Parts Not Suitable for New Designs

For Additional Information

End-Of-Life Product Change Notice

MOTOROLA

MC68349

Dragon I™— High Performance Integrated Microprocessor User's Manual

PREFACE

The complete documentation package for the MC68349 consists of the MC68349UM/AD, *MC68349 Dragon I—High Performance Integrated Microprocessor User's Manual,* M68000PM/AD, *MC68000 Family Programmer's Reference Manual,* and the MC68349P/D, *MC68349 Dragon I—High Performance Integrated Microprocessor Product Brief*.

The *MC68349 Dragon I—High Performance Integrated Microprocessor User's Manual* describes the programming, capabilities, registers, and operation of the MC68349; the *MC68000 Family Programmer's Reference Manual* provides instruction details for the MC68349; and the *MC68349 Dragon I—High Performance Integrated Microprocessor Product Brief* provides a brief description of the MC68349 capabilities.

This user's manual is organized as follows:

Section 1	Overview
Section 2	Signal Descriptions
Section 3	Bus Operation
Section 4	System Integration Module
Section 5	CPU030
Section 6	Quad Data Memory Module
Section 7	DMA Controller Module
Section 8	Serial Module
Section 9	IEEE 1149.1 Test Access Port
Section 10	Applications
Section 11	Electrical Characteristics
Section 12	Ordering Information and Mechanical Data

MC68349 ACRONYM LIST

ACR — Auxiliary Control Register
ADDR — Address Column
AVR — Autovector Register
BDM — Background Debug Mode
BSA — Bus State Analyzer
BTC — Byte Transfer Count Register
CCR — Channel Control Register
CCR — Condition Code Register
CIC — Configurable Instruction Cache
CR — Command Register
CSR — Channel Status Register
CSR — Clock Select Register
DAR — Destination Address Register
DDRA — Port A Data Direction Register
DDRB — Port B Data Direction Register
DFC — Destination Fuction Code Register
DHR — Data Holding Register
DMA — Direct Memory Access
DRAM — Dynamic Random Access Memory
DUART — Dual Universal Asynchronous Receiver/Transmitter
EA — Effective Address
FAR — Fault Address Register
EBI — External Bus Interface
FC — Function Code
FCR — Function Control Register
FIFO — First-in/First-out
HCMOS — High-Density Complimentary Metal Oxide Semiconductor
HLL — High Level Language
IDR — Identification Register
IEEE — Institute of Electrical and Electronic Engineers
IER — Interrupt Status Register
ILR — Interrupt Level Register
IMB — Intermodule Bus
I/O — Input/Output
IP — Input Port Register
IPCR — Input Port Change Register
IRA — 16-Bit Instruction Register A
IRB — 16-Bit Instruction Register B
IRC — 16-Bit Instruction Register C

MC68349 ACRONYM LIST (Continued)

IRL — 16-Bit Instruction Register L
ISR — Interrupt Service Routine
ISR — Interrupt Status Register
IVR — Interrupt Vector Register
JTAG — Joint Test Action Group
LPSTOP — Low Power Stop
MBAR — Module Base Address Register
MCR — Module Configuration Register
MR — Mode Register
MSB — Most Significant Bit
OP — Output Port Data Register
OPCR — Output Port Control Register
PC — Program Counter
PCC — Current Instruction Program Counter
PDA — Personal Digital Assistants
PICR — Periodic Interrupt Control Register
PITR — Periodic Interrupt Timing Register
PLL — Phase Lock Loop
POR — Power On Reset
PORTA — Port A Data Register
PORTB — Port B Data Register
PPARA — Port A Pin Assignment Register
PPARB — Port B Pin Assignment Register
QDMM — Quad Data Memory Module
RAM — Random Access Memory
RAREG/RDREG — Read A/D Register
ROM — Read Only Memory
RPC — Return Program Counter
RSR — Reset Status Register
RSREG — Read System Register
S0, S1 — State Zero, State One
SAR — Source Address Register
SFC — Source Funtion Code Register
SIM49 — MC68349 System Integration Module
SP — Stack Pointer
SPI — Serial Peripheral Interface
SR — Channel Status Register
SRA — Channel A Status Register
SRAM — Static Random Access Memory

MC68349 ACRONYM LIST (Concluded)

SSP — Supervisor Stack Pointer
SSW — Software Status Word
SWSR — Software Service Register
SYNCR — Clock Synthesizer Control Register
SYPCR — System Protection Control Register
TAP — Test Access Port
TB — Transmitter Buffer
USART — Universal Synchronous/Asynchronous Receiver/Transmitter
VBR — Vector Base Register
VCCGND — System Power Supply and Ground
VCO — Voltage Controlled Oscillator
WAREG/WDREG — Write A/D Register
WSREG — Write System Register

TABLE OF CONTENTS

TABLE OF CONTENTS (Continued)

TABLE OF CONTENTS (Continued)

TABLE OF CONTENTS (Continued)

TABLE OF CONTENTS (Continued)

SECTION 5
CPU030

TABLE OF CONTENTS (Continued)

TABLE OF CONTENTS (Continued)

TABLE OF CONTENTS (Continued)

TABLE OF CONTENTS (Continued)

TABLE OF CONTENTS (Continued)

TABLE OF CONTENTS (Continued)

TABLE OF CONTENTS (Continued)

TABLE OF CONTENTS (Concluded)

LIST OF ILLUSTRATIONS

LIST OF ILLUSTRATIONS (Continued)

LIST OF ILLUSTRATIONS (Continued)

LIST OF ILLUSTRATIONS (Continued)

LIST OF ILLUSTRATIONS (Concluded)

LIST OF TABLES

LIST OF TABLES (Continued)

SECTION 1
OVERVIEW

The MC68349 Dragon I™ is the highest performance member of the Motorola M68300 family of integrated processors. The MC68349 is designed to serve as the central processor of personal intelligent communicators and similar products requiring an optimal balance of performance, integration, cost, and power consumption. The MC68349 is the first in a series of M68300 family integrated processors designed specifically to support rigorous requirements of consumer-oriented intelligent personal electronics.

The MC68349 (shown in Figure 1-1) is based on the powerful CPU030 processor which combines a full 32-bit central processor with a configurable instruction cache and dedicated data memory structures. The MC68349 also incorporates a high-speed 32-bit dual direct memory access (DMA) controller, a two-channel serial communications interface, and clock synthesis, power management functions, system protection features, and a glueless memory interface.

The MC68349 is fully compatible with previous members of the M68000 family and offers a long-term future migration path to successors based on MC68040 and MC68060 processors. The modular nature of the MC68349 also provides for cost-effective migration paths to higher levels of integration in future products.

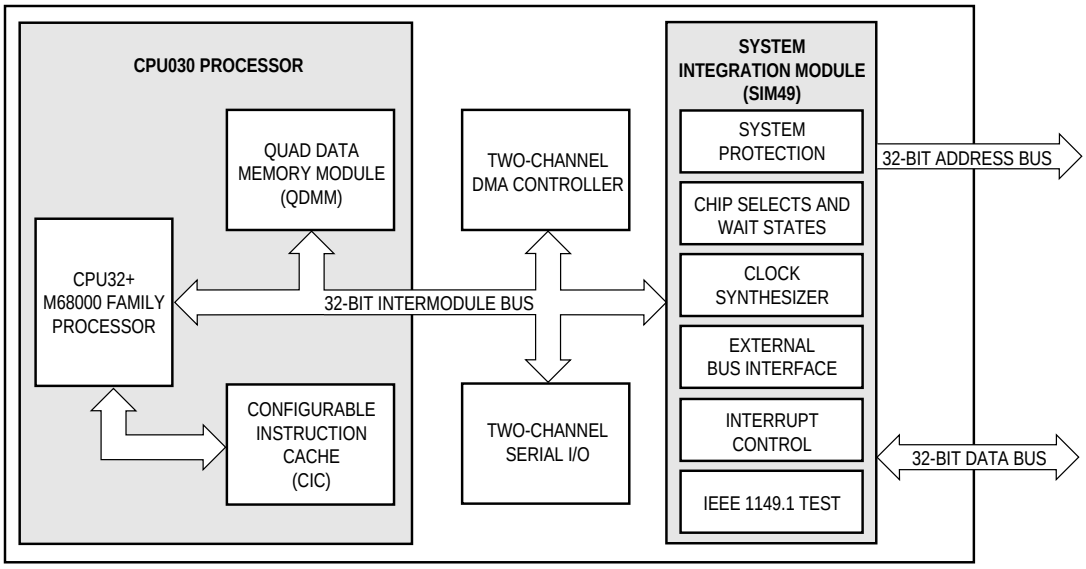


Figure 1-1. MC68349 Block Diagram

Dragon I is a trademark of Motorola.

1.1 FEATURES

The primary features of the MC68349 include:

- High-Performance CPU030 Processor
  - —CPU32+ 32-Bit Execution Unit
    - M68000 Upward User Code Compatible
    - 32-Bit Bus Interface
    - Two-Clock Basic Instruction Execution Rate
  - —Configurable Instruction Cache (CIC)
    - Four Independent Blocks, Each Configurable as Cache or SRAM
    - 1-Kbyte Instruction Cache or 2-Kbyte SRAM Total Storage
    - Four-Way Set Associative (All Blocks Configured as Cache)
    - Each Block Independently Lockable
    - Supports Simultaneous CPU and DMA Bus Activity
  - —Quad Data Memory Module (QDMM)
    - Four Independent 1-Kbyte SRAM Blocks
    - Useful for Variable Storage, Stacks, and Interrupt Handlers
    - Each Block Independently Mapped and Protected
- High-Speed 32-Bit Dual DMA Controllers for Low-Latency Data Transfers
  - —Full 32-Bit Data Transfers for Highest Performance
  - —50-Mbyte/Sec Sustained Transfer Rate @ 25 MHz
  - —Dual or Single Address Transfers
  - —8-, 16-, or 32-Bit Transfers
- Dual Serial Communication Ports
  - —Synchronous or Asynchronous Operation
  - —3-Mbit/Sec Sustained Transfer Rate @ 25 MHz
  - —Modem Control
  - —Baud Rate Generation
  - —MC68681/MC2681 Compatible
- System Integration Module for Flexible and Cost-Effective System Interface
  - —32-Bit Address Bus
  - —32-Bit Data Bus with Dynamic Bus Sizing
  - —System Protection, Reset, and Configuration Control
  - —Chip-Select, Wait State Generation, Bus Watchdog
  - —Periodic Interrupt/System Timer
  - —Interrupt Controller
  - —Dual 8-Bit Parallel Ports
  - —IEEE 1149.1 Boundary Scan (JTAG)

- Power Management
  —5 V or 3.3 V Operation
  —Fully Static HCMOS Technology
  —Programmable Clock Synthesizer for Full Frequency Control
  —Power-Down/Low Power Stop Capabilities
  —Idle Modules Can Be Individually Powered Down
- 32-Bit Data Paths for On-Chip and Off-Chip Access
- 0–16- or 25-MHz Operation
- 160-Pin Plastic QFP

NOTE

As a low voltage part, the MC68349V can operate with a 3.3-V power supply. MC68349 is used throughout this manual to refer to both the low voltage and standard 5-V parts since both are functionally equivalent.

1.2 M68300 FAMILY

The MC68349 is one of a series of components in the M68300 family.

1.2.1 Organization

The M68300 family of integrated processors and controllers is built on an M68000 core processor, an on-chip bus, and a selection of intelligent peripherals appropriate for a set of applications. The CPU030 blends the powerful CPU32+ with on-chip instruction caches and data memory to deliver MC68030 performance with minimal degradation due to slow external memory systems. A system integration module incorporates the external bus interface and many of the smaller circuits that typically surround a microprocessor for address decoding, wait-state insertion, interrupt prioritization, clock generation, arbitration, watchdog timing, and power-on reset timing.

Each member of the M68300 family is distinguished by its selection of peripherals. Peripherals are chosen to address specific applications but are often useful in a wide variety of applications. The peripherals may be highly sophisticated timing or protocol engines that have their own processors, or they may be more traditional peripheral functions, such as universal asynchronous receivers/transmitters (UARTs) and timers. Since each major function is designed in a standalone module, each module might be found in many different M68300 family parts. Driver software written for a module on one M68300 part can be used to run the same module that appears on another part.

1.2.2 Advantages

By incorporating so many major features into a single M68300 family chip, a system designer can realize significant savings in design time, power consumption, cost, board space, pin count, and programming. The equivalent functionality can easily require 20 separate components. Each component might have 16–64 pins, totaling over 350 connections. Most of these connections require interconnects or are duplications. 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 to get the signal to another chip. The cumulative power consumption of all the components must be available from the power supply. The signals between the cental processing unit (CPU) and a peripheral might not be compatible nor run from the same clock, requiring time delays or other special design considerations.

In an M68300 family component, the major functions and glue logic are all properly connected internally, timed with the same clock, fully tested, and uniformly documented. Power consumption stays well under a watt, and a special standby mode drops current well under a milliamp during idle periods. Only essential signals are brought out to pins. The primary package is the surface-mount quad flat pack for the smallest possible footprint.

1.3 CPU030 CENTRAL PROCESSOR UNIT

Processing power for the MC68349 is provided by the CPU030 central processing unit. The CPU030 delivers 32-bit performance in a modular form factor that is highly optimized for the needs of portable intelligent personal electronics applications.

The CPU030 employed in the MC68349 is composed of three principle modular elements: the CPU32+ 32-bit processor, a configurable instruction cache, and a quad data memory module. Memory management is an option not supported in the MC68349.

1.3.1 CPU32+

The CPU32+ processor is the full 32-bit extension of the CPU32 processor found on many M68300 family of integrated processors and provides the execution units for the CPU030. The CPU32+ is a 32-bit execution unit with 32-bit data paths (internal and external), and has a two-clock basic instruction execution rate for a 32-bit operation. The CPU32+ is completely upward software compatible with the MC68000 and CPU32.

In addition to performing basic instruction execution, the CPU32+ also supervises power management capabilities of the MC68349 as well as providing a sophisticated background debug port for software development and debug environments.

1.3.2 Configurable Instruction Cache (CIC)

The configurable instruction cache is a highly configurable memory resource designed to optimize the supply of instructions to the CPU32+. The CIC is closely coupled with the CPU32+, yet isolated from the IMB, allowing the processor to continue instruction execution while a DMA controller or other bus master occupies the external bus. The CIC also moderates power consumption by reducing off-chip accesses.

The CIC is composed of four identical blocks each of which can be independently configured as a 256-byte instruction cache or as a 512-byte static random access memory (SRAM). Either instructions or operands can be stored in the SRAM. When configured as SRAM, a block is independently relocatable in the system's address space. When configured as a cache, a block operates as a direct-mapped cache of sixty-four 32-bit entries. When multiple blocks are configured as caches, set-associativity is supported up to a maximum of four sets. Each cache block can be independently locked to freeze its internal contents.

Using CIC flexibility, it is feasible to place high-priority interrupt or operating system routines in protected SRAM, which is always instantly available, while allowing remaining blocks to operate as caches to increase the performance of general system and user tasks. The CIC configuration can be established at system initialization or on a task-by-task basis, yielding maximum flexibility.

1.3.3 Quad Data Memory Module (QDMM)

The QDMM provides dedicated data storage resources for the CPU030 in the form of four independent 1-Kbyte SRAM blocks. Each of these blocks can be independently relocated anywhere in the system address space, and each is independently protected (supervisor/user, read/write).

The QDMM can be used as scratchpad memory, stack caches for independent tasks, buffers for I/O operations, or parameter storage. The QDMM provides significant performance and power-management benefits to MC68349-based systems. As general SRAM, this space can also contain instructions for fast access to additional interrupt handlers, operating system code, algorithms, or other frequently accessed routines. A total of 4096 bytes of SRAM is provided in the QDMM.

1.4 ON-CHIP PERIPHERALS

To improve total system throughput and reduce part count, board size, and cost of system implementation, the M68300 family integrates on-chip, intelligent peripheral modules and typical glue logic. These functions on the MC68349 include the SIM49, a DMA controller, and a serial module.

The processor communicates with these modules over the on-chip intermodule bus (IMB). This backbone of the chip is similar to traditional external buses with address, data, clock, interrupt, arbitration, and handshake signals. Because bus masters (like the CPU32+ and DMA), peripherals, and the SIM49 are all on the chip, the IMB ensures that

communication between these modules is fully synchronized and that arbitration and interrupts can be handled in parallel with data transfers, greatly improving system performance. The IMB on the MC68349 is 32 bits wide, allowing transfers of 32 bits of data in a single bus cycle. Internal accesses across the IMB may be monitored from outside the chip, if desired.

Each module operates independently. Modules and their registers are accessed in the memory map of the CPU32+ (and DMA) for easy access by general M68000 instructions. Each module may be assigned its own interrupt level, response vector, and arbitration priority. Since each module is a self-contained design and adheres to the IMB interface specifications, the modules may appear on other M68300 family products, retaining the investment in the software drivers for the module.

1.4.1 Direct Memory Access Module

The MC68349 contains a high-speed 32-bit DMA controller, used to quickly move large blocks of data between internal peripherals, external peripherals, or memory without processor intervention. With the MC68349's 32-bit wide external bus, this is an excellent high performance DMA controller to work in many 32-bit microprocessor systems. The DMA module consists of two independent programmable channels. Each channel has separate request, acknowledge, and done signals. Each channel can operate in either single-address (flyby) or dual-address mode and supports 32 bits of address and 8-, 16-, or 32-bit data transfers.

In single-address mode, only one address (the source or the destination) is provided, and a peripheral device such as a serial communications controller receives or supplies the data. An external request must start a single-address transfer.

In dual-address mode, two bus transfers occur, one from a source device and the other to a destination device. Dual-address transfers can be started by either an internal or external request. The source and destination port size can be selected independently.

Byte, word, and long-word counts up to 32 bits can be transferred. All addresses and transfer counters are 32 bits. The DMA channels support both burst and cycle steal external request modes. Internal requests can be programmed to occupy form 25 to 100 percent of the data bus bandwidth. The DMA controller can be configured in all modes to release the bus back to the CPU when a high-priority interrupt occurs.

The DMA module can sustain a transfer rate of 25 Mbytes/sec in dual-address mode and 50 Mbytes/sec in single-address mode @ 25.16 MHz (8.4 and 33.3 Mbytes/sec @ 16.78 MHz). The DMA controller and CPU32+ arbitrate for the bus in parallel with existing bus cycles, typically eliminating all bus arbitration overhead and allowing DMA and CPU bus cycles to occur back-to-back without intervening idle clocks. Three-clock and slower DMA transfers have no arbitration overhead; some fast termination DMA transfers incur a single idle clock before the next CPU access.

