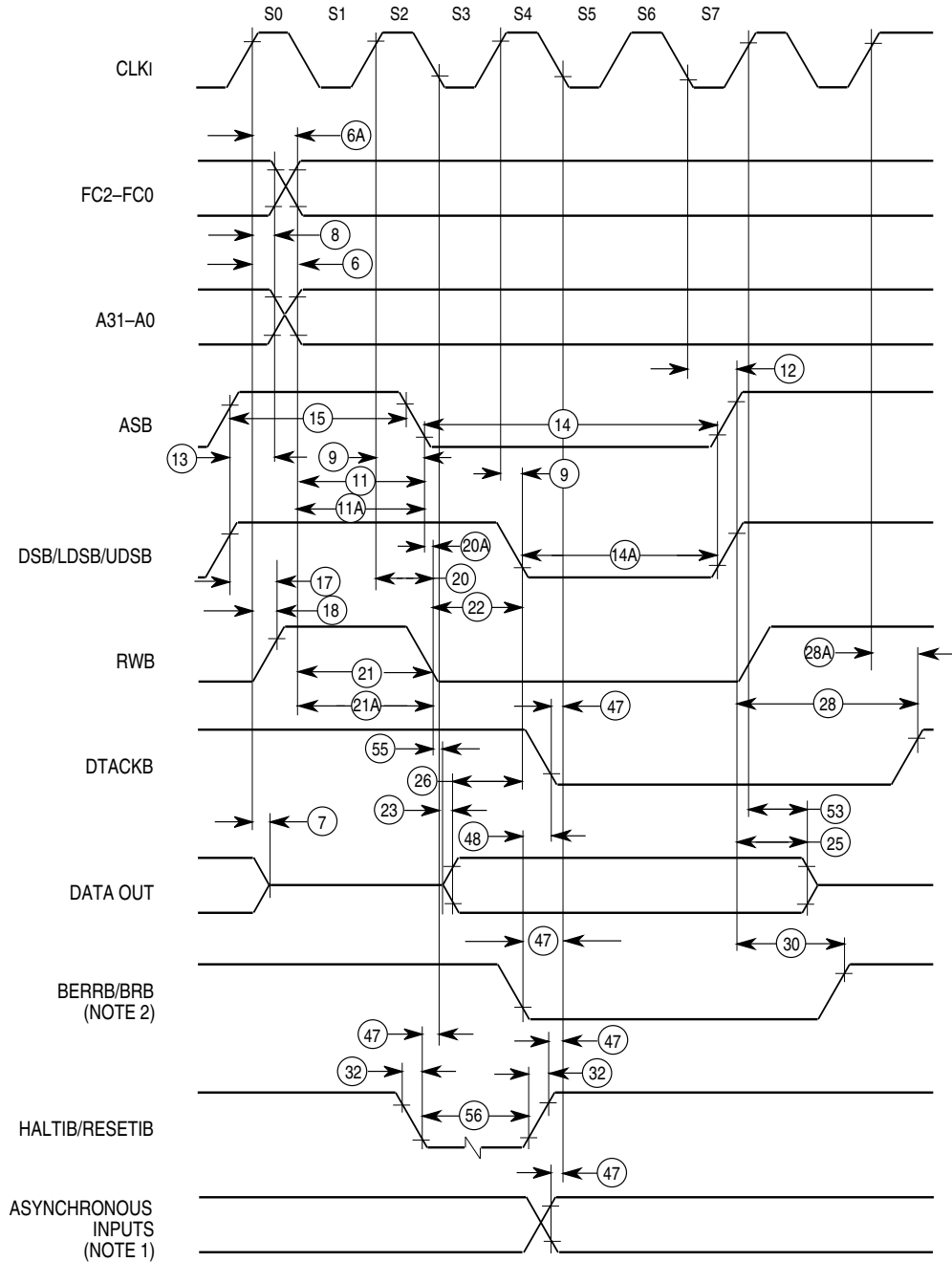
NOTES:
1. Timing measurements are referenced to and from a low voltage of 0.2 VDD and a high voltage of 0.8 VDD, unless otherwise noted. The voltage swing through this range should start outside and pass through the range such that the rise or fall is linear between 0.2 VDD and 0.8 VDD.
2. Because of loading variations, RWB may be valid after ASB even though both are initiated by the rising edge of S2 (specification #20A).