1.4.2 Serial Module

Most digital systems use serial I/O to communicate with host computers, operator terminals, or remote devices. The MC68349 contains a two-channel, full-duplex universal synchronous/asynchronous receiver/transmitter (USART). An on-chip baud rate generator provides standard baud rates up to 76.8k baud independently to each channel's receiver and transmitter. The module is functionally equivalent to the MC68681/MC2681 DUART.

Each communication channel is completely independent. Data formats can be 5, 6, 7, or 8 bits with even, odd, or no parity and stop bits up to 2 in 1/16 increments. Four-byte receive buffers and two-byte transmit buffers minimize CPU service calls. A wide variety of error detection and maskable interrupt capability is provided on each channel. Full-duplex, autoecho loopback, local loopback, and remote loopback modes can be selected. Multidrop applications are supported.

A 3.6864-MHz crystal drives the baud rate generators. Each transmit and receive channel can be programmed for a different baud rate, or an external 1× and 16× clock input can be selected. Full modem support is provided with separate request-to-send (RTS) and clear-to-send (CTS) signals for each channel. Channel A also provides service request signals. The two serial ports can sustain rates of 3 Mbps with a 25-MHz system clock in 1× mode, 612 kbps in 16× mode (6.5 Mbps and 410 kbps @ 16.78 MHz).

1.4.3 System Integration Module

The MC68349 SIM49 provides the external bus interface for both the CPU32+ and the DMA. It also eliminates much of the glue logic that typically supports the microprocessor and its interface with the peripheral and memory system. The SIM49 provides programmable circuits to perform address decoding and chip selects, wait-state insertion, interrupt handling, clock generation, bus arbitration, watchdog timing, discrete I/O, and power-on reset timing. A boundary scan test capability is also provided.

1.4.3.1 EXTERNAL BUS INTERFACE. The external bus interface handles the transfer of information between the internal CPU32+ or DMA controller and memory, peripherals, or other processing elements in the external address space. Based on the MC68030 bus, the external bus provides up to 32 address lines and 32 data lines. Address extensions identify each bus cycle as CPU32+ or DMA initiated, supervisor or user privilege level, and instruction or data access. The data bus allows dynamic bus sizing for 8-, 16-, or 32-bit memory devices. Synchronous transfers from the CPU32+ or the DMA can be made in as little as two clock cycles. Asynchronous transfers allow the memory system to signal the CPU32+ or DMA when the transfer is complete and to signal the transfer size. An external master can arbitrate for the bus using a three-wire handshaking interface.

1.4.3.2 SYSTEM CONFIGURATION AND PROTECTION. To achieve maximum system protection, the MC68349 provides system configuration and various monitors and timers as part of the SIM49. The SIM49 contains basic power-on reset circuitry as well as a bus monitor which ensures that the system does not lock up when there is no response to a memory access. When a catastrophic bus failure occurs, the SIM49 bus

fault monitor can reset the processor. The software watchdog timer can pull the processor out of an infinite loop. A periodic interrupt timer can be sent to the CPU32+ for dynamic random access memory (DRAM) refresh, time-of-day clock, task switching, etc.

1.4.3.3 CLOCK SYNTHESIZER. The clock synthesizer generates the clock signals used by all internal operations as well as a clock output used by external devices. The clock synthesizer can operate with an inexpensive 32.768-kHz watch crystal or an external oscillator for reference, using an internal phase-locked loop and voltage-controlled oscillator. At any time, software can select clock frequencies from 131 kHz to the maximum frequency rate, favoring either low power consumption or high performance. Alternately, an external clock can drive the clock signal directly at the operating frequency. With its fully static high-density complimentary metal oxide semiconductor (HCMOS) design, it is possible to completely stop the system clock without losing the contents of the internal registers.

1.4.3.4 CHIP SELECT AND WAIT STATE GENERATION. Four programmable chip selects provide signals to enable external memory and peripheral circuits. Address space and write protection can be selected for each. The block size can be selected from 256 bytes up to 4 Gbytes in increments of 2^n. Accesses can be preselected for either 8-, 16-, or 32-bit transfers. Fast synchronous termination or up to six wait states can be programmed. External handshakes can also signal the end of a bus transfer. A system can boot from reset out of 8-, 16- , or 32-bit-wide memory.

1.4.3.5 INTERRUPT HANDLING. Up to seven discrete interrupt inputs are available for external interrupt sources. Each interrupt can be configured to either autovector to an interrupt handler or generate an external interrupt acknowledge to allow the external device to provide the handler vector. Decoded interrupt acknowledge signals are also available for each interrupt input.

1.4.3.6 DISCRETE I/O PINS. When not used for other functions, 16 pins can be programmed as discrete input or output signals. Additionally, in other peripheral modules, pins for otherwise unused functions can often be used for general input/output.

1.4.3.7 IEEE 1149.1 TEST ACCESS PORT. To aid in system diagnostics, the MC68349 includes dedicated user-accessible test logic that is fully compliant with the IEEE 1149.1 standard for boundary scan testability, often referred to as JTAG (Joint Test Action Group).

1.5 POWER MANAGEMENT

Power consumption on the M68300 family parts is low overall because more of a system's circuitry is on a single piece of silicon, minimizing capacitances and buffering. The MC68349V operates from a 3.3-V power source, saving over 50% of the power required to operate the 5-V MC68349. The MC68349 is implemented in a low-power HCMOS process, assuring high performance with low power consumption. All circuits on the MC68349 are implemented using static circuits, maintaining stable operation for clocking frequencies down to zero.

The programmer can further control power consumption dynamically, as the demand on the processor fluctuates with system requirements and program execution. The on-chip lock circuitry can be varied under software control from 131 kHz to the full rated speed of the chip. Individual peripheral circuits can be turned off when they are not needed. A special low power stop (LPSTOP) instruction drops power consumptions three orders of magnitude as it halts all operation of the chip except for logic needed to reawaken the processor.

1.6 APPLICATIONS AREAS

The following paragraphs discuss some of the applications for which the MC68349 is ideal.

1.6.1 Personal Intelligent Communications

Representing a new class of consumer electronics, the personal intelligent communicator is oriented towards providing maximum personal productivity to its user in a form-factor emphasizing low cost, minimal physical size, extended operation on batteries, seamless communications in a complex wireless/wireline environment, and extreme ease-of-use.

The solution to the needs of the personal intelligent communicator applications requires a carefully selected balance of performance and integration to satisfy the system's functional requirements while also meeting rigorous cost, weight, and power consumption. The MC68349 was developed to the specifications of General Magic and its Alliance Founders and, coupled with the proprietary Astro™ circuit, provides the complete digital core of a personal intelligent communicator.

The MC68349 represents a carefully selected balance of capabilities oriented towards enabling the personal intelligent communicator vision:

- Advanced Motorola CMOS Process for Proven Reliability and Low Cost
- Static Design and Extensive Power Management for Long Battery Life
- Highly Integrated for Low Power, Low System Cost and High Reliability
- CPU030 Provides 8 to 10 Sustained MIPS for Support of Advanced User Interfaces
- Integration of Key System Functions for Performance/Power Optimizations
- M68000 Architecture Provides Strong Software Base and Upward Migration Paths

1.6.2 Additional Application Areas

While specifically optimized for the needs of personal intelligent communications, the MC68349 was partitioned to provide the general-purpose elements of the personal intelligent communicator application while highly specialized features were placed in a proprietary companion circuit. Thus, the MC68349 can provide substantial benefit to other application areas as well.

Astro is a trademark of Motorola.

In general, applications requiring moderate performance, low power consumption, high-bandwidth DMA, and cost-effectiveness can be well-serviced by the MC68349. In addition, since the MC68349 is a member of the M68000 family, its users have access to a broad range of superior support tools and services to facilitate rapid system development and deployment.

Application areas suited to the MC68349 include:

- Personal Digital Assistants (PDAs)
- I/O Processors for High-Performance Systems
- Real-Time Control Processors
- Sophisticated 32-Bit DMA Controllers

1.7 MORE INFORMATION

The following table lists documentation (available through the Literature Distribution Center) related to the MC68349:

Table 1-1. MC68349-Related Documentation

Document Title	Order Number
M68300 Integrated Processor Family	BR1114/D
MC68349 Product Brief	MC68349/D
M68000 Family Programmer's Reference Manual	M68000PM/AD
DRAM Controller for the MC68340	AN1063/D
Software Implementation of SPI on the MC68340	AN453
The 68K Source	BR729/D
3.3 Volt Logic and Interface Circuits	BR1407/D

SECTION 2
SIGNAL DESCRIPTIONS

This section contains brief descriptions of the MC68349 input and output signals in their functional groups as shown in Figure 2-1.

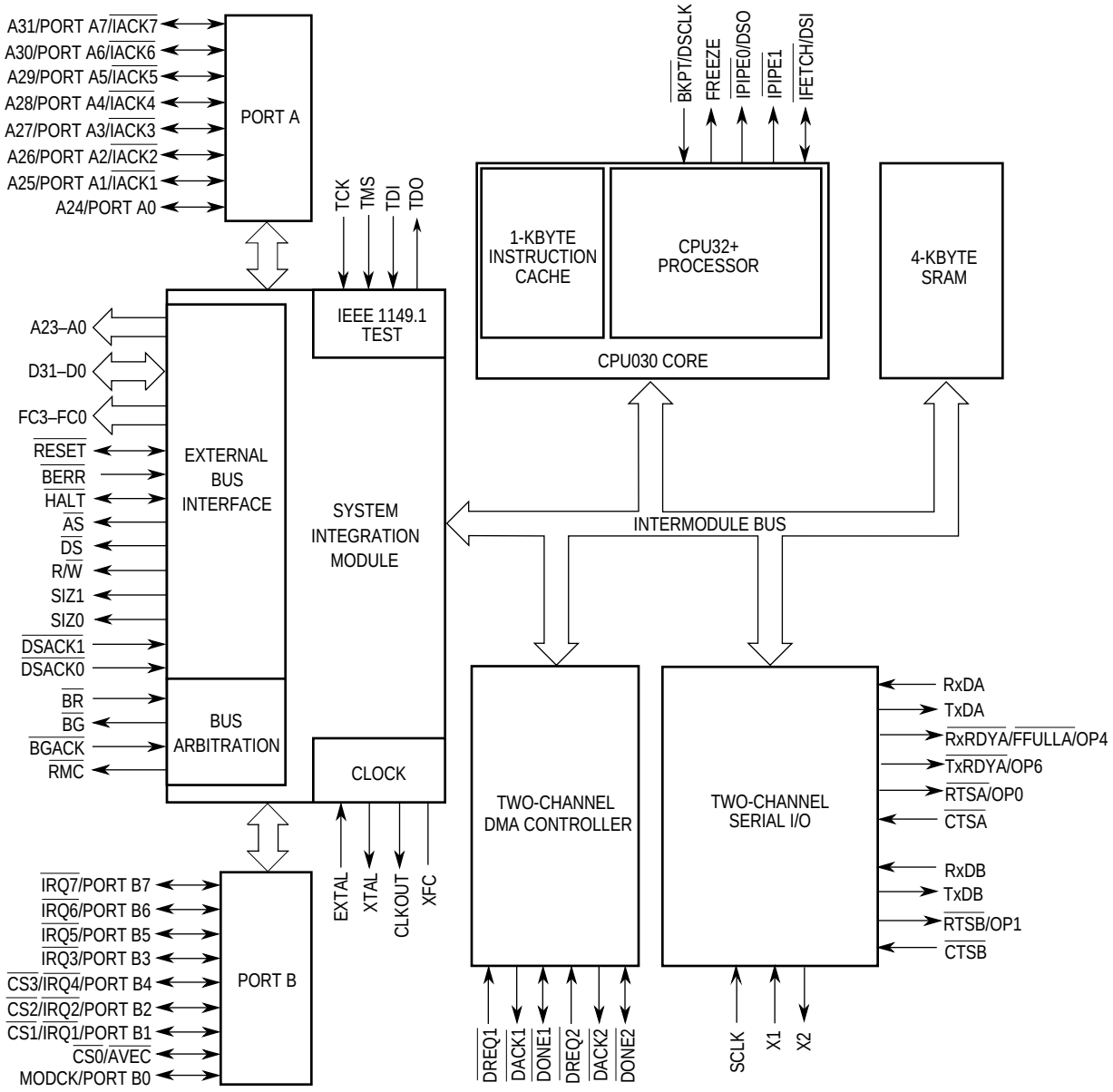


Figure 2-1. Functional Signal Groups

2.1 SIGNAL INDEX

The input and output signals for the MC68349 are listed in Table 2-1. The name, mnemonic, and brief functional description are presented. For more detail on each signal, refer to the signal paragraph. Guaranteed timing specifications for the signals listed in Table 2-1 can be found in **Section 11 Electrical Characteristics**.

Table 2-1. Signal Index

Signal Name	Mnemonic	Function	Input/ Output
Address Bus Bits 23–0	A23–A0	Lower 24 bits of the address bus	Out
Address Bits 31–24/PortA7–0/ Interrupt Acknowledge 7–1	A31–A24	Upper eight bits of the address bus, parallel I/O, or interrupt acknowledge lines	Out/I/O/ Out
Data Bus	D31–D0	32-bit data bus used to transfer byte, word, three-byte, or long-word data	I/O
Function Codes	FC3–FC0	Identify the processor state and the address space of the current bus cycle	Out
Reset	$\overline{\text{RESET}}$	System reset	I/O
Bus Error	$\overline{\text{BERR}}$	Indicates an invalid bus operation is being attempted	In
Halt	$\overline{\text{HALT}}$	Suspends external bus activity	I/O
Address Strobe	$\overline{\text{AS}}$	Indicates that a valid address is on the address bus	Out
Data Strobe	$\overline{\text{DS}}$	During a read cycle, $\overline{\text{DS}}$ indicates that an external device should place valid data on the data bus. During a write cycle, $\overline{\text{DS}}$ indicates that valid data is on the data bus.	Out
Read/Write	R/$\overline{\text{W}}$	Indicates the direction of data transfer on the bus	Out
Size	SIZ1, SIZ0	Indicates the number of bytes remaining to be transferred for this cycle	Out
Data and Size Acknowledge	$\overline{\text{DSACK1}}$, $\overline{\text{DSACK0}}$	Provides asynchronous data transfers and dynamic bus sizing	In
Bus Request	$\overline{\text{BR}}$	Indicates that an external device requires bus mastership	In
Bus Grant	$\overline{\text{BG}}$	Indicates that current bus cycle is complete and the MC68349 has relinquished the bus	Out
Bus Grant Acknowledge	$\overline{\text{BGACK}}$	Indicates that an external device has assumed bus mastership	In
Read-Modify-Write Cycle	$\overline{\text{RMC}}$	Identifies the bus cycle as part of an indivisible read-modify-write operation	Out
Interrupt Request Levels 7, 6, 5, 3/ PortB7, 6, 5, 3	$\overline{\text{IRQ7}}$, $\overline{\text{IRQ6}}$, $\overline{\text{IRQ5}}$, $\overline{\text{IRQ3}}$	Provides an interrupt priority level to the CPU32+ or becomes a parallel I/O port	In/I/O
Chip Select 3–1/Interrupt Request Levels 4, 2, 1/PortB4, 2, 1	$\overline{\text{CS3}}$–$\overline{\text{CS1}}$	Enables peripherals at programmed addresses, interrupt priority level to the CPU32+, or parallel I/O port	Out/In/ I/O
Chip Select 0/Autovector	$\overline{\text{CS0}}$	Enables peripherals at programmed addresses or requests an automatic vector	Out/In

Table 2-1. Signal Index (Continued)

Signal Name	Mnemonic	Function	Input/ Output
Clock Mode Select/Port B0	MODCK/PORTB0	Selects the source of the internal system clock upon reset or becomes a parallel I/O port	In/I/O
Crystal Oscillator	EXTAL, XTAL	Connections for an external crystal or oscillator to the internal oscillator circuit	In, Out
System Clock	CLKOUT	System clock out	Out
External Filter Capacitor	XFC	Connection pin for an external capacitor to filter the circuit of the phase-locked loop	In
Test Clock	TCK	Provides a clock for IEEE 1149.1 test logic	In
Test Mode Select	TMS	Controls test mode operations	In
Test Data In	TDI	Shifts in instructions and test data	In
Test Data Out	TDO	Shifts out instructions and test data	Out
Breakpoint/Development Serial Clock	$\overline{\text{BKPT}}$	Signals a hardware breakpoint to the CPU32+ or provides background debug mode serial clock	In/—
Freeze	FREEZE	Indicates that the CPU32+ has entered background debug mode	Out
Instruction Pipe 0/Development Serial Out	$\overline{\text{IPIPE0}}$	Used to track movement of words through the instruction pipeline or provides background debug mode serial out	Out/Out
Instruction Pipe 1	$\overline{\text{IPIPE1}}$	Used to track movement of words through the instruction pipeline	Out
Instruction Fetch/ Development Serial In	$\overline{\text{IFETCH}}$	Indicates when the CPU32+ is performing an instruction word prefetch and when the instruction pipeline has been flushed or provides background debug mode serial in	Out/In
DMA Request	$\overline{\text{DREQ2}}$, $\overline{\text{DREQ1}}$	Input for requesting a DMA transfer	In
DMA Acknowledge	$\overline{\text{DACK2}}$, $\overline{\text{DACK1}}$	Output that signals an access during DMA	Out
DMA Done	$\overline{\text{DONE2}}$, $\overline{\text{DONE1}}$	Bi-directional signal that indicates the last transfer	I/O
Serial Clock	SCLK	External serial clock input	In
Serial Crystal Oscillator	X1, X2	Connections for an external crystal to the serial module internal oscillator circuit	—
Receive Data	RxDA, RxDB	Receiver serial data input	In
Transmit Data	TxDA, TxDB	Transmitter serial data output	Out
Receiver Ready/ FIFO Full/Output 4	$\overline{\text{RxRDYA}}$	Indicates receive buffer has a character, the receiver FIFO buffer is full or becomes a parallel output	Out/Out/ Out
Transmitter Ready/Output 6	$\overline{\text{TxRDYA}}$	Indicates transmit buffer has a character or becomes a parallel output	Out/Out

Table 2-1. Signal Index (Concluded)

Signal Name	Mnemonic	Function	Input/ Output
Request-to-Send A/OP0, Request-to-Send B/OP1	$\overline{\text{RTSA}}$, $\overline{\text{RTSB}}$/	Channel request-to-send outputs or discrete outputs	Out/Out
Clear-to-Send	$\overline{\text{CTSA}}$, $\overline{\text{CTSB}}$	Serial module clear-to-send inputs	In
Synchronizer Power	VCCSYN	Quiet power supply to VCO; also used to control synthesizer mode after reset.	—
System Power Supply and Ground	V_{CC}, GND	Power supply and ground to the MC68349	—

NOTE

The terms *assert* and *negate* are used throughout this section to avoid confusion when dealing with a mixture of active-low and active-high signals. The term *assert* or *assertion* indicates that a signal is active or true, independent of the level represented by a high or low voltage. The term *negate* or *negation* indicates that a signal is inactive or false.

2.2 ADDRESS BUS

The address bus signals are outputs that define the address of the byte (or the most significant byte) to be transferred during a bus cycle. The MC68349 places the address on the bus at the beginning of a bus cycle. The address is valid while $\overline{\text{AS}}$ is asserted.

The address bus consists of the following two groups. Refer to **Section 3 Bus Operation** for information on the address bus and its relationship to bus operation.

2.2.1 Address Bus (A23–A0)

These three-state outputs (along with A31–A24) provide the address for the current bus cycle, except in the CPU address space.

2.2.2 Address Bus (A31–A24)

These pins can be programmed as the most significant eight address bits, port A parallel I/O, or interrupt acknowledge signals. These pins can be used for more than one of their multiplexed functions as long as the external demultiplexing circuit properly resolves interaction between the different functions.

A31–A24

These pins can function as the most significant eight address bits.

Port A7–A0

These eight pins can serve as a dedicated parallel I/O port. See **Section 4 System Integration Module** for more information on programming these pins.

$\overline{IACK7}$–$\overline{IACK1}$

The MC68349 asserts one of these pins to indicate the level of an external interrupt during an interrupt acknowledge cycle. Peripherals can use the $\overline{IACKx}$ signals instead of monitoring the address bus and function codes to determine that an interrupt acknowledge cycle is in progress and to obtain the current interrupt level.

2.3 DATA BUS (D31–D0)

This bidirectional, nonmultiplexed, parallel bus contains the data being transferred to or from the MC68349. A read or write operation may transfer 8, 16, 24, or 32 bits of data in one bus cycle. During a read cycle, the data is latched by the MC68349 on the last falling edge of the clock for that bus cycle. For a write cycle, all 32 bits of the data bus are driven, regardless of the port width or operand size. The MC68349 places the data on the data bus approximately one-half clock cycle after $\overline{AS}$ is asserted in a write cycle.

2.4 FUNCTION CODES (FC3–FC0)

These signals are outputs that indicate one of 16 address spaces to which the address applies. Fifteen of these spaces are designated as either user or supervisor, program or data, and normal or direct memory access (DMA) spaces. One other address space is designated as CPU space to allow the CPU32+ to acquire specific control information not normally associated with read or write bus cycles. The function code signals are valid while $\overline{AS}$ is asserted. See Table 2-2 for more information.

Table 2-2. Address Space Encoding

Function Code Bits				Address Spaces
3	2	1	0	
0	0	0	0	Reserved (Motorola)
0	0	0	1	User Data Space
0	0	1	0	User Program Space
0	0	1	1	Reserved (User)
0	1	0	0	Reserved (Motorola)
0	1	0	1	Supervisor Data Space
0	1	1	0	Supervisor Program Space
0	1	1	1	CPU Space
1	x	x	x	DMA Space

2.5 EXCEPTION CONTROL SIGNALS

These signals are used by the MC68349 to recover from an exception.

2.5.1 Reset ($\overline{\text{RESET}}$)

This active-low, open-drain, bidirectional signal is used to initiate a system reset. An external reset signal (as well as a reset from the SIM49) resets the MC68349 and all external devices. A reset signal from the CPU32+ (asserted as part of the RESET instruction) resets external devices; the internal state of the CPU32+ is not affected. The on-chip modules are reset, except for the SIM49. However, the module configuration register for each on-chip module is not altered. When asserted by the MC68349, this signal is guaranteed to be asserted for a minimum of 512 clock cycles. Refer to **Section 3 Bus Operation** for a description of reset operation and **Section 5 CPU030** for information about the reset exception.

2.5.2 Bus Error ($\overline{\text{BERR}}$)

This active-low input signal indicates that an invalid bus operation is being attempted or, when used with $\overline{\text{HALT}}$, that the processor should retry the current cycle. Refer to **Section 3 Bus Operation** for a description of the effects of $\overline{\text{BERR}}$ on bus operation.

2.5.3 Halt ($\overline{\text{HALT}}$)

This active-low, open-drain, bidirectional signal is asserted to suspend external bus activity, to request a retry when used with $\overline{\text{BERR}}$, or to perform a single-step operation. As an output, $\overline{\text{HALT}}$ indicates the processor has halted due to a double bus fault. Refer to **Section 3 Bus Operation** for a description of the effects of $\overline{\text{HALT}}$ on bus operation.

2.6 CHIP SELECTS ($\overline{\text{CS3}}$–$\overline{\text{CS0}}$)

These pins can be programmed to be chip select output signals, port B parallel I/O and autovector input, or additional interrupt request lines. Refer to **Section 4 System Integration Module** for more information on these signals.

$\overline{\text{CS3}}$ – $\overline{\text{CS0}}$

The chip select output signals enable peripherals at programmed addresses. These signals are inactive high (not high impedance) after reset. $\overline{\text{CS0}}$ is the chip select for a boot ROM containing the reset vector and initialization program and functions as the boot chip select immediately after reset.

$\overline{\text{IRQ4}}$, $\overline{\text{IRQ2}}$, $\overline{\text{IRQ1}}$

Interrupt request lines are external interrupt lines to the CPU32+. These additional interrupt request lines are selected by the FIRQ bit in the module configuration register.

Port B4, B2, B1, $\overline{\text{AVEC}}$

This signal group functions as three bits of parallel I/O and the autovector input. $\overline{\text{AVEC}}$ requests an automatic vector during an interrupt acknowledge cycle.

2.7 INTERRUPT REQUEST LEVEL ($\overline{IRQ7}$, $\overline{IRQ6}$, $\overline{IRQ5}$, $\overline{IRQ3}$)

These pins can be programmed to be either prioritized interrupt request lines or port B parallel I/O.

$\overline{IRQ7}$, $\overline{IRQ6}$, $\overline{IRQ5}$, $\overline{IRQ3}$

$\overline{IRQ7}$, the highest priority, is nonmaskable. $\overline{IRQ6}$–$\overline{IRQ1}$ are internally maskable interrupts. Refer to **Section 5 CPU030** for more information on interrupt request lines.

Port B7, B6, B5, B3

These pins can be used as port B parallel I/O. Refer to **Section 4 System Integration Module** for more information on parallel I/O signals.

2.8 BUS CONTROL SIGNALS

These signals control the bus transfer operations of the MC68349. Refer to **Section 3 Bus Operation** for more information on these signals.

2.8.1 Address Strobe ($\overline{AS}$)

$\overline{AS}$ is an output timing signal that indicates the validity of both an address on the address bus and many control signals. $\overline{AS}$ is asserted approximately one-half clock cycle after the beginning of a bus cycle.

2.8.2 Data Strobe ($\overline{DS}$)

$\overline{DS}$ is an output timing signal that applies to the data bus. For a read cycle, the MC68349 asserts $\overline{DS}$ and $\overline{AS}$ simultaneously to signal the external device to place data on the bus. For a write cycle, $\overline{DS}$ signals to the external device that the data to be written is valid. The MC68349 asserts $\overline{DS}$ approximately one clock cycle after the assertion of $\overline{AS}$ during a write cycle.

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

This active-high output signal is driven by the bus master to indicate the direction of a data transfer on the bus. A logic one indicates a read from a slave device; a logic zero indicates a write to a slave device.

2.8.4 Transfer Size (SIZ1, SIZ0)

These output signals are driven by the bus master to indicate the number of operand bytes remaining to be transferred in the current bus cycle as noted in Table 2-3.

Table 2-3. SIZx Signal Encoding

SIZ1	SIZ0	Transfer Size
0	0	Long Word
0	1	Byte
1	0	Word
1	1	Three Byte

2.8.5 Data and Size Acknowledge ($\overline{\text{DSACK1}}$, $\overline{\text{DSACK0}}$)

These two active-low input signals allow asynchronous data transfers and dynamic data bus sizing between the MC68349 and external devices as listed in Table 2-4. During bus cycles, external devices assert $\overline{\text{DSACK1}}$ and/or $\overline{\text{DSACK0}}$ as part of the bus protocol. During a read cycle, this assertion signals the MC68349 to terminate the bus cycle and to latch the data. During a write cycle, this assertion indicates that the external device has successfully stored the data and that the cycle may terminate.

Table 2-4. $\overline{\text{DSACKx}}$ Encoding

$\overline{\text{DSACK1}}$	$\overline{\text{DSACK0}}$	Result
1	1	Insert Wait States in Current Bus Cycle
1	0	Complete Cycle—Data Bus Port Size Is 8 Bits
0	1	Complete Cycle—Data Bus Port Size Is 16 Bits
0	0	Complete Cycle—Data Bus Port Size Is 32 Bits

2.9 BUS ARBITRATION SIGNALS

The following signals are the bus arbitration control signals used to determine the bus master. Refer to **Section 3 Bus Operation** for more information on these signals.

2.9.1 Bus Request ($\overline{\text{BR}}$)

This active-low input signal indicates that an external device needs to become the bus master.

2.9.2 Bus Grant ($\overline{\text{BG}}$)

Assertion of this active-low output signal indicates that the MC68349 has relinquished the bus.

2.9.3 Bus Grant Acknowledge ($\overline{\text{BGACK}}$)

Assertion of this active-low input indicates that an external device has become the bus master.

2.9.4 Read-Modify-Write Cycle ($\overline{RMC}$)

This output signal identifies the bus cycle as part of an indivisible read-modify-write operation. It remains asserted during all bus cycles of the read-modify-write operation to indicate that bus ownership cannot be transferred.

2.10 CLOCK SIGNALS

These signals are used by the MC68349 for controlling or generating the system clocks. See **Section 4 System Integration Module** for more information on the various clocking methods and frequencies.

2.10.1 System Clock (CLKOUT)

This output signal is the system clock output and is used as the bus timing reference by external devices. CLKOUT can be varied in frequency or slowed in low power stop mode to conserve power.

2.10.2 Crystal Oscillator (EXTAL, XTAL)

These two pins are the connections for an external crystal to the internal oscillator circuit. If an external oscillator is used, it should be connected to EXTAL, with XTAL left open.

2.10.3 External Filter Capacitor (XFC)

This pin is used to add an external capacitor to the filter circuit of the phase-locked loop. The capacitor should be connected between XFC and VCCSYN.

2.10.4 Clock Mode Select (MODCK)

This pin selects the source of the internal system clock during reset. After reset, it can be programmed to be port B parallel I/O.

MODCK

The state of this active-high input signal during reset selects the source of the internal system clock. If MODCK is high during reset, the internal voltage-controlled oscillator (VCO) furnishes the system clock in crystal mode. If MODCK is low during reset, an external clock source at the EXTAL pin furnishes the system clock output in external clock mode.

PORTB0

This pin can be used as a port B parallel I/O.

2.11 TEST SIGNALS

The following signals are used with the on-board test logic defined by the IEEE 1149.1 standard. See **Section 9 IEEE 1149.1 Test Access Port** for more information on the use of these signals.

2.11.1 Test Clock (TCK)

This input provides a clock for on-board test logic.

2.11.2 Test Mode Select (TMS)

This input controls test mode operations for on-board test logic.

2.11.3 Test Data In (TDI)

This input is used for serial test instructions and test data for on-board test logic.

2.11.4 Test Data Out (TDO)

This output is used for serial test instructions and test data for on-board test logic.

2.12 DEBUG AND EMULATION SUPPORT SIGNALS

These signals are used for test or software debugging. See **Section 5 CPU030** for more information on these signals and background debug mode.

2.12.1 Breakpoint ($\overline{\text{BKPT}}$)

This pin functions as $\overline{\text{BKPT}}$ in normal operation and as DSCLK in background debug mode.

$\overline{\text{BKPT}}$

This active-low input signal is used to signal a hardware breakpoint to the CPU32+.

DSCLK

This development serial clock input helps to provide serial communications for background debug mode.

2.12.2 Freeze (FREEZE)

Assertion of this active-high output signal indicates that the CPU32+ has acknowledged a breakpoint and has initiated background mode operation.

2.12.3 Instruction Pipe ($\overline{\text{IPIPE0}}$, $\overline{\text{IPIPE1}}$)

The $\overline{\text{IPIPE0}}$ pin functions as $\overline{\text{IPIPE0}}$ in normal operation and as DSO in background debug mode. The $\overline{\text{IPIPE1}}$ pin functions as $\overline{\text{IPIPE1}}$ at all times.

$\overline{\text{IPIPE0}}$, $\overline{\text{IPIPE1}}$

These active-low output signals are used to track movement of words through the instruction pipeline.

DSO

This development serial output signal helps to provide serial communications for background debug mode.

2.12.4 Instruction Fetch ($\overline{\text{IFETCH}}$)

This pin functions as $\overline{\text{IFETCH}}$ in normal operation and as DSI in background debug mode.

$\overline{\text{IFETCH}}$

This active-low output signal indicates when the CPU32+ is performing an instruction word prefetch and when the instruction pipeline has been flushed.

DSI

This development serial input signal helps to provide serial communications for background debug mode.

2.13 DMA MODULE SIGNALS

The following signals are used by the direct memory access (DMA) controller module to provide external device handshaking. See **Section 7 DMA Module** for additional information on these signals.

2.13.1 DMA Request ($\overline{\text{DREQ2}}$, $\overline{\text{DREQ1}}$)

This active-low input is asserted by a peripheral device to request an operand transfer between that peripheral and memory. The assertion of $\overline{\text{DREQx}}$ starts the DMA process. The assertion level in external burst mode is level sensitive; in external cycle steal mode, it is falling-edge sensitive.

2.13.2 DMA Acknowledge ($\overline{\text{DACK2}}$, $\overline{\text{DACK1}}$)

This active-low output is asserted by the DMA to signal to a peripheral that an operand is being transferred in response to a previous transfer request.

2.13.3 DMA Done ($\overline{\text{DONE2}}$, $\overline{\text{DONE1}}$)

This active-low bidirectional signal is asserted by the DMA or a peripheral device during any DMA bus cycle to indicate that the last data transfer is being performed. $\overline{\text{DONEx}}$ is an active input in any mode. As an output, it is only active in external request mode. An external pullup resistor is required even during operation in the internal request mode.

2.14 SERIAL MODULE SIGNALS

The following signals are used by the serial module for data and clock signals. See **Section 8 Serial Module** for more information on these signals.

2.14.1 Serial External Clock Input (SCLK)

This input can be used as the external clock input for channel A or channel B, bypassing the baud rate generator.

2.14.2 Serial Crystal Oscillator (X1, X2)

These pins furnish the connection to a crystal or external clock, which must be supplied when using the baud rate generator. An external clock should be connected to the X1 pin with X2 left floating.

2.14.3 Receive Data (RxDA, RxDB)

These signals are the receiver serial data input for each channel. Data received on this signal is sampled on the rising edge of the clock source, with the least significant bit received first.

2.14.4 Transmit Data (TxDA, TxDB)

These signals are the transmitter serial data output for each channel. The output is held high ('mark' condition) when the transmitter is disabled, idle, or operating in the local loopback mode. Data is shifted out on this signal at the falling edge of the clock source, with the least significant bit transmitted first.

2.14.5 Receiver Ready ($\overline{\text{RxRDYA}}$)

This active-low output signal can be programmed as the channel A receiver ready, channel A first-in-first-out (FIFO) full indicator, or a dedicated parallel output.

$\overline{\text{RxRDYA}}$

When used for this function, this signal reflects the complement of the status of bit 1 of the serial interrupt status register. This signal can be used to control parallel data flow by acting as an interrupt to indicate when the receiver contains a character.

$\overline{\text{FFULLA}}$

When used for this function, this signal reflects the complement of the status of bit 1 of the interrupt status register. This signal can be used to control parallel data flow by acting as an interrupt to indicate when the receiver FIFO is full.

OP4

When used for this function, this output is controlled by bit 4 in the output port data registers.

2.14.6 Transmitter Ready ($\overline{\text{TxRDYA}}$)

This active-low output can be programmed as the channel A transmitter ready status indicator or used as a discrete output.

$\overline{\text{TxRDYA}}$

When used for this function, this signal reflects the complement of the status of bit 2 of the channel A status register. This signal can be used to control parallel data flow by acting as an interrupt to indicate when the transmitter contains a character.

OP6

When used for this function, this output is controlled by bit 6 in the output port data registers.

2.14.7 Request to Send ($\overline{\text{RTSA}}$, $\overline{\text{RTSB}}$)

These active-low signals can be programmed as request-to-send outputs or used as discrete outputs.

$\overline{\text{RTSB}}$, $\overline{\text{RTSA}}$

When used for this function, these signals function as the request-to-send outputs.

OP1, OP0

When used for this function, these outputs are controlled by the value of bit 1 and bit 0, respectively, in the output port data registers.

2.14.8 Clear to Send ($\overline{\text{CTSA}}$, $\overline{\text{CTSB}}$)

These active-low signals can be programmed as the clear-to-send inputs for each channel.

2.15 SYNTHESIZER POWER (VCCSYN)

This pin supplies a quiet power source to the voltage controlled oscillator (VCO) to provide greater frequency stability. It is also used to control the synthesizer mode after reset. See **Section 4 System Integration Module** for more information.

2.16 SYSTEM POWER AND GROUND (V_{CC} AND GND)

These pins provide system power and ground to the MC68349. Multiple pins are provided for adequate current capability. All power supply pins must have adequate bypass capacitance for high-frequency noise suppression.

2.17 SIGNAL SUMMARY

Table 2-5 presents a summary of all the signals discussed in the preceding paragraphs.