Figure 7-3. Write Cycle Timing Diagram

7.7 AC ELECTRICAL SPECIFICATIONS—SCM68000 TO EXTERNAL PERIPHERALS

(Frequency = 0 to 20 MHz; GND = 0 Vdc; $T_A = T_L$ to T_H; refer to Figure 7-4)

Num	Characteristic	3.3 V		5.0 V		Unit
		Min	Max	Min	Max	
12	Clock Low to ASB, DSB Negated	1	20	1	18	ns
18	Clock High to RWB High (Read)	0	20	0	18	ns
20	Clock High to RWB Low (Write)	0	20	0	20	ns
23	Clock Low to Data-Out Valid (Write)	—	20	—	18	ns
27	Data-In Valid to Clock Low (Setup Time on Read)	5	—	5	—	ns
29	ASB, DSB Negated to Data-In Invalid (Hold Time on Read)	0	—	0	—	ns
41	Clock Low to TESTCLK Transition	—	20	—	18	ns
42	TESTCLK Output Rise and Fall Time	—	12	—	12	ns
44	ASB, DSB Negated to AVECB Negated	0	42	0	42	ns
45	TESTCLK Low to Control, Address Bus Invalid (Address Hold Time)	10	—	10	—	ns
47	Asynchronous Input Setup Time	5	—	5	—	ns
49[1]	ASB, DSB Negated to TESTCLK Low	−30	30	−30	30	ns
50	TESTCLK Width High	190	—	190	—	ns
51	TESTCLK Width Low	290	—	290	—	ns
54	TESTCLK Low to Data-Out Invalid	5	—	5	—	ns

NOTE:

1.The falling edge of S6 triggers both the negation of the strobes (ASB and DSB) and the falling edge of TESTCLK. Either of these events can occur first, depending upon the loading on each signal. Specificaton #49 indicates the absolute maximum skew that will occur between the rising edge of the strobes and the falling edge of TESTCLK.
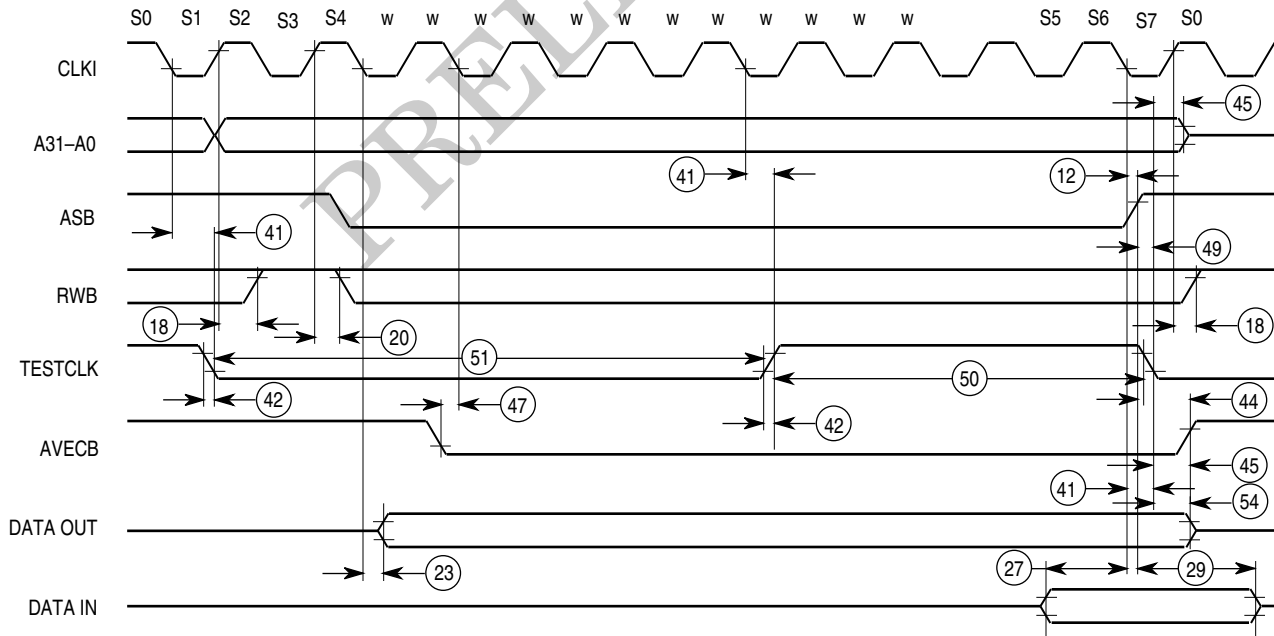
Figure 7-4. SCM68000 to External Peripherals Timing Diagram

EC000 CORE PROCESSOR USER'S MANUAL MOTOROLA

7.8 AC ELECTRICAL SPECIFICATIONS—BUS ARBITRATION

(Frequency = 0 to 20 MHz; GND = 0 Vdc, $T_A = T_L$ to T_H; see Figure 7-5 through Figure 7-8)

Num	Characteristic	3.3 V		5.0 V		Unit
		Min	Max	Min	Max	
7	Clock High to Address, Data Bus High Impedance (Maximum)	—	25	—	20	ns
33	Clock High to BGB Asserted	0	25	0	25	ns
34	Clock High to BGB Negated	0	25	0	25	ns
35	BRB Asserted to BGB Asserted	1.5	3.5	1.5	3.5	Clks
36[1]	BRB Negated to BGB Negated	1.5	3.5	1.5	3.5	Clks
37	BGACKB Asserted to BGB Negated	1.5	3.5	1.5	3.5	Clks
37A[2,4]	BGACKB Asserted to BRB Negated	10 ns	1.5 Clks	10 ns	1.5 Clks	—
38	BGB Asserted to Control, Address, Data Bus High Impedance (ASB Negated)	—	42	—	42	ns
39	BGB Width Negated	1.5	—	1.5	—	Clks
46	BGACKB Width Low	1.5	—	1.5	—	Clks
47	Asynchronous Input Setup Time	5	—	5	—	ns
57	BGACKB Negated to ASB, DSB, RWB Driven	1.5	—	1.5	—	Clks
57A	BGACKB Negated to FC Driven	1	—	1	—	Clks
58[3]	BRB Negated to ASB, DSB, RWB Driven	1.5	—	1.5	—	Clks
58A[3]	BRB Negated to FC Driven	1	—	1	—	Clks

NOTES:

1. Setup time for the synchronous inputs BGACKB, IPLB2–IPLB0, and AVECB guarantees their recognition at the next falling edge of the clock.

2. BRB need fall at this time only in order to insure being recognized at the end of the bus cycle.

3. The processor will negate BG and begin driving the bus again if external arbitration logic negates BR before asserting BGACKB.

4. The minimum value must be met to guarantee proper operation. If the maximum value is exceeded, BG may be reasserted.
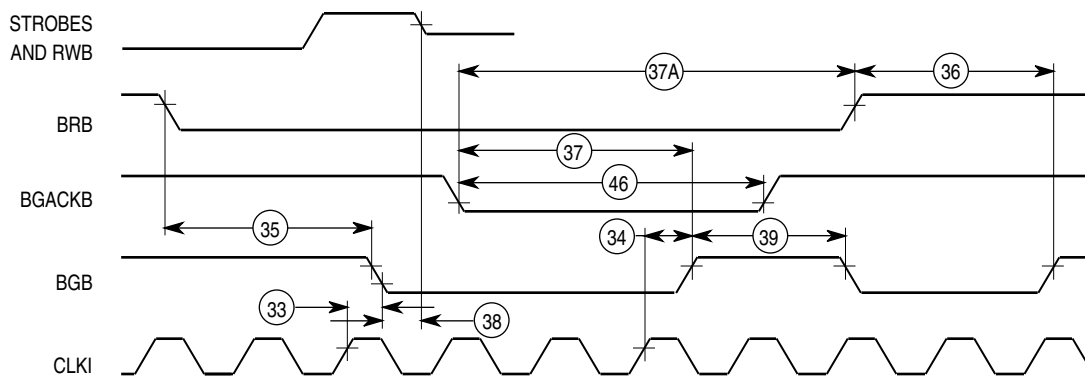
NOTE: Setup time to the clock (#47) for the asynchronous inputs BERRB, BGACKB, BRB, DTACKB, IPLB2–IPLB0, and AVECB guarantees their recognition at the next falling edge of the clock.