Table 2-5. Signal Summary

Signal Name	Mnemonic	Input/Output	Active State	Three-State
Address Bus	A23–A0	Out	—	Yes
Address Bus/Port A7–A0/ Interrupt Acknowledge 7–1	A31–A24	Out/I/O/Out	—/—/Low	Yes
Data Bus	D31–D0	I/O	—	Yes
Function Codes	FC3–FC0	Out	—	Yes
Reset	$\overline{\text{RESET}}$	I/O	Low	No
Bus Error	$\overline{\text{BERR}}$	In	Low	—
Halt	$\overline{\text{HALT}}$	I/O	Low	No
Address Strobe	$\overline{\text{AS}}$	Out	Low	Yes
Data Strobe	$\overline{\text{DS}}$	Out	Low	Yes
Read/Write	R/$\overline{\text{W}}$	Out	High/Low	Yes
Size	SIZ1, SIZ0	Out	—	Yes
Data and Size Acknowledge	$\overline{\text{DSACK1}}$, $\overline{\text{DSACK0}}$	In	Low	—
Bus Request	$\overline{\text{BR}}$	In	Low	—
Bus Grant	$\overline{\text{BG}}$	Out	Low	No
Bus Grant Acknowledge	$\overline{\text{BGACK}}$	In	Low	—
Read-Modify-Write Cycle	$\overline{\text{RMC}}$	Out	Low	Yes
Interrupt Request Level/ Port B7, B6, B5, B3	$\overline{\text{IRQ7}}$, $\overline{\text{IRQ6}}$, $\overline{\text{IRQ5}}$, $\overline{\text{IRQ3}}$	In/I/O	Low/—	—
Chip Select 3/Interrupt Request Level 4, 2, 1/Port B4, B2, B1	$\overline{\text{CS3}}$–$\overline{\text{CS1}}$	Out/In/I/O	Low/Low/—	No
Chip Select 0/Autovector	$\overline{\text{CS0}}$/ $\overline{\text{AVEC}}$	Out/In	Low/Low	No
Clock Mode Select/Port B0	MODCK	In/I/O	—/—	—
Crystal Oscillator	EXTAL, XTAL	In, Out	—	—
System Clock	CLKOUT	Out	—	No
External Filter Capacitor	XFC	—	—	—
Test Clock	TCK	In	—	—
Test Mode Select	TMS	In	High	—
Test Data In	TDI	In	High	—
Test Data Out	TDO	Out	High	—

Table 2-5. Signal Summary (Continued)

Signal Name	Mnemonic (Pinout)	Input/Output	Active State	Three-State
Breakpoint/ Development Serial Clock	$\overline{\text{BKPT}}$/DSCLK	In/In	Low/—	—/—
Freeze	FREEZE	Out	High	No
Instruction Pipe 0/ Development Serial Out	$\overline{\text{IPIPE0}}$/DSO	Out/Out	Low/—	No/—
Instruction Pipe 1	$\overline{\text{IPIPE1}}$	Out	Low	No
Instruction Fetch/ Development Serial In	$\overline{\text{IFETCH}}$/DSI	Out/In	Low/—	No/—
DMA Request	$\overline{\text{DREQ2}}$, $\overline{\text{DREQ1}}$	In	Low	—
DMA Acknowledge	$\overline{\text{DACK2}}$, $\overline{\text{DACK1}}$	Out	Low	No
DMA Done	$\overline{\text{DONE2}}$, $\overline{\text{DONE1}}$	I/O	Low	No
Serial Clock	SCLK	In	—	—
Serial Crystal Oscillator	X1, X2	Out/In	—	—
Receive Data	RxDA, RxDB	In	—	—
Transmit Data	TxDA, TxDB	Out	—	No
Receiver Ready/ FIFO Full/OP4	$\overline{\text{RxRDYA}}$	Out/Out/Out	Low/Low/—	No
Transmitter Ready/OP6	$\overline{\text{TxRDYA}}$	Out/Out	Low/—	No
Request-to-Send/ OP1, OP0	$\overline{\text{RTSB}}$, $\overline{\text{RTSA}}$	Out/Out	Low/—	No
Clear-to-Send	$\overline{\text{CTSA}}$, $\overline{\text{CTSB}}$	In	Low	—
Synthesizer Power	VCCSYN	—	—	—
System Power and Ground	V_{CC}, GND	—	—	—

SECTION 3
BUS OPERATION

This section provides a functional description of the bus, the signals that control it, and the bus cycles provided for data transfer operations. It also describes the error and halt conditions, bus arbitration, and reset operation. Operation of the external bus is the same whether the MC68349 or an external device is the bus master; the names and descriptions of bus cycles are from the viewpoint of the bus master. For exact timing specifications, refer to **Section 11 Electrical Characteristics**.

The MC68349 architecture supports byte, word, and long-word operands allowing access to 8-, 16-, and 32-bit data ports through the use of asynchronous cycles controlled by the SIZ1/SIZ0 outputs and $\overline{\text{DSACK1}}$/$\overline{\text{DSACK0}}$ inputs. The MC68349 also supports misaligned word and long word operands (i.e., word and long word operands not located on word boundaries in memory).

3.1 BUS TRANSFER SIGNALS

The bus transfers information between the MC68349 and external memory or a peripheral device. External devices can accept or provide 8, 16, or 32 bits in parallel and must follow the handshake protocol described in this section. The maximum number of bits accepted or provided during a bus transfer is defined as the port width. The MC68349 contains an address bus that specifies the address for the transfer and a data bus that transfers the data. Control signals indicate the beginning and type of the cycle as well as the address space and size of the transfer. The selected device then controls the length of the cycle with the signal(s) used to terminate the cycle. Strobe signals, one for the address bus and another for the data bus, indicate the validity of the address and provide timing information for the data. Both asynchronous and synchronous operation is possible for any port width. In asynchronous operation, the bus and control input signals are internally synchronized to the MC68349 clock, introducing a delay. This delay is the time required for the MC68349 to sample an input signal, synchronize the input to the internal clocks, and determine whether it is high or low. In synchronous mode, the bus and control input signals must be timed to setup and hold times. Since no synchronization is needed, bus cycles can be completed in three clock cycles in this mode. Additionally, using the fast-termination option of the chip select signals, two-clock operation is possible.

Furthermore, for all inputs, the MC68349 latches the level of the input during a sample window around the falling edge of the clock signal. This window is illustrated in Figure 3-1, where t_{su} and t_h are the input setup and hold times, respectively. To ensure that an input signal is recognized on a specific falling edge of the clock, that input must be stable during the sample window. If an input makes a transition during the window time period, the level recognized by the MC68349 is not predictable; however, the MC68349 always resolves

the latched level to either a logic high or low before using it. In addition to meeting input setup and hold times for deterministic operation, all input signals must obey the protocols described in this section.

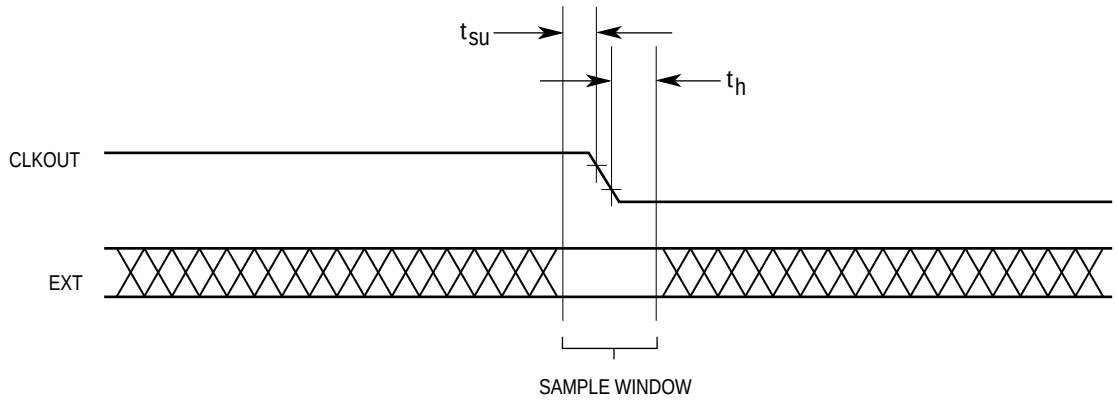


Figure 3-1. Input Sample Window

NOTE

The terms *assert* and *negate* are used throughout this section to avoid confusion when dealing with a mixture of active-low and active-high signals. The term *assert* or *assertion* indicates that a signal is active or true, independent of the level represented by a high or low voltage. The term *negate* or *negation* indicates that a signal is inactive or false.

3.1.1 Bus Control Signals

The MC68349 initiates a bus cycle by driving the A31–A0, SIZx, FCx, and R/$\overline{W}$ outputs. At the beginning of a bus cycle, SIZ1 and SIZ0 are driven with FC3–FC0. SIZ1 and SIZ0 indicate the number of bytes remaining to be transferred during an operand cycle (consisting of one or more bus cycles). Table 3-3 lists the encoding of the SIZx signals. The SIZx signals are valid while $\overline{AS}$ is asserted. The R/$\overline{W}$ signal determines the direction of the transfer during a bus cycle. Driven at the beginning of a bus cycle, R/$\overline{W}$ is valid while $\overline{AS}$ is asserted. R/$\overline{W}$ only transitions when a write cycle is preceded by a read cycle or vice versa. The signal may remain low for consecutive write cycles. The $\overline{RMC}$ signal is asserted at the beginning of the first bus cycle of a read-modify-write operation and remains asserted until completion of the final bus cycle of the operation.

3.1.2 Function Code Signals

FC3–FC0 are outputs that indicate one of 16 address spaces to which the address applies. Fifteen of these spaces are designated as either user or supervisor, program or data, and normal or direct memory access (DMA) spaces. One other address space is designated as CPU space to allow the CPU32+ to acquire specific control information not normally associated with read or write bus cycles. FC3–FC0 are valid while $\overline{AS}$ is asserted.

Function codes (see Table 3-1) can be considered as extensions of the 32-bit address that can provide up to 16 different 4-Gbyte address spaces. Function codes are automatically generated by the CPU32+ to select address spaces for data and program at both user and supervisor privilege levels, a CPU address space for processor functions, and an alternate master address space. User programs access only their own program and data areas to increase protection of system integrity and can be restricted from accessing other information. The S-bit in the CPU32+ status register is set for supervisor accesses and cleared for user accesses to provide differentiation. Refer to **3.4 CPU Space Cycles** for more information.

Table 3-1. Address Space Encoding

Function Code Bits				
3	**2**	**1**	**0**	**Address Spaces**
0	0	0	0	Reserved (Motorola)
0	0	0	1	User Data Space
0	0	1	0	User Program Space
0	0	1	1	Reserved (User)
0	1	0	0	Reserved (Motorola)
0	1	0	1	Supervisor Data Space
0	1	1	0	Supervisor Program Space
0	1	1	1	CPU Space
1	x	x	x	DMA Space

3.1.3 Address Bus (A31–A0)

These signals are outputs that define the address of the byte (or the most significant byte) to be transferred during a bus cycle. The MC68349 places the address on the bus at the beginning of a bus cycle. The address is valid while $\overline{AS}$ is asserted.

3.1.4 Address Strobe ($\overline{AS}$)

This output timing signal indicates the validity of many control signals and the address on the address bus. $\overline{AS}$ is asserted approximately one-half clock cycle after the beginning of a bus cycle.

3.1.5 Data Bus (D31–D0)

This bidirectional, nonmultiplexed, parallel bus contains the data being transferred to or from the MC68349. A read or write operation may transfer 8, 16, or 32 bits of data in one bus cycle. During a read cycle, the data is latched by the MC68349 on the last falling edge of the clock for that bus cycle. For a write cycle, all 32 bits of the data bus are driven, regardless of the port width or operand size. The MC68349 places the data on the data bus approximately one-half clock cycle after $\overline{AS}$ is asserted in a write cycle.

3.1.6 Data Strobe ($\overline{DS}$)

$\overline{DS}$ is an output timing signal that applies to the data bus. For a read cycle, the MC68349 asserts $\overline{DS}$ and $\overline{AS}$ simultaneously to signal the external device to place data on the bus. For a write cycle, $\overline{DS}$ signals to the external device that the data to be written is valid. The MC68349 asserts $\overline{DS}$ approximately one clock cycle after the assertion of $\overline{AS}$ during a write cycle.

3.1.7 Bus Cycle Termination Signals

The following signals can terminate a bus cycle.

3.1.7.1 DATA TRANSFER AND SIZE ACKNOWLEDGE SIGNALS ($\overline{DSACK1}$ AND $\overline{DSACK0}$). During bus cycles, external devices assert $\overline{DSACK1}$ and/or $\overline{DSACK0}$ as part of the bus protocol. During a read cycle, this signals the MC68349 to terminate the bus cycle and to latch the data. During a write cycle, this indicates that the external device has successfully stored the data and that the cycle may terminate. These signals also indicate to the MC68349 the size of the port for the bus cycle just completed (see Table 3-3). Refer to **3.3.1 Read Cycle** for timing relationships of $\overline{DSACK1}$ and $\overline{DSACK0}$.

Additionally, the system integration module (SIM49) chip select address mask register can be programmed to internally generate $\overline{DSACK1}$ and $\overline{DSACK0}$ for external accesses, eliminating logic required to generate these signals. However, if external $\overline{DSACKx}$ signals are returned earlier than indicated by the DD bits in the chip select address mask register, the cycle will terminate sooner than programmed. Refer to **Section 4 System Integration Module** for additional information. The SIM49 can alternatively be programmed to generate a fast termination cycle, providing a two-cycle external access. Refer to **3.2.6 Fast Termination Cycles** for additional information on these cycles.

3.1.7.2 BUS ERROR ($\overline{BERR}$). This signal is a bus cycle termination indicator and can be used in the absence of $\overline{DSACKx}$ to indicate a bus error condition. $\overline{BERR}$ can also be asserted in conjunction with $\overline{DSACKx}$ to indicate a bus error condition, provided it meets the appropriate timing described in this section and in **Section 11 Electrical Characteristics**. Additionally, $\overline{BERR}$ and $\overline{HALT}$ can be asserted together to indicate a retry termination. Refer to **3.5 Bus Exception Control Cycles** for additional information on the use of these signals.

The internal bus monitor can be used to generate an internal bus error signal for internal and internal-to-external transfers. If the bus cycles of an external bus master are to be monitored, external $\overline{\text{BERR}}$ generation must be provided since the internal bus error monitor has no information about transfers initiated by an external bus master.

3.1.7.3 AUTOVECTOR ($\overline{\text{AVEC}}$). This signal can be used to terminate interrupt acknowledge cycles, indicating that the MC68349 should internally generate a vector (autovector) number to locate an interrupt handler routine. $\overline{\text{AVEC}}$ can be generated either externally or internally by the SIM49 (see **Section 4 System Integration Module** for additional information). $\overline{\text{AVEC}}$ is ignored during all other bus cycles.

3.2 DATA TRANSFER MECHANISM

The MC68349 supports byte, word, and long-word operands, allowing access to 8-,16-, and 32-bit data ports through the use of asynchronous cycles controlled by $\overline{\text{DSACK1}}$ and $\overline{\text{DSACK0}}$. The MC68349 also supports byte, word, and long-word operands, allowing access to 8-, 16, and 32-bit data ports through the use of synchronous cycles controlled by the fast-termination capability of the SIM49.

3.2.1 Dynamic Bus Sizing

The MC68349 dynamically interprets the port size of the addressed device during each bus cycle, allowing operand transfers to or from 8-, 16-, and 32-bit ports. During an operand transfer cycle, the slave device signals its port size (byte, word, or long word) and indicates completion of the bus cycle to the MC68349 through the use of the $\overline{\text{DSACKx}}$ inputs. Refer to Table 3-2 for $\overline{\text{DSACKx}}$ encoding.

Table 3-2. $\overline{\text{DSACKx}}$ Encoding

$\overline{\text{DSACK1}}$	$\overline{\text{DSACK0}}$	Result
1	1	Insert Wait States in Current Bus Cycle
1	0	Complete Cycle—Data Bus Port Size is 8 Bits
0	1	Complete Cycle—Data Bus Port Size is 16 Bits
0	0	Complete Cycle—Data Bus Port Size is 32 Bits

For example, if the MC68349 is executing an instruction that reads a long-word operand from a long-word aligned address, it attempts to read 32 bits during the first bus cycle. (Refer to **3.2.2 Misaligned Operands** for the case of a word or byte address.) If the port responds that it is 32 bits wide, the MC68349 latches all 32 bits of data and continues with the next operation. If the port responds that it is 16 bits wide, the MC68349 latches the 16 bits of valid data and runs another bus cycle to obtain the other 16 bits. The operation for an 8-bit port is similar, but requires four read cycles. The addressed device uses the $\overline{\text{DSACKx}}$ signals to indicate the port width. For instance, a 32-bit device always returns $\overline{\text{DSACKx}}$ for a 32-bit port (regardless of whether the bus cycle is a byte, word, or long-word operation).

Dynamic bus sizing requires that the portion of the data bus used for a transfer to or from a particular port size be fixed. A 32-bit port must reside on data bus bits 31–0, a 16-bit port must reside on data bus bits 31–16, and an 8-bit port must reside on data bus bits 31–24. This requirement minimizes the number of bus cycles needed to transfer data to 8- and 16-bit ports and ensures that the MC68349 correctly transfers valid data. The MC68349 always attempts to transfer the maximum amount of data on all bus cycles; for a long-word operation, it always assumes that the port is 32 bits wide when beginning the bus cycle.

The bytes of operands are designated as shown in Figure 3-2. The most significant byte of a long-word operand is OP0, and OP3 is the least significant byte. The two bytes of a word-length operand are OP2 (most significant) and OP3. The single byte of a byte-length operand is OP3. These designations are used in the figures and descriptions that follow.

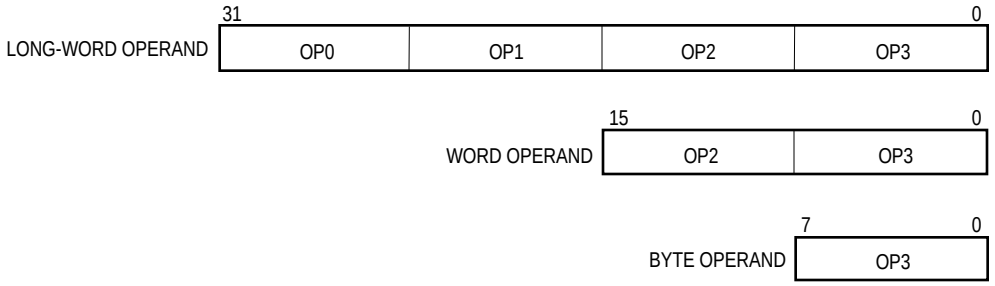


Figure 3-2. Internal Operand Representation

Figure 3-3 shows the required organization of data ports on the MC68349 bus for 8, 16, and 32-bit devices. The four bytes shown are connected through the internal data bus and data multiplexer to the external data bus. This path is the means through which the MC68349 supports dynamic bus sizing and operand misalignment. Refer to **3.2.2 Misaligned Operands** for the definition of misaligned operand. The data multiplexer establishes the necessary connections for different combinations of address and data sizes.

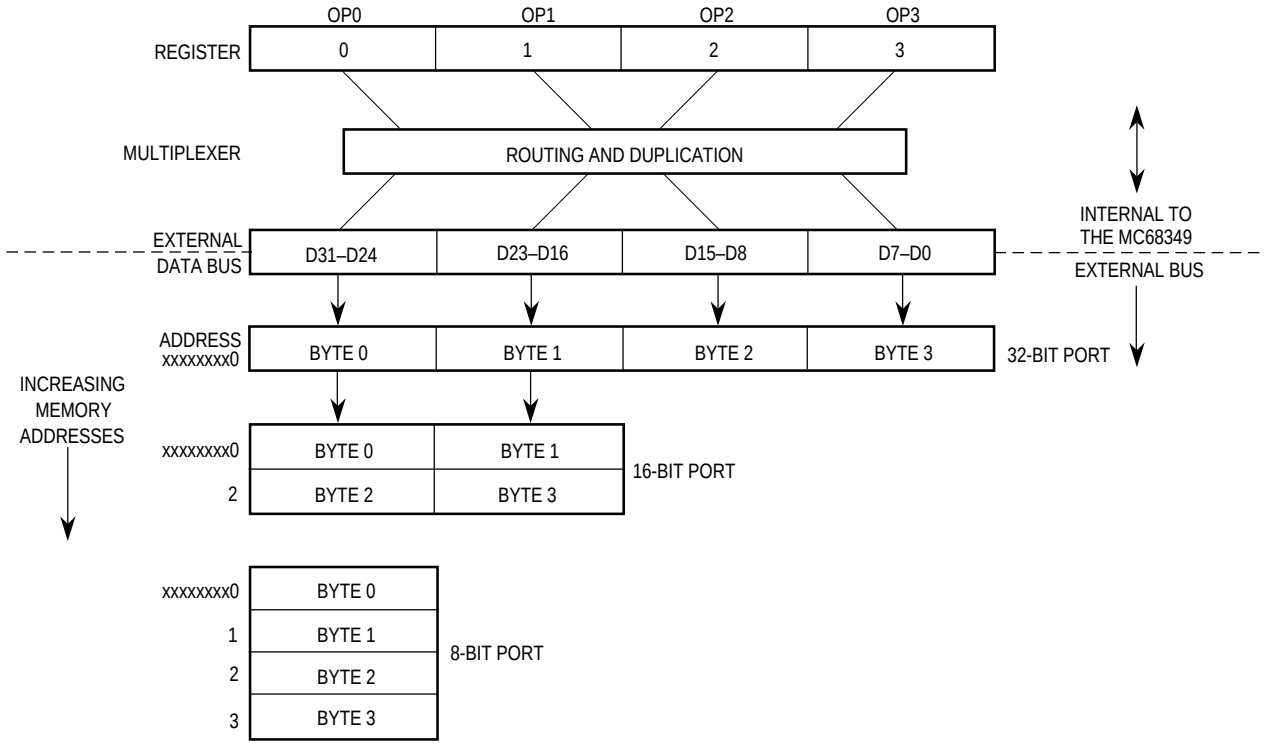


Figure 3-3. MC68349 Interface to Various Port Sizes

The multiplexer takes the four bytes of the 32-bit bus and routes them to their required positions. For example, OP0 can be routed to D24–D31, as would be the normal case, or it can be routed to any other byte position to support a misaligned transfer. The same is true for any of the operand bytes. The positioning of bytes is determined by the size and address outputs.

The SIZ0 and SIZ1 outputs indicate the remaining number of bytes to be transferred during the current bus cycle (see Table 3-3).

Table 3-3. SIZx Encoding

SIZ1	SIZ0	Size
0	1	Byte
1	0	Word
1	1	3 Bytes
0	0	Long Word

The number of bytes transferred during a write or read bus cycle is equal to or less than the size indicated by the SIZx outputs, depending on port width and operand alignment. For example, during the first bus cycle of a long-word transfer to a word port, the SIZx outputs indicate that four bytes are to be transferred, although only two bytes are moved on that bus cycle.

A1 and A0 also affect operation of the data multiplexer. During an operand transfer, A31–A2 indicate the long-word base address of that portion of the operand to be accessed; A1 and A0 indicate the byte offset from the base. Table 3-4 lists the encoding of A1 and A0 and the corresponding byte offset from the long-word base.

Table 3-4. Address Offset Encoding

A1	A0	Offset
0	0	+0 Byte
0	1	+1 Byte
1	0	+2 Bytes
1	1	+3 Bytes

Table 3-5 lists the bytes required on the data bus for read cycles. The entries shown as OPx are portions of the requested operand that are read during that bus cycle and are defined by SIZ1, SIZ0, A1, and A0 for the bus cycle. Bytes labeled x are “don’t cares” and are not required during that read cycle.

When the MC68349 executes code from 32-bit memory with one or more blocks of the configurable instruction cache (CIC) used as instruction cache, the memory subsystem must respond to word-sized code fetches with valid data on both words of the data bus. Although only the instruction word specifically requested is routed to the CPU32+ instruction pipeline, both words on the bus are assumed valid and are cached in the CIC.

Table 3-6 lists the combinations of SIZ1, SIZ0, A1, and A0 and the corresponding pattern of the data transfer for write cycles from the internal multiplexer of the MC68349 to the external data bus. Bytes labeled x are “don't cares.”

Figure 3-4 shows the transfer of a long-word operand to a word port. In the first bus cycle, the MC68349 places the four operand bytes on the external bus. Since the address is long-word aligned in this example, the multiplexer follows the pattern in the entry of Table 4-6 corresponding to SIZ0, SIZ1, A0, A1 = 0000. The port latches the data on bits D16–D31 of the data bus, asserts $\overline{\text{DSACK1}}$ ($\overline{\text{DSACK0}}$ remains negated), and the MC68349 terminates the bus cycle. It then starts a new bus cycle with SIZ0, SIZ1, A0, A1 = 1010 to transfer the remaining 16 bits. SIZ0 and SIZ1 indicate that a word remains to be transferred; A0 and A1 indicate that the word corresponds to an offset of two from the base address. The multiplexer follows the pattern corresponding to this configuration of the size and address signals and places the two least significant bytes of the long word on the word portion of the bus (D16–D31). The bus cycle transfers the remaining bytes to the word-size port. Figure 3-5 shows the timing of the bus transfer signals for this operation.

Table 3-5. Data Bus Requirements for Read Cycles*

Transfer Size	Size		Address		External Data Bytes Required						
					Long-Word Port				Word Port		Byte Port
	SIZ1	SIZ0	A1	A0	D31:D24	D23:D16	D15:D8	D7:D0	D31:D24	D23:D16	D31:D24
Byte	0	1	0	0	OP3	x	x	x	OP3	x	OP3
	0	1	0	1	x	OP3	x	x	x	OP3	OP3
	0	1	1	0	x	x	OP3	x	OP3	x	OP3
	0	1	1	1	x	x	x	OP3	x	OP3	OP3
Word	1	0	0	0	OP2	OP3	x	x	OP2	OP3	OP2
	1	0	0	1	x	OP2	OP3	x	x	OP2	OP2
	1	0	1	0	x	x	OP2	OP3	OP2	OP3	OP2
	1	0	1	1	x	x	x	OP2	x	OP2	OP2
3 Bytes	1	1	0	0	OP1	OP2	OP3	x	OP1	OP2	OP1
	1	1	0	1	x	OP1	OP2	OP3	x	OP1	OP1
	1	1	1	0	x	x	OP1	OP2	OP1	OP2	OP1
	1	1	1	1	x	x	x	OP1	x	OP1	OP1
Long Word	0	0	0	0	OP0	OP1	OP2	OP3	OP0	OP1	OP0
	0	0	0	1	x	OP0	OP1	OP2	x	OP0	OP0
	0	0	1	0	x	x	OP0	OP1	OP0	OP1	OP0
	0	0	1	1	x	x	x	OP0	x	OP0	OP0

* 32-bit memory must supply valid data for the entire 32-bit port width when responding to a word-sized code access

Table 3-6. MC68349 Internal to External Data Bus Multiplexer—Write Cycle

Transfer Size	Size		Address		External Data Bus Connection			
	SIZ1	SIZ0	A1	A0	D31:D24	D23:D16	D15:D8	D7:D0
Byte	0	1	0	0	OP3	x	x	x
	0	1	0	1	OP3	OP3	x	x
	0	1	1	0	OP3	x	OP3	x
	0	1	1	1	OP3	OP3	x	OP3
Word	1	0	0	0	OP2	OP3	x	x
	1	0	0	1	OP2	OP2	OP3	x
	1	0	1	0	OP2	OP3	OP2	OP3
	1	0	1	1	OP2	OP2	x	OP2
3 Bytes	1	1	0	0	OP1	OP2	OP3	x
	1	1	0	1	OP1	OP1	OP2	OP3
	1	1	1	0	OP1	OP2	OP1	OP2
	1	1	1	1	OP1	x	OP2	OP1
Long Word	0	0	0	0	OP0	OP1	OP2	OP3
	0	0	0	1	OP0	OP0	OP1	OP2
	0	0	1	0	OP0	OP1	OP0	OP1
	0	0	1	1	OP0	OP0	x	OP0

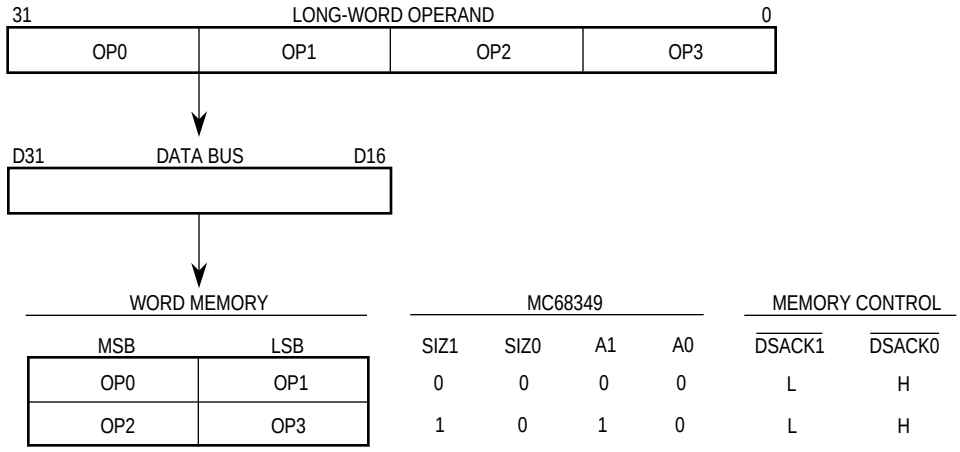


Figure 3-4. Long-Word Operand Write to Word Port Example

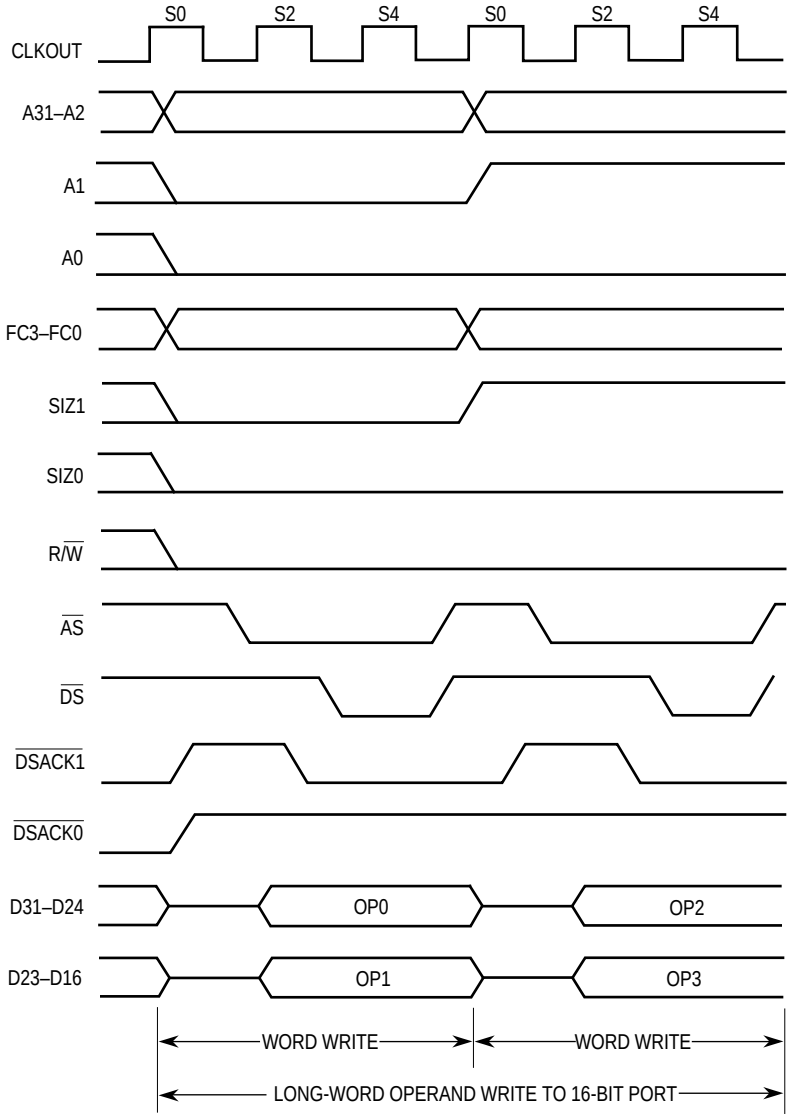


Figure 3-5. Long-Word Operand Write to Word Port Timing

Figure 3-6 shows a word transfer to an 8-bit port. Like the preceding example, this example requires two bus cycles. Each bus cycle transfers a single byte. The size signals for the first cycle specify two bytes; for the second cycle, they specify one byte. Figure 3-7 shows the associated bus transfer signal timing.

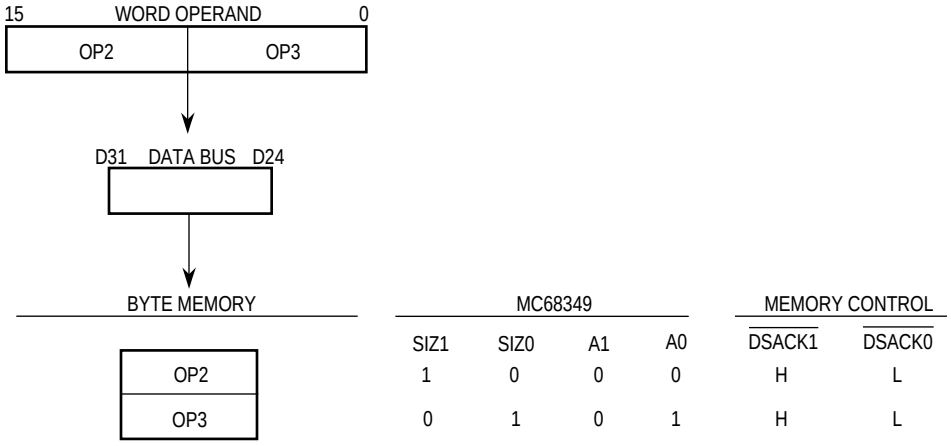


Figure 3-6. Word Operand Write to Byte Port Example

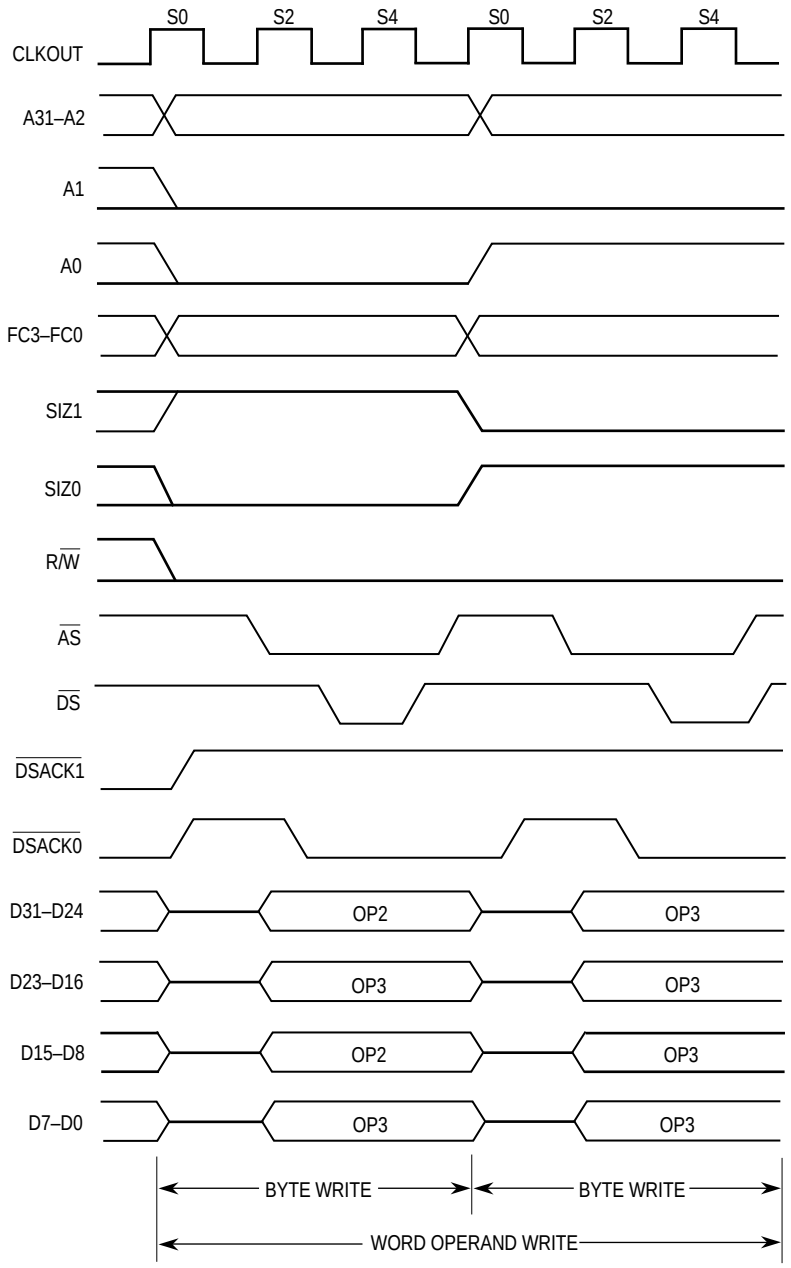


Figure 3-7. Word Operand Write to Byte Port Timing

3.2.2 Misaligned Operands

Since operands may reside at any byte boundaries, they may be misaligned. A byte operand is properly aligned at any address; a word operand is misaligned at an odd address; a long word is misaligned at an address that is not evenly divisible by four. The MC68302, MC68000/MC68008, MC68010, and MC68340 implementations allow long-word transfers on odd-word boundaries but force exceptions if word or long-word operand transfers are attempted at odd-byte addresses. Although the MC68349 does not enforce any alignment restrictions for data operands (including program counter (PC) relative data addresses), some performance degradation occurs when additional bus cycles are

required for long-word or word operands that are misaligned. For maximum performance, data items should be aligned on their natural boundaries. All instruction words and extension words must reside on word boundaries. Attempting to prefetch an instruction word at an odd address causes an address error exception.