Figure 7-5. Bus Arbitration Timing Diagram
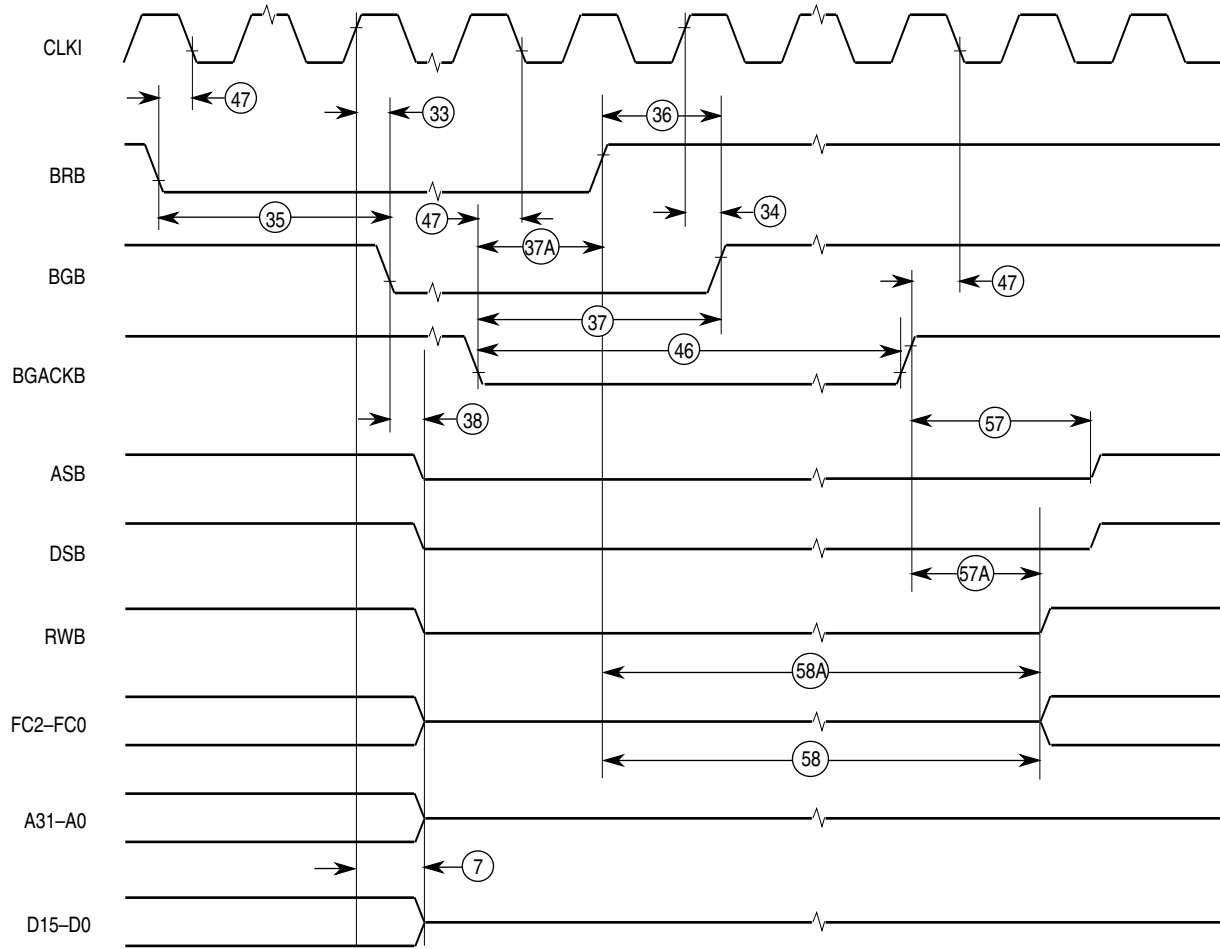
NOTE: Waveform measurements for all inputs and outputs are specified at: logic high = 0.8 VDD, logic low = 0.2 VDD.

Figure 7-6. Bus Arbitration Timing Diagram—Idle Bus Case
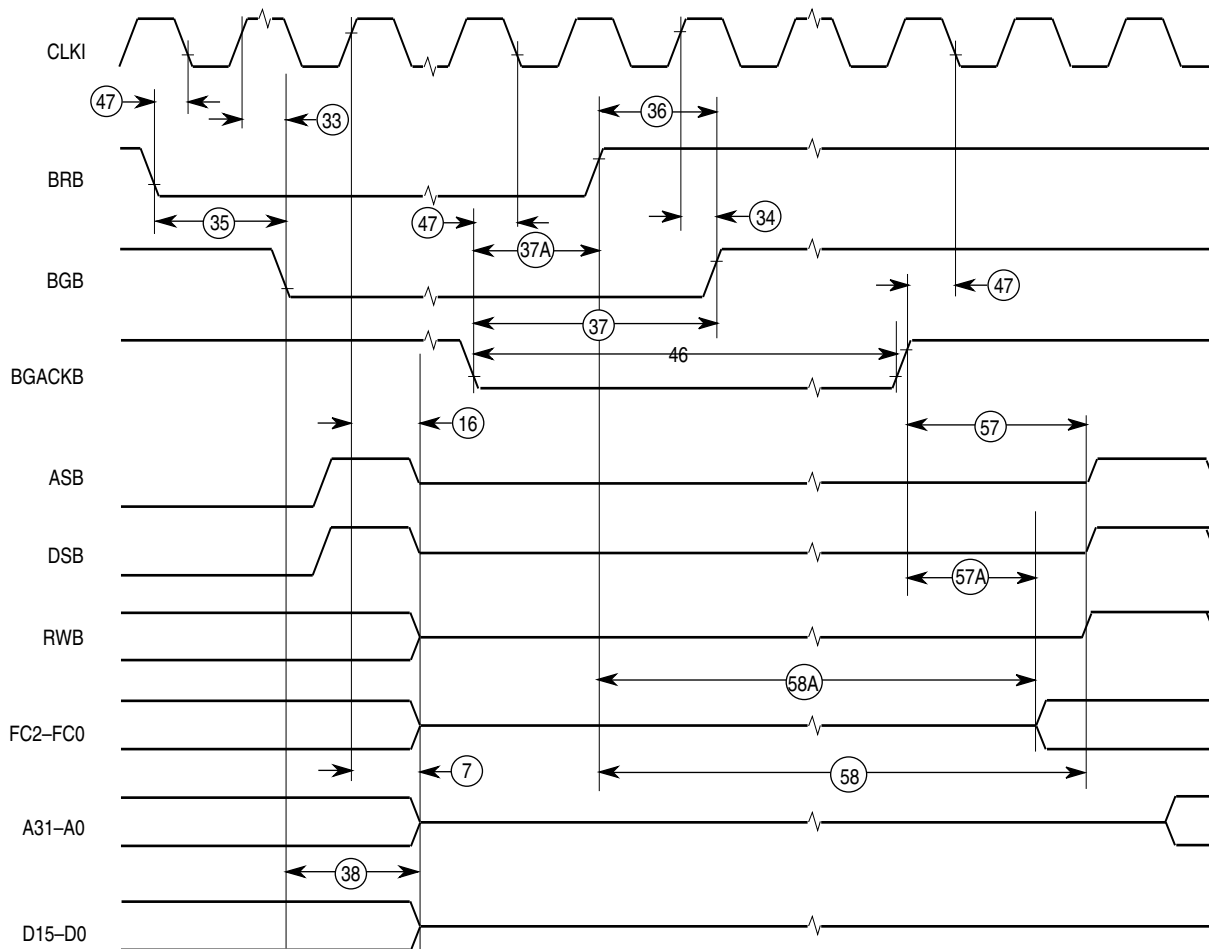
EC000 CORE PROCESSOR USER'S MANUAL MOTOROLA
For More Information On This Product,
Go to: www.freescale.com

NOTE: Waveform measurements for all inputs and outputs are specified at: logic high = 0.8 VDD, logic low = 0.2 VDD.