Figure 3-8 shows the transfer of a long-word operand to an odd address in word-organized memory, which requires three bus cycles. For the first cycle, the SIZx signals specify a long-word transfer, and the address offset (A2–A0) is 001. Since the port width is 16 bits, only the first byte of the long word is transferred. The slave device latches the byte and acknowledges the data transfer, indicating that the port is 16 bits wide. When the processor starts the second cycle, the SIZx signals specify that three bytes remain to be transferred with an address offset (A2–A0) of 010. The next two bytes are transferred during this cycle. The processor then initiates the third cycle, with the SIZx signals indicating one byte remaining to be transferred. The address offset (A2–A0) is now 100; the port latches the final byte, and the operation is complete. Figure 3-9 shows the associated bus transfer signal timing.

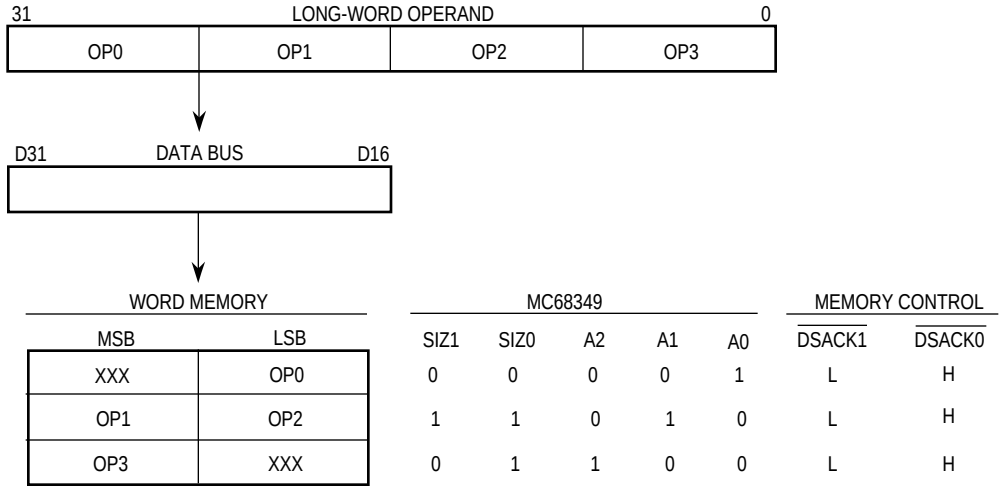


Figure 3-8. Misaligned Long-Word Operand Write to Word Port Example

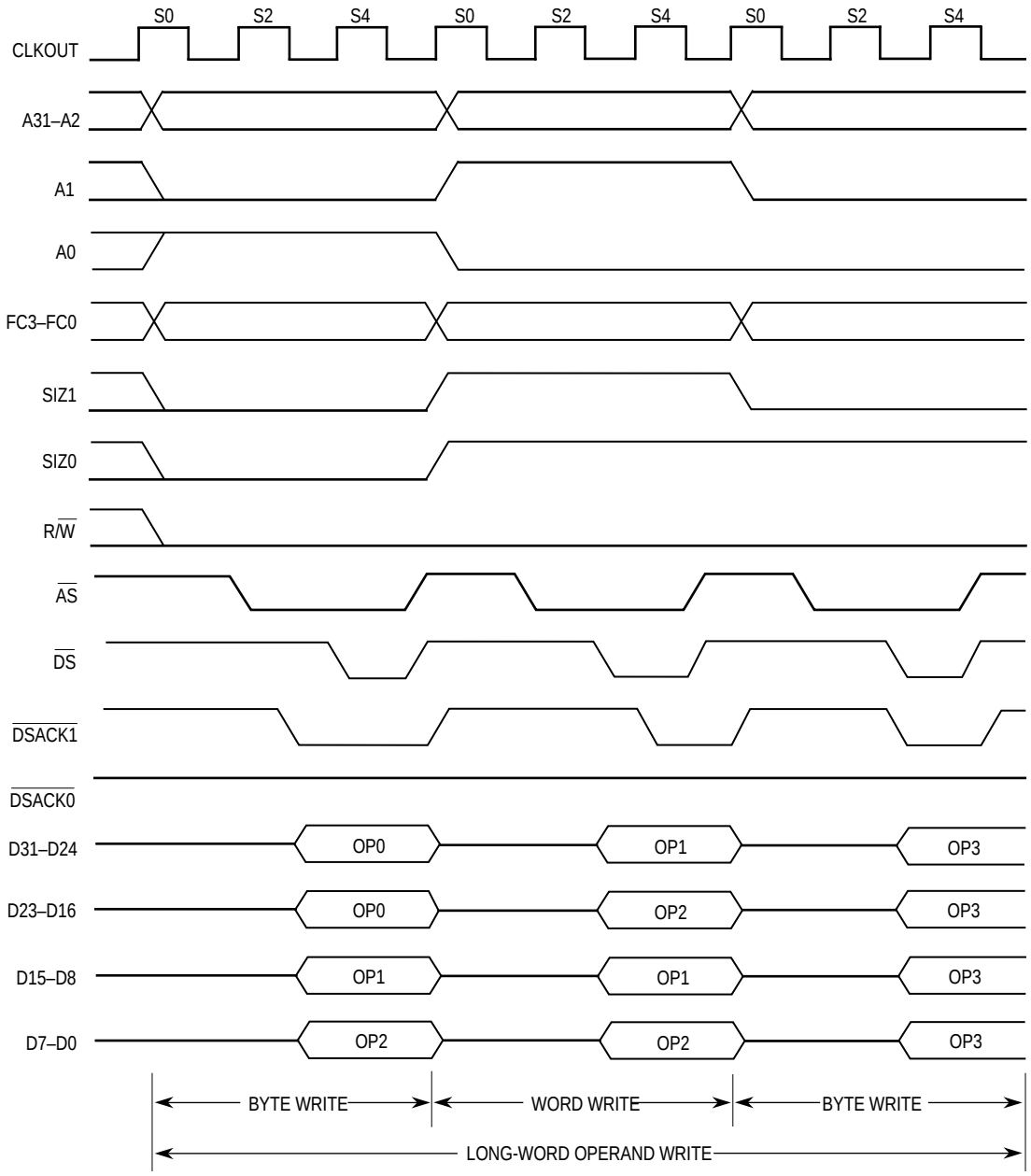


Figure 3-9. Misaligned Long-Word Operand Write to Word Port Timing

Figures 3-10 and 3-11 show a word transfer to an odd address in word-organized memory. This example is similar to the one shown in Figures 3-8 and 3-9 except that the operand is word sized and the transfer requires only two bus cycles.

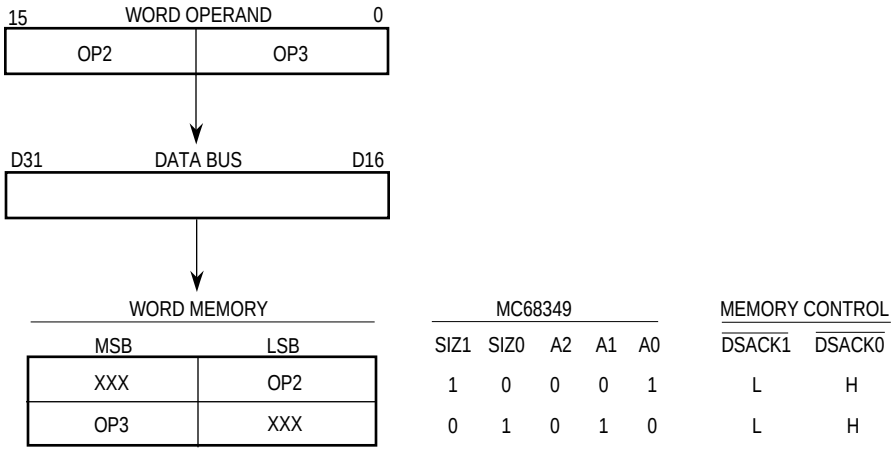


Figure 3-10. Misaligned Word Operand Write to Word Port Example

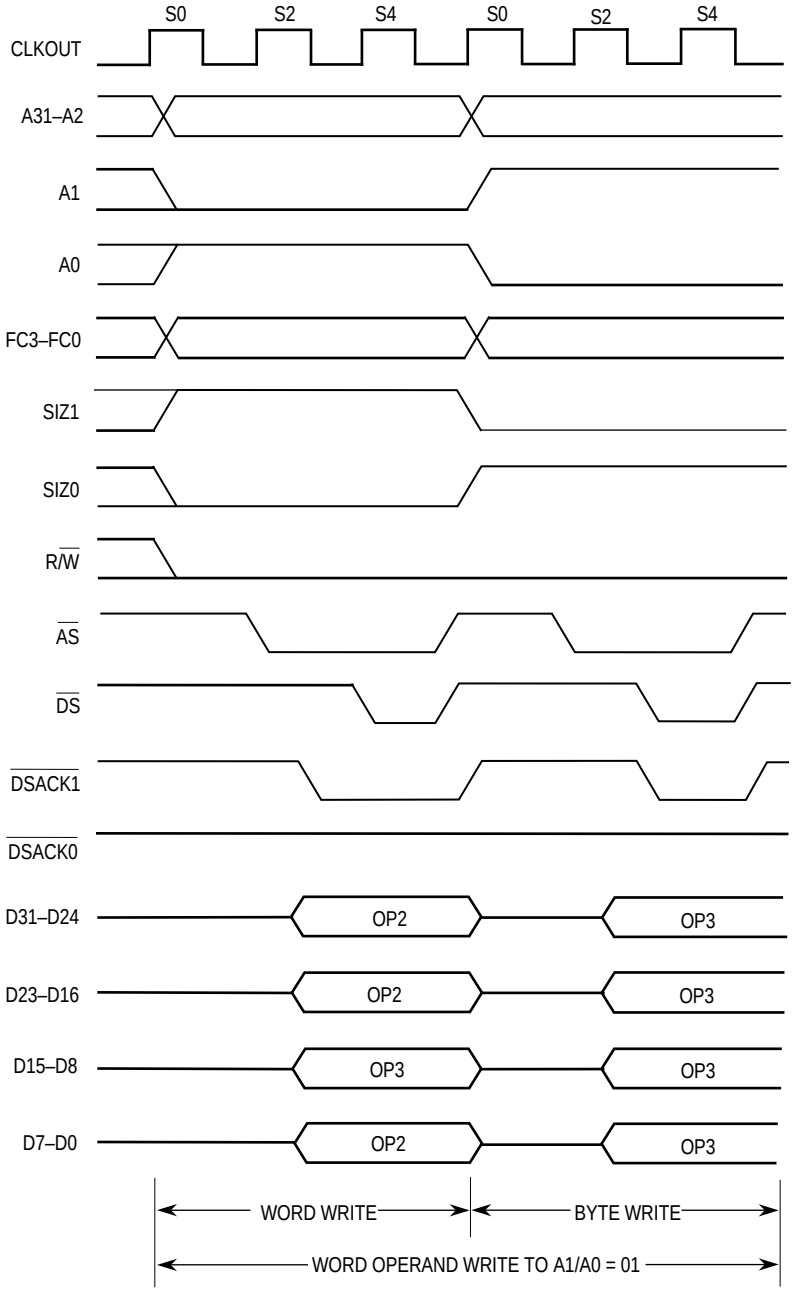


Figure 3-11. Misaligned Word Operand Write to Word Port Timing

Figures 3-12 and 3-13 show an example of a long-word transfer to an odd address in long-word-organized memory. In this example, a long-word access is attempted beginning at the least significant byte of a long-word-organized memory. Only one byte can be transferred in the first bus cycle. The second bus cycle then consists of a three-byte access to a long-word boundary. Since the memory is long-word organized, no further bus cycles are necessary.

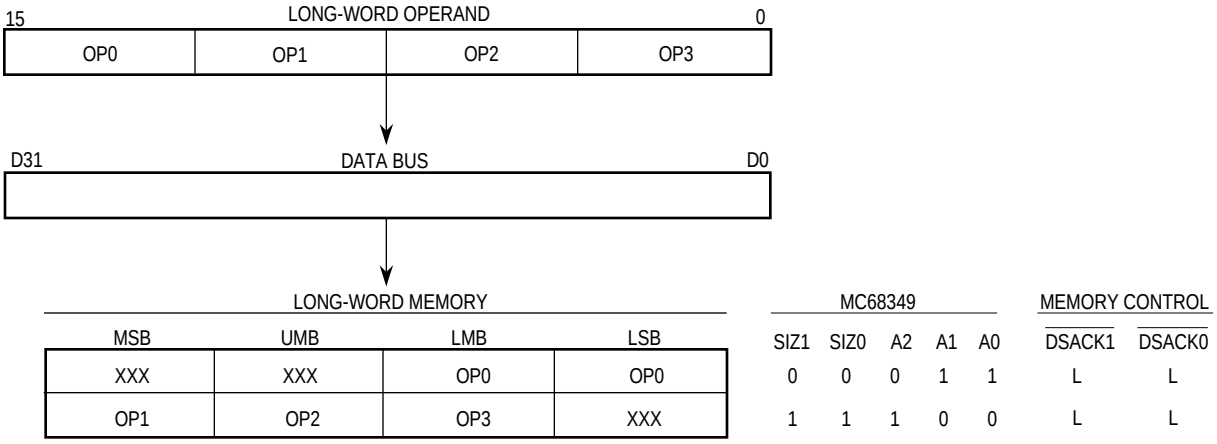


Figure 3-12. Misaligned Long-Word Operand Write to Long-Word Port Example

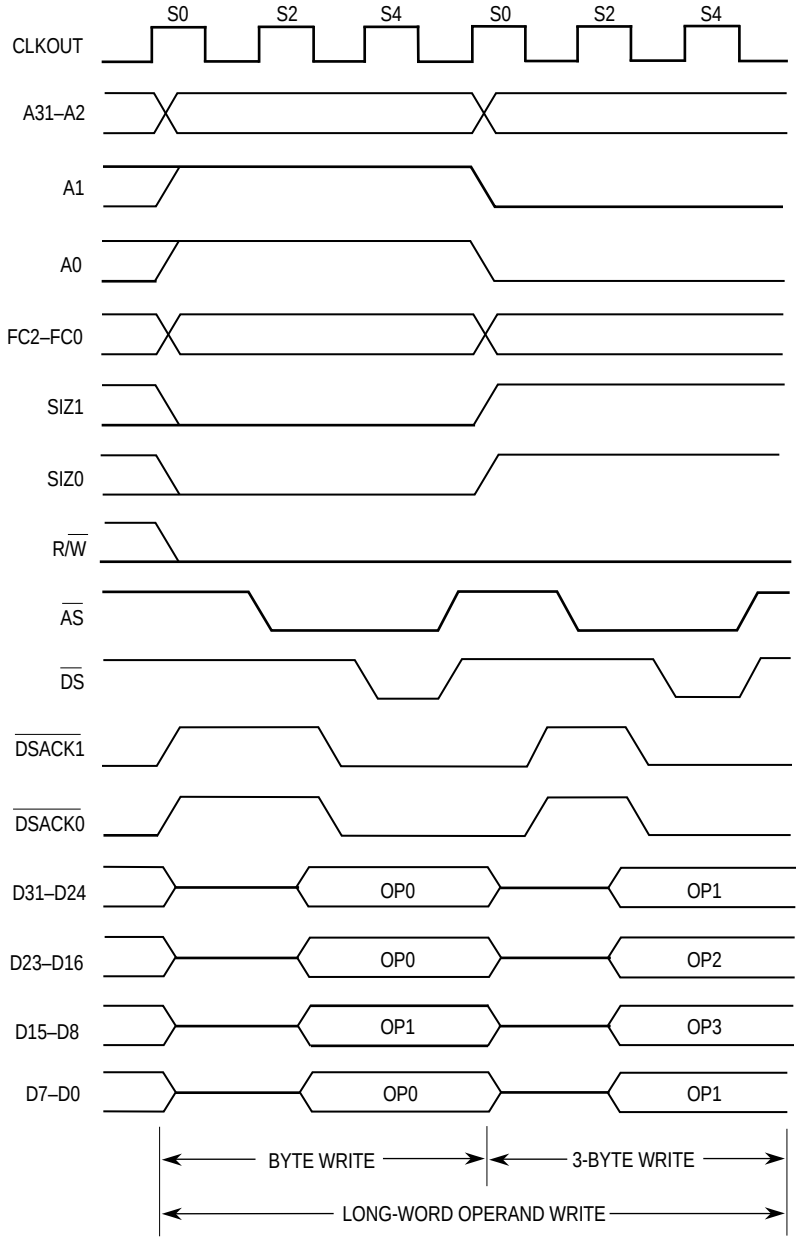


Figure 3-13. Misaligned Long-Word Operand Write to Long-Word Port Timing

3.2.3 Effects of Dynamic Bus Sizing and Operand Misalignment

The combination of operand size, operand alignment, and port size determines the number of bus cycles required to perform a particular memory access. Table 3-7 lists the number of bus cycles required for different operand sizes to different port sizes with all possible alignment conditions for write cycles and read cycles.

Table 3-7. Memory Alignment and Port Size Influence on Read/Write Bus Cycles

	Number of Bus Cycles (Data Port Size = 32 Bits:16 Bits:8 Bits)			
	A1, A0			
Operand Size	00	01	10	11
Long-Word Instruction*	1:2:4	N/A	N/A	N/A
Word Instruction	1:1:2	N/A	1:1:2	N/A
Byte Operand	1:1:1	1:1:1	1:1:1	1:1:1
Word Operand	1:1:2	1:2:2	1:1:2	2:2:2
Long-Word Operand	1:2:4	2:3:4	2:2:4	2:3:4

This table verifies that bus cycle throughput is significantly affected by port size and alignment. The MC68349 system designer and programmer should be aware of and account for these effects, particularly in time-critical applications.

The MC68349 uses a simple adaptive prefetch algorithm to minimize unused prefetches resulting from change-of-flow instructions. Instruction fetches are either aligned long-word or aligned word accesses, with the size of the access determined by the memory port size signaled for the previous fetch. When a byte or word port size termination is returned for an instruction fetch (regardless of whether that fetch is long-word or word), the next fetch is forced to be a word access. Likewise, if a long-word termination is returned, the next instruction fetch is a long-word access. Note that the prefetch size algorithm is not affected by the termination size for any intervening data accesses.