Figure 7-7. Bus Arbitration Timing Diagram—Active Bus Case
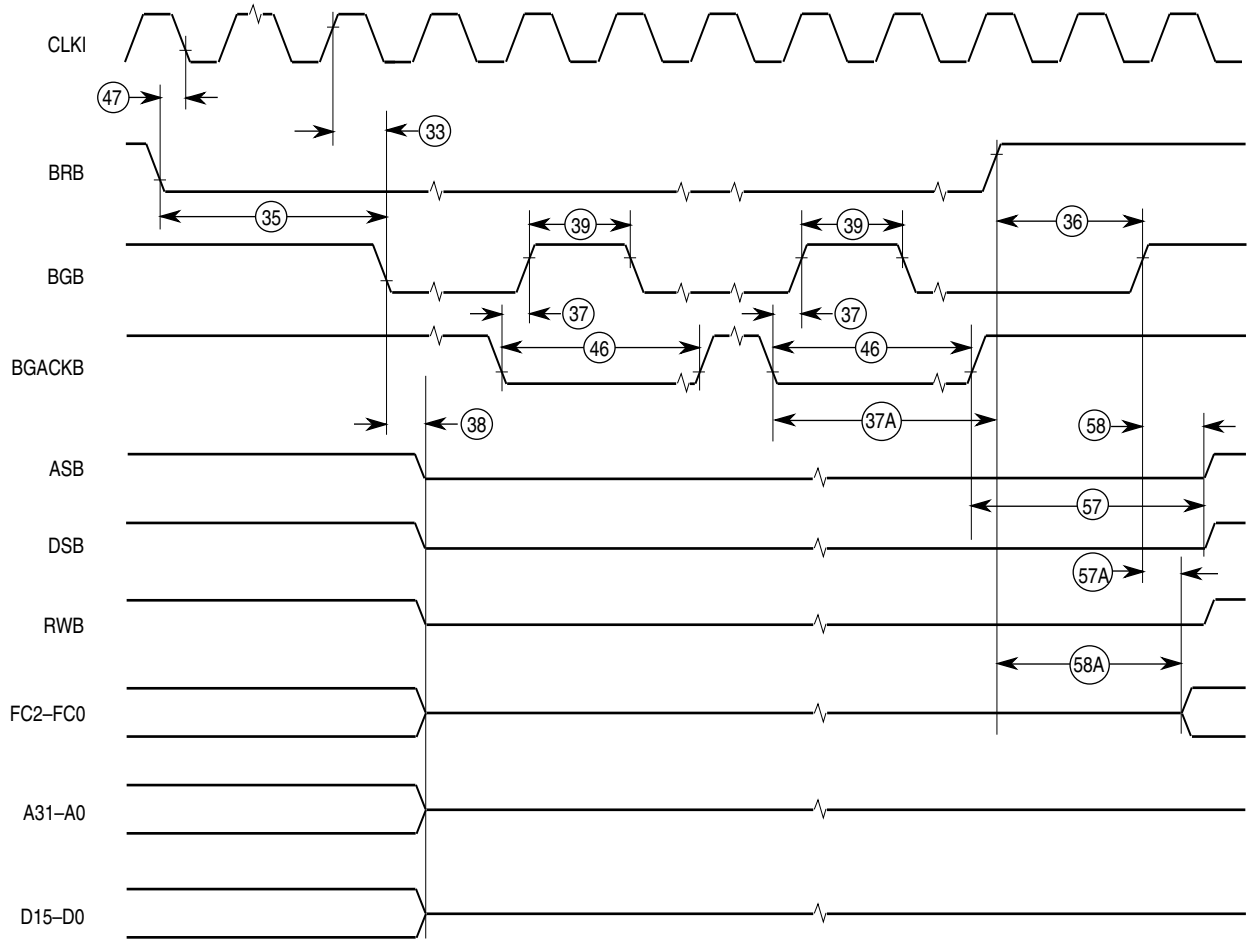
NOTE: Waveform measurements for all inputs and outputs are specified at: logic high = 0.8 VDD, logic low = 0.2 VDD.

Figure 7-8. Bus Arbitration Timing Diagram—Multiple Bus Request

7.9 AC ELECTRICAL SPECIFICATIONS—CORE APPLICATIONS SIGNALS

(Frequency = 0 to 20 MHz; GND = 0 Vdc; $T_A = T_L$ to T_H; refer to Figure 7-9)

Num	Characteristic	3.3 V		5.0 V		Unit
		Min	Max	Min	Max	
6	Clock Low to SIZ1–SIZ0 Asserted	—	20	—	18	ns
17	ASB, DSB Negated to ERWB High	15	—	15	—	ns
18	Clock High to ERWB High	0	20	0	18	ns
C3	Clock Low to ERWB Low	—	20	—	18	Clks
C4	Clock Low to RMCB Asserted	—	20	—	18	Clks
C5	Clock Low to RMCB Negated	—	20	—	18	ns
C7	Clock Low to TSCAE Asserted	—	20	—	18	ns
C8	Clock High to STOP Asserted	—	20	—	18	ns
C9	Clock Low to IPENDB Asserted	—	20	—	18	ns
C10	Clock Low to REFILLB Asserted	—	20	—	18	ns
C11	Clock Low to STATUSB Asserted	—	20	—	18	ns
C12	STOP Pulse Width	—	2	—	2	Clks
C14	REFILLB Pulse Width	—	1	—	1	Clks
C15	STATUSB Pulse Width	1	3	1	3	Clks
C16	CLKI to CLKO	—	5	—	5	ns
C17[1]	Clock to AOEB, COEB, and DOEB Asserted	—	20	—	18	ns

NOTE:
1. AOEB, COEB, and DOEB will be a logic high only to put the appropriate signals into a high-impedance state. See **Section 2 Signal Description** for more information about these signals.
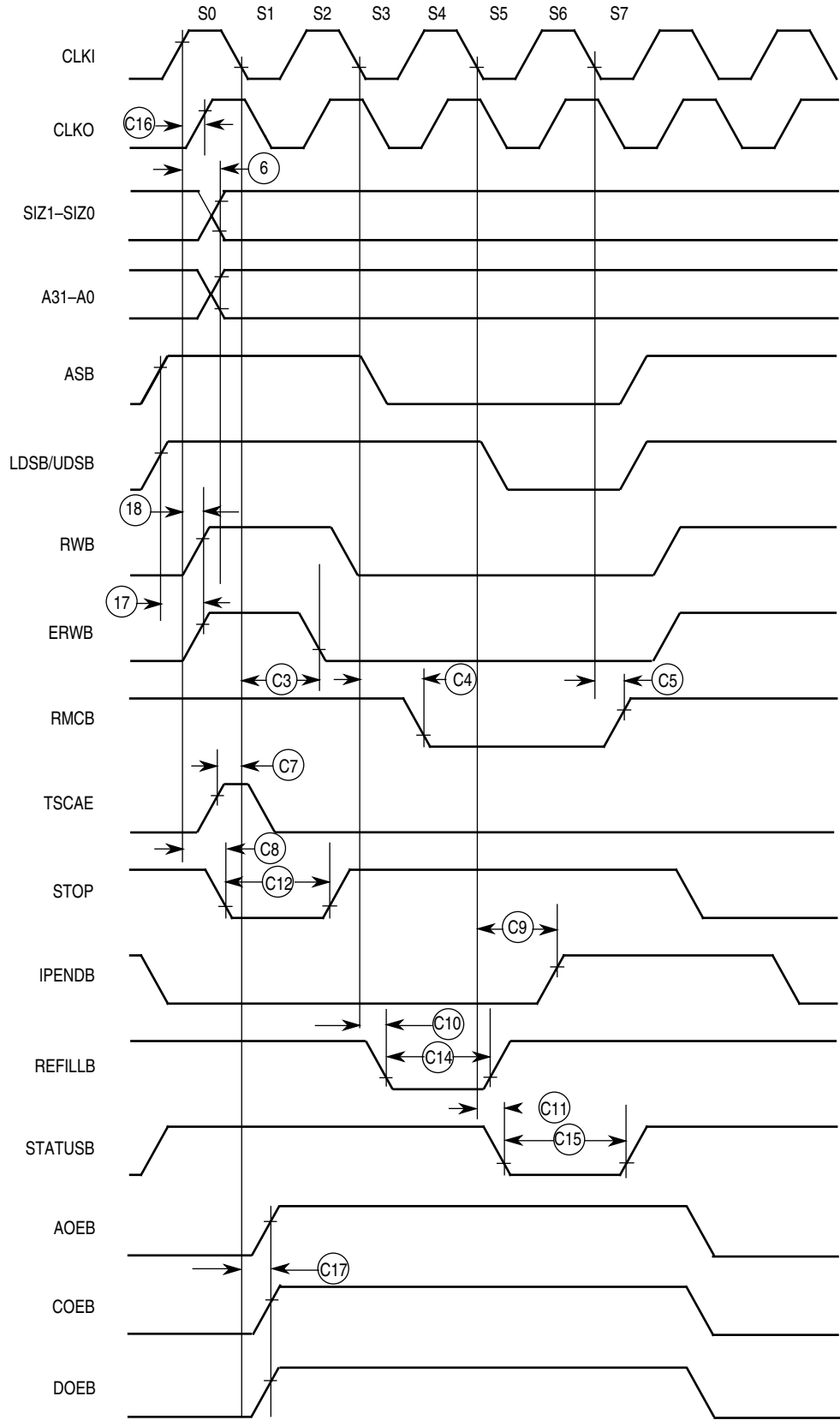
Figure 7-9. Core Application Signals Timing Diagram

EC000 CORE PROCESSOR USER'S MANUAL MOTOROLA
For More Information On This Product,
Go to: www.freescale.com

INDEX