3.2.4 Bus Operation

The MC68349 bus is asynchronous, allowing external devices connected to the bus to operate at clock frequencies different from the clock for the MC68349. Bus operation uses the handshake lines ($\overline{AS}$, $\overline{DS}$, $\overline{DSACK1}$/$\overline{DSACK0}$, $\overline{BERR}$, and $\overline{HALT}$) to control data transfers. $\overline{AS}$ signals a valid address on the address bus, and $\overline{DS}$ is used as a condition for valid data on a write cycle. Decoding the SIZx outputs and lower address line A0 provides strobes that select the active portion of the data bus. The slave device (memory or peripheral) responds by placing the requested data on the correct portion of the data bus for a read cycle or by latching the data on a write cycle; the slave asserts the $\overline{DSACK1}$/$\overline{DSACK0}$ combination that corresponds to the port size to terminate the cycle. Alternatively, the SIM49 can be programmed to assert the $\overline{DSACK1}$/$\overline{DSACK0}$ combination

internally and respond for the slave. If no slave responds or the access is invalid, external control logic may assert $\overline{\text{BERR}}$ to abort the bus cycle or $\overline{\text{BERR}}$ with $\overline{\text{HALT}}$ to retry the bus cycle.

$\overline{\text{DSACKx}}$ can be asserted before the data from a slave device is valid on a read cycle. The length of time that $\overline{\text{DSACKx}}$ may precede data must not exceed a specified value in any asynchronous system to ensure that valid data is latched into the MC68349. (See **Section 11 Electrical Characteristics** for timing parameters.) Note that no maximum time is specified from the assertion of $\overline{\text{AS}}$ to the assertion of $\overline{\text{DSACKx}}$. Although the MC68349 can transfer data in a minimum of three clock cycles when the cycle is terminated with $\overline{\text{DSACKx}}$, the MC68349 inserts wait cycles in clock-period increments until $\overline{\text{DSACKx}}$ is recognized. $\overline{\text{BERR}}$ and/or $\overline{\text{HALT}}$ can be asserted after $\overline{\text{DSACKx}}$ is asserted. $\overline{\text{BERR}}$ and or $\overline{\text{HALT}}$ must be asserted within the time specified after $\overline{\text{DSACKx}}$ is asserted in any asynchronous system. If this maximum delay time is violated, the MC68349 may exhibit erratic behavior.

3.2.5 Synchronous Operation with $\overline{\text{DSACKx}}$

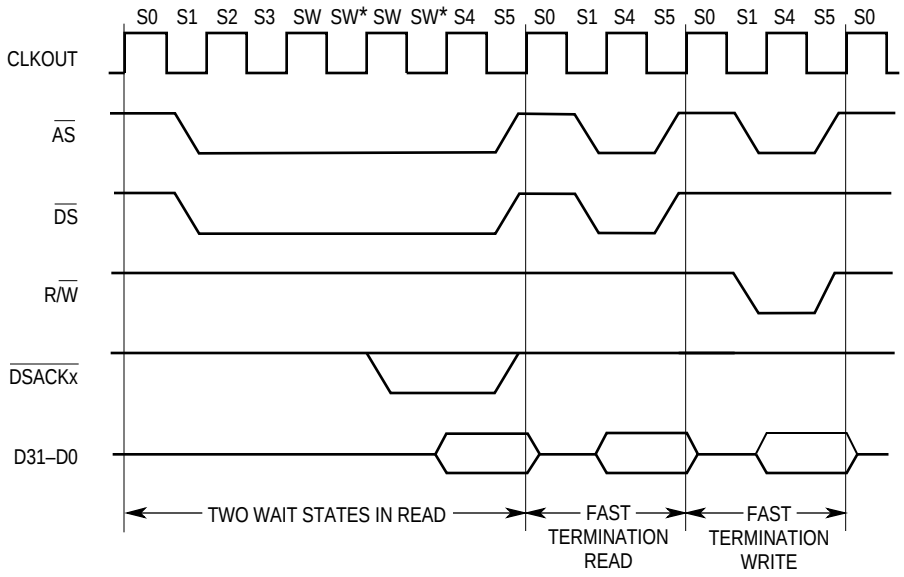
Although cycles terminated with $\overline{\text{DSACKx}}$ are classified as asynchronous, cycles terminated with $\overline{\text{DSACKx}}$ can also operate synchronously in that signals are interpreted relative to clock edges. The devices that use these cycles must synchronize the response to the MC68349 clock (CLKOUT) to be synchronous. Since the devices terminate bus cycles with $\overline{\text{DSACKx}}$, the dynamic bus sizing capabilities of the MC68349 are available. The minimum cycle time for these cycles is also three clocks. To support systems that use the system clock to generate $\overline{\text{DSACKx}}$ and other asynchronous inputs, the asynchronous input setup time and the asynchronous input hold time are given. If the setup and hold times are met for the assertion or negation of a signal such as $\overline{\text{DSACKx}}$, the MC68349 is guaranteed to recognize that signal level on that specific falling edge of the system clock. If the assertion of $\overline{\text{DSACKx}}$ is recognized on a particular falling edge of the clock, valid data is latched into the MC68349 (for a read cycle) on the next falling clock edge if the data meets the data setup time. In this case, the parameter for asynchronous operation can be ignored. The timing parameters are described in **Section 11 Electrical Characteristics**.

If a system asserts $\overline{\text{DSACKx}}$ for the required window around the falling edge of S2 and obeys the proper bus protocol by maintaining $\overline{\text{DSACKx}}$ (and/or $\overline{\text{BERR}}$/$\overline{\text{HALT}}$) until and throughout the clock edge that negates $\overline{\text{AS}}$ (with the appropriate asynchronous input hold time), no wait states are inserted. The bus cycle runs at its maximum speed for bus cycles terminated with $\overline{\text{DSACKx}}$ (three clocks per cycle). When $\overline{\text{BERR}}$ (or $\overline{\text{BERR}}$ and $\overline{\text{HALT}}$) is asserted after $\overline{\text{DSACKx}}$, $\overline{\text{BERR}}$ (and $\overline{\text{HALT}}$) must meet the appropriate setup time prior to the falling clock edge one clock cycle after $\overline{\text{DSACKx}}$ is recognized. This setup time is critical, and the MC68349 may exhibit erratic behavior if it is violated. When operating synchronously, the data-in setup and hold times for synchronous cycles may be used instead of the timing requirements for data relative to $\overline{\text{DS}}$.

3.2.6 Fast Termination Cycles

With an external device that has a fast access time, a fast termination cycle can provide a two-clock external bus transfer. Fast termination is enabled when the EDS bit in the base address register as well as the DD bits in the corresponding address mask register are set (EDS = 1, DDx = 11). Since the chip select circuits are driven from the system clock, the bus cycle termination is inherently synchronized with the system clock. Refer to **Section 4 System Integration Module** for more information on chip selects.

To use the fast termination option, an external device should be fast enough to have data ready, within the specified setup time, by the falling edge of S4. Figure 3-14 shows the $\overline{\text{DSACKx}}$ timing for a read with two wait states, followed by a fast termination read and write. When using the fast termination option, $\overline{\text{DS}}$ is asserted only in a read cycle, not in a write cycle.



* $\overline{\text{DSACKx}}$ only internally asserted for fast termination cycles.

Figure 3-14. Fast Termination Timing

3.3 DATA TRANSFER CYCLES

The transfer of data between the MC68349 and other devices involves the following signals:

- Address Bus A31–A0
- Data Bus D31–D0
- Control Signals

The address bus and data bus are parallel, nonmultiplexed buses. The bus master moves data on the bus by issuing control signals, and the bus uses a handshake protocol to ensure correct movement of the data. In all bus cycles, the bus master is responsible for de-skewing all signals it issues at both the start and end of the cycle. In addition, the bus master is responsible for de-skewing the acknowledge and data signals from the slave devices. The following paragraphs define read, write, and read-modify-write cycle operations. Each bus cycle is defined as a succession of states that apply to the bus operation. These states are different from the MC68349 states described for the CPU32+. The clock cycles used in the descriptions and timing diagrams of data transfer cycles are independent of the clock frequency. Bus operations are described in terms of external bus states.

3.3.1 Read Cycle

During a read cycle, the MC68349 receives data from a memory or peripheral device. If the instruction specifies a long-word, the MC68349 attempts to read four bytes at once. For a word operation, the MC68349 attempts to read two bytes at once. For a byte operation, the MC68349 reads one byte. The section of the data bus from which each byte is read depends on the operand size, address signals (A1, A0), and the port size. Refer to **3.2.1 Dynamic Bus Sizing** and **3.2.2 Misaligned Operands** for more information.

Figure 3-15 shows a long-word read cycle flowchart and Figure 3-16 illustrates a byte read cycle flowchart. Figures 3-17 and 3-18 show functional read cycles timing diagrams specified in terms of clock periods.

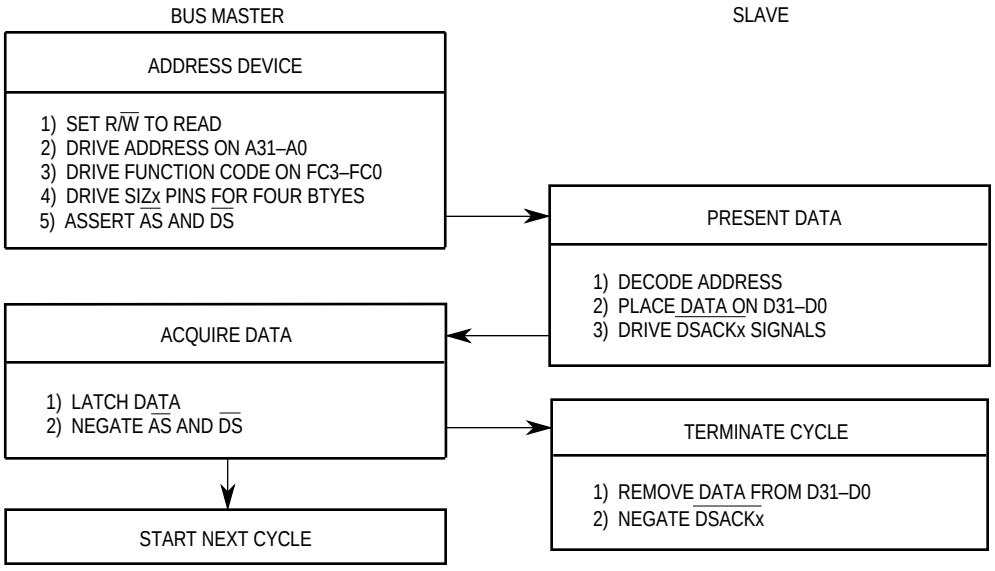


Figure 3-15. Long-Word Read Cycle Flowchart

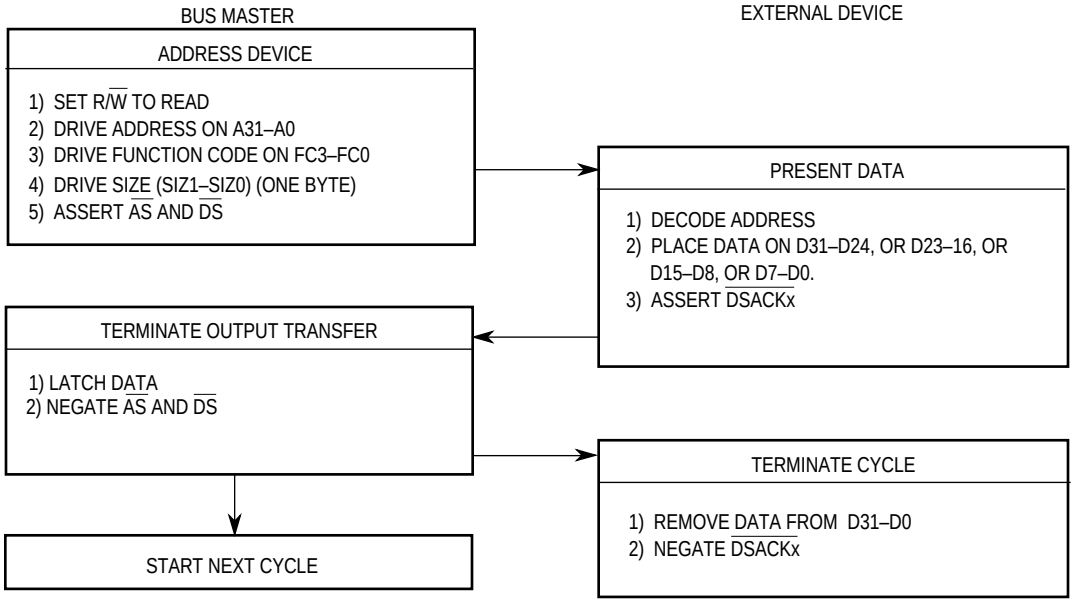


Figure 3-16. Byte Read Cycle Flowchart

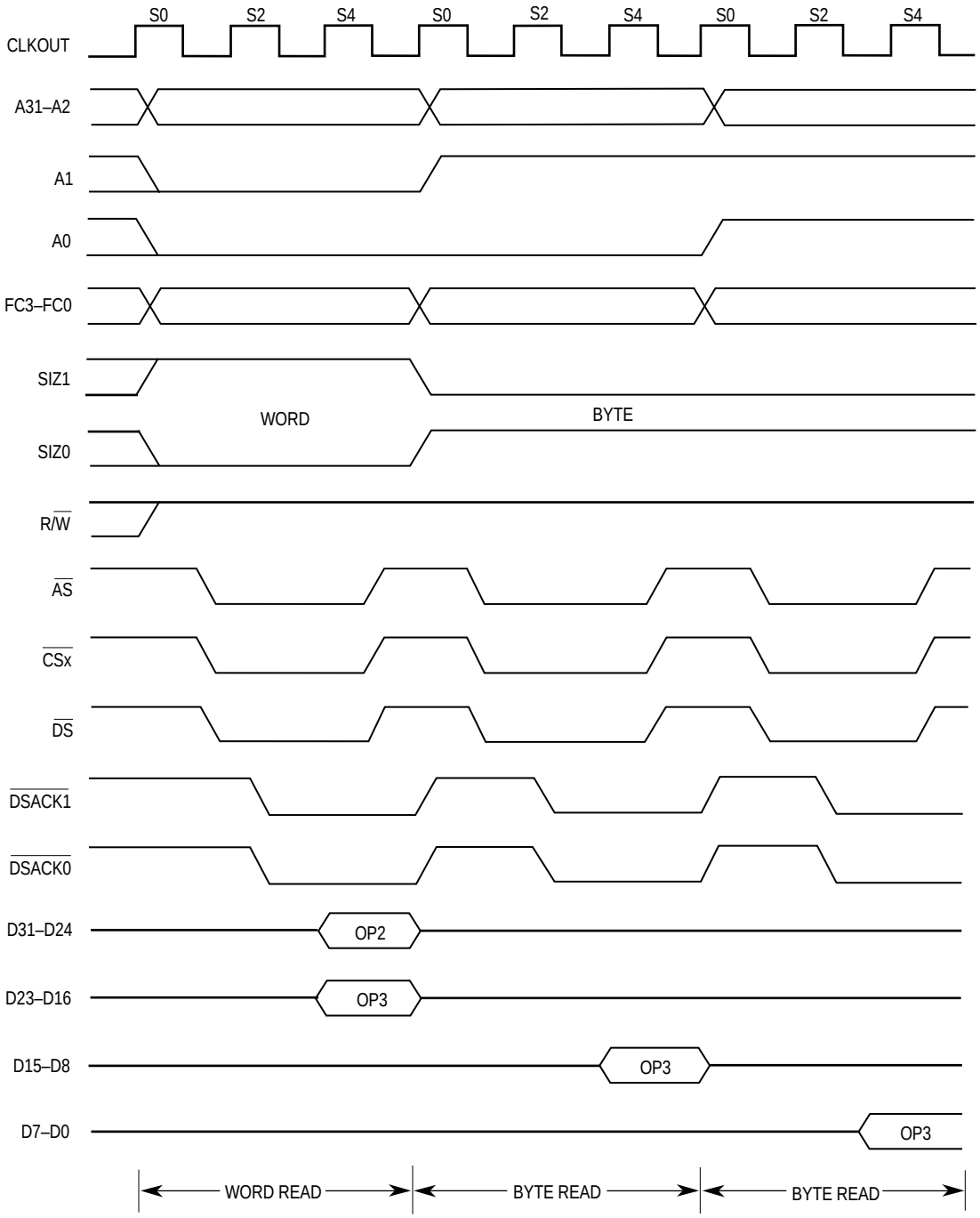


Figure 3-17. Byte and Word Read Cycles—32-Bit Port Timing

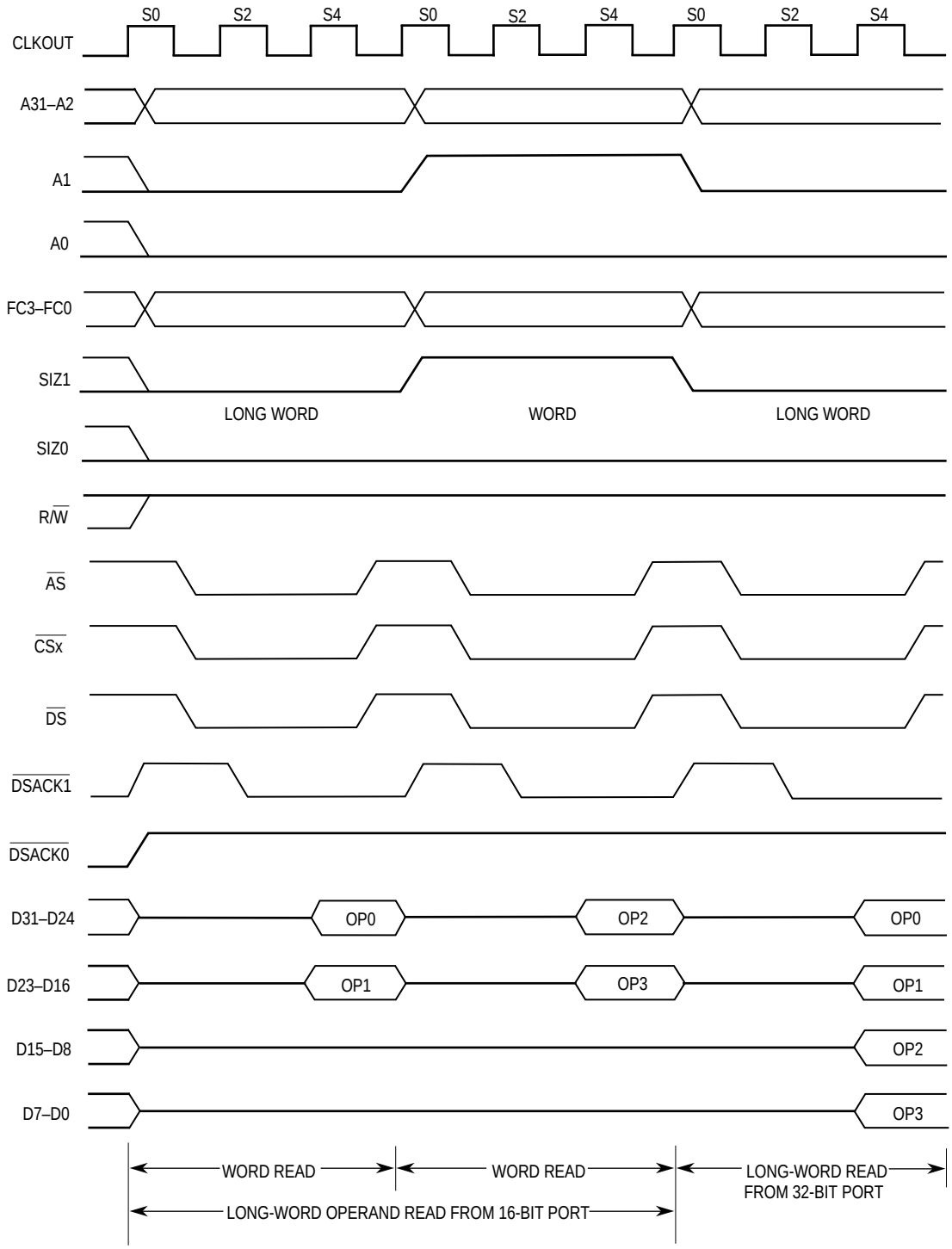


Figure 3-18. Long-Word Read—16-Bit and 32-Bit Port Timing

State 0—The read cycle starts in state 0 (S0). During S0, the MC68349 places a valid address on A31–A0 and valid function codes on FC3–FC0. The function codes select the address space for the cycle. The MC68349 drives R/$\overline{W}$ high for a read cycle. SIZ1/SIZ0 become valid, indicating the number of bytes requested for transfer.

State 1—One-half clock later, in state 1 (S1), the MC68349 asserts $\overline{AS}$ indicating a valid address on the address bus. The MC68349 also asserts $\overline{DS}$ during S1. The selected device uses R/$\overline{W}$, SIZ1 or SIZ0, A0, and $\overline{DS}$ to place its information on the data bus. One or both of the bytes (D15–D8 and D7–D0) are selected by SIZ1/SIZ0 and A0.

State 2—As long as at least one of the $\overline{DSACKx}$ signals is recognized on the falling edge of S2 (meeting the asynchronous input setup time requirement), data is latched on the falling edge of S4, and the cycle terminates.

State 3—If $\overline{DSACKx}$ is not recognized by the start of state 3 (S3), the MC68349 inserts wait states instead of proceeding to states 4 and 5. To ensure that wait states are inserted, both $\overline{DSACK1}$ and $\overline{DSACK0}$ must remain negated throughout the asynchronous input setup and hold times around the end of S2. If wait states are added, the MC68349 continues to sample $\overline{DSACKx}$ on the falling edges of the clock until one is recognized.

State 4—At the falling edge of state 4 (S4), the MC68349 latches the incoming data and samples $\overline{DSACKx}$ to get the port size.

State 5—The MC68349 negates $\overline{AS}$ and $\overline{DS}$ during state 5 (S5). It holds the address valid during S5 to provide address hold time for memory systems. R/$\overline{W}$, SIZ1 and SIZ0, and FC3–FC0 also remain valid throughout S5. The external device keeps its data and $\overline{DSACKx}$ signals asserted until it detects the negation of $\overline{AS}$ or $\overline{DS}$ (whichever it detects first). The device must remove its data and negate $\overline{DSACKx}$ within approximately one clock period after sensing the negation of $\overline{AS}$ or $\overline{DS}$. $\overline{DSACKx}$ signals that remain asserted beyond this limit may be prematurely detected for the next bus cycle.

3.3.2 Write Cycle

During a write cycle, the MC68349 transfers data to memory or a peripheral device. Figure 3-19 is a flowchart of a write cycle operation for a long-word transfer. Figure 3-20 shows the functional write cycle timing diagram specified in clock periods for two write cycles (between two read cycles with no idle time) for a 32-bit port.

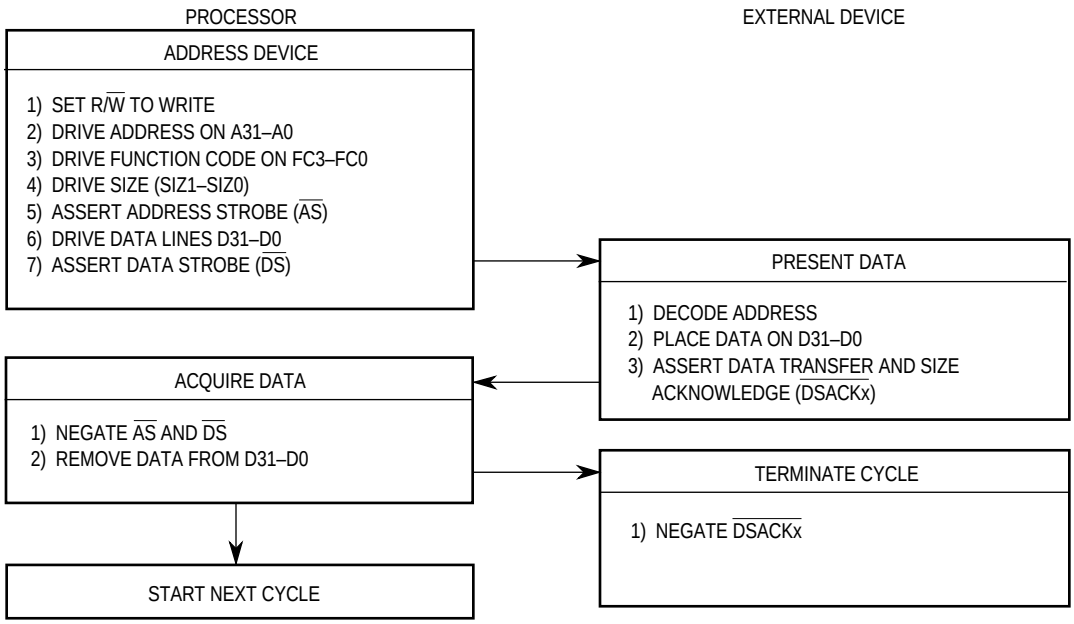


Figure 3-19. Long Word Write Cycle Flowchart

State 0—The write cycle starts in S0. During S0, the MC68349 places a valid address on A31–A0 and valid function codes on FC3–FC0. The function codes select the address space for the cycle. The MC68349 drives R/$\overline{W}$ low for a write cycle. SIZ1/SIZ0 become valid, indicating the number of bytes to be transferred.

State 1—One-half clock later during S1, the MC68349 asserts $\overline{AS}$, indicating a valid address on the address bus.

State 2—During S2, the MC68349 places the data to be written onto D15–D0, and samples $\overline{DSACKx}$ at the end of S2.

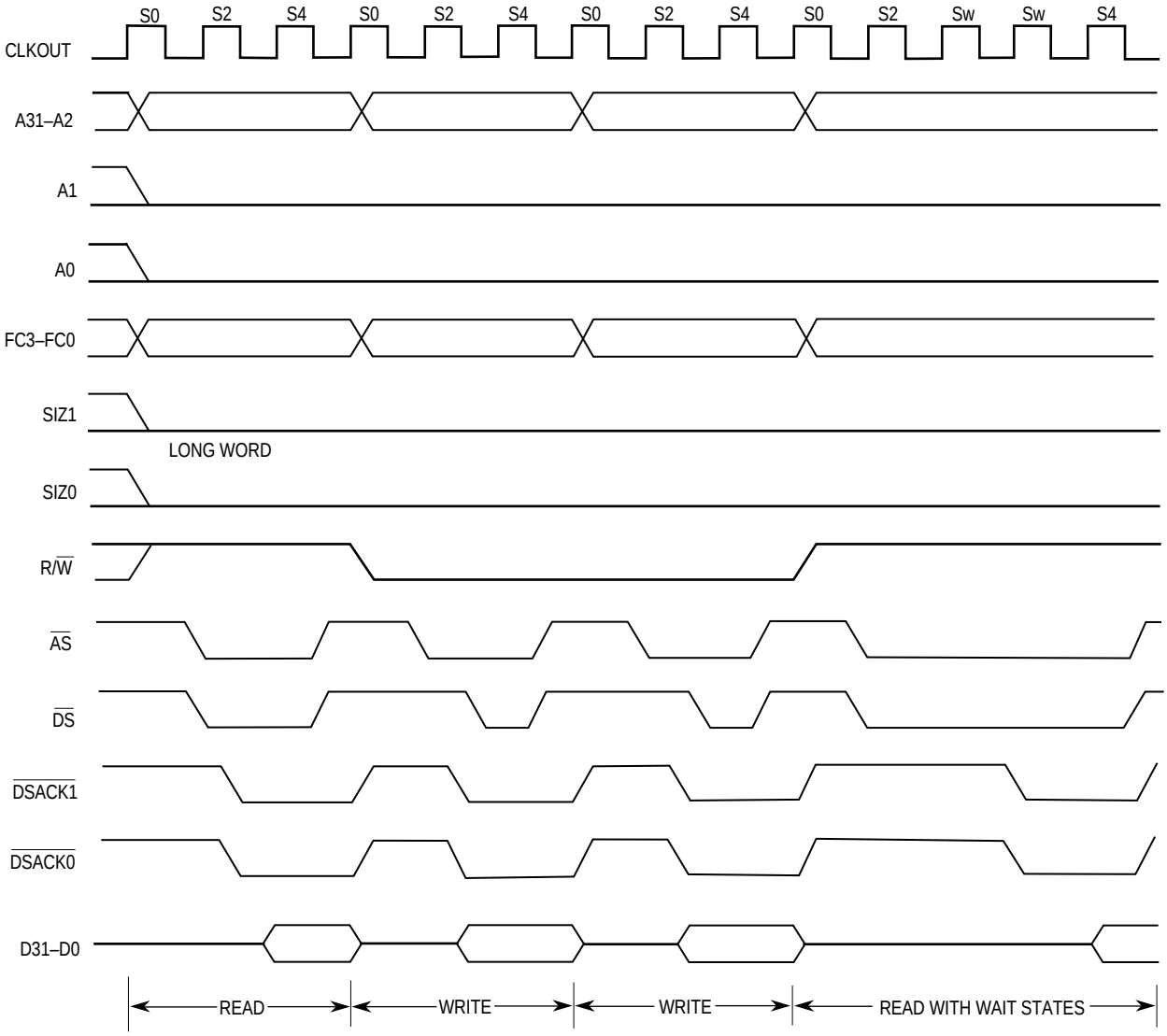


Figure 3-20. Read-Write-Read Cycles—32-Bit Port Timing

State 3—The MC68349 asserts DS during S3, indicating that data is stable on the data bus. As long as at least one of the DSACKx signals is recognized by the end of S2 (meeting the asynchronous input setup time requirement), the cycle terminates one clock later. If DSACKx is not recognized by the start of S3, the MC68349 inserts wait states instead of proceeding to S4 and S5. To ensure that wait states are inserted, both DSACK1 and DSACK0 must remain negated throughout the asynchronous input setup and hold times around the end of S2. If wait states are added, the MC68349 continues to sample DSACKx on the falling edges of the clock until one is recognized. The selected device uses R/W, SIZ1/SIZ0, and A0 to latch data from the appropriate byte(s) of D15–D8 and D7–D0. SIZ1/SIZ0 and A0 select the bytes of the data bus. If it has not already done so, the device asserts DSACKx to signal that it has successfully stored the data.

State 4—The MC68349 issues no new control signals during S4.

State 5—The MC68349 negates $\overline{AS}$ and $\overline{DS}$ during S5. It holds the address and data valid during S5 to provide address hold time for memory systems. R/$\overline{W}$, SIZ1/SIZ0, and FC3–FC0 also remain valid throughout S5. The external device must keep $\overline{DSACKx}$ asserted until it detects the negation of $\overline{AS}$ or $\overline{DS}$ (whichever it detects first). The device must negate $\overline{DSACKx}$ within approximately one clock period after sensing the negation of $\overline{AS}$ or $\overline{DS}$. $\overline{DSACKx}$ signals that remain asserted beyond this limit may be prematurely detected for the next bus cycle.

3.3.3 Read-Modify-Write Cycle

The read-modify-write cycle performs a read, conditionally modifies the data in the arithmetic logic unit, and may write the data out to memory. In the MC68349, this operation is indivisible, providing semaphore capabilities for multiprocessor systems. During the entire read-modify-write sequence, the MC68349 asserts $\overline{RMC}$ to indicate that an indivisible operation is occurring. The MC68349 does not issue a $\overline{BG}$ signal in response to a $\overline{BR}$ signal during this operation. Figure 3-21 is an example of a functional timing diagram of a read-modify-write instruction specified in terms of clock periods.

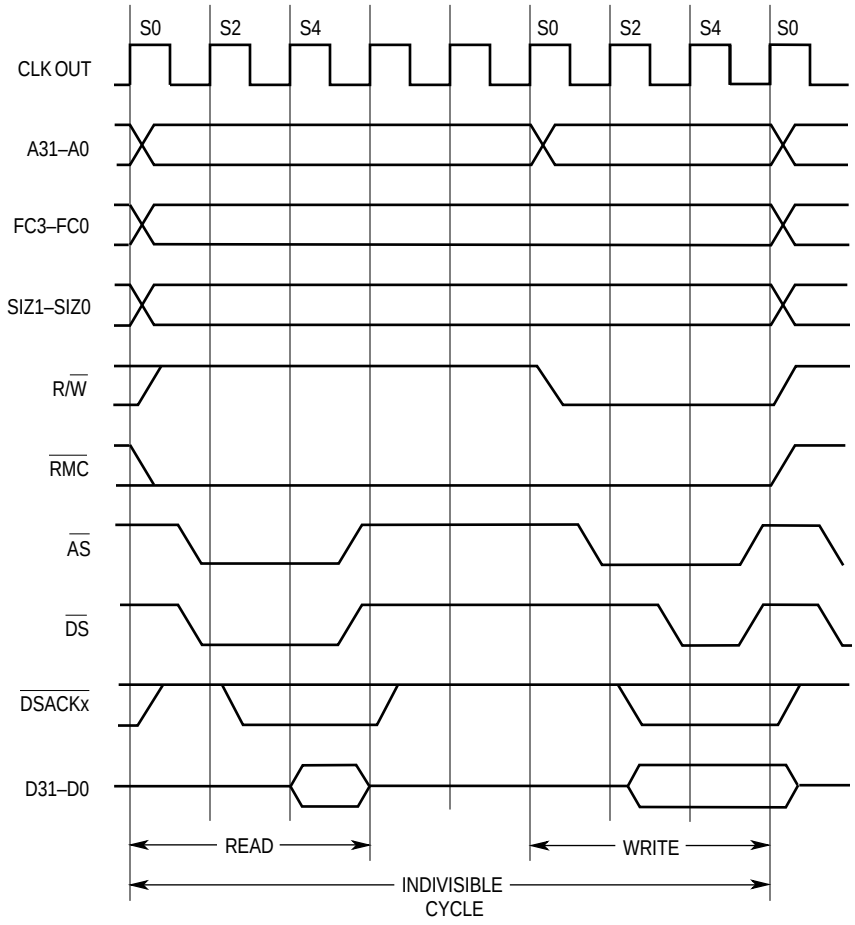


Figure 3-21. Read-Modify-Write Cycle Timing

State 0—The MC68349 asserts $\overline{RMC}$ in S0 to identify a read-modify-write cycle. The MC68349 places a valid address on A31–A0 and valid function codes on FC3–FC0. The function codes select the address space for the operation. SIZ1/SIZ0 become valid in S0 to indicate the operand size. The MC68349 drives R/$\overline{W}$ high for the read cycle.

State 1—One-half clock later during S1, the MC68349 asserts $\overline{AS}$ indicating a valid address on the address bus. The MC68349 also asserts $\overline{DS}$ during S1.

State 2—The selected device uses R/$\overline{W}$, SIZ1/SIZ0, A0, and $\overline{DS}$ to place information on the data bus. Any of the bytes (D31–D24, D23–D16, D15–D8, and D7–D0) are selected by SIZ1, SIZ0, A1, and A0. Concurrently, the selected device may assert $\overline{DSACKx}$.

State 3—As long as at least one of the $\overline{DSACKx}$ signals is recognized by the end of S2 (meeting the asynchronous input setup time requirement), data is latched on the next falling edge of the clock, and the cycle terminates. If $\overline{DSACKx}$ is not recognized by the start of S3, the MC68349 inserts wait states instead of proceeding to S4 and S5. To ensure that wait states are inserted, both $\overline{DSACK1}$ and $\overline{DSACK0}$ must remain negated throughout the asynchronous input setup and hold times around the end of S2. If wait states are added, the MC68349 continues to sample the $\overline{DSACKx}$ signals on the falling edges of the clock until one is recognized.

State 4—At the end of S4, the MC68349 latches the incoming data.

State 5—The MC68349 negates $\overline{AS}$ and $\overline{DS}$ during S5. If more than one read cycle is required to read in the operand(s), S0–S5 are repeated for each read cycle. When finished reading, the MC68349 holds the address, R/$\overline{W}$, and FC3–FC0 valid in preparation for the write portion of the cycle. The external device keeps its data and $\overline{DSACKx}$ signals asserted until it detects the negation of $\overline{AS}$ or $\overline{DS}$ (whichever it detects first). The device must remove the data and negate $\overline{DSACKx}$ within approximately one clock period after sensing the negation of $\overline{AS}$ or $\overline{DS}$. $\overline{DSACKx}$ signals that remain asserted beyond this limit may be prematurely detected for the next portion of the operation.

Idle States—The MC68349 does not assert any new control signals during the idle states, but it may internally begin the modify portion of the cycle at this time. S0–S5 are omitted if no write cycle is required. If a write cycle is required, R/$\overline{W}$ remains in the read mode until S0 to prevent bus conflicts with the preceding read portion of the cycle; the data bus is not driven until S2.

State 0—The MC68349 drives R/$\overline{W}$ low for a write cycle. Depending on the write operation to be performed, the address lines may change during S0.

State 1—In S1, the MC68349 asserts $\overline{AS}$, indicating a valid address on the address bus.

State 2—During S2, the MC68349 places the data to be written onto D31–D0.

State 3—The MC68349 asserts $\overline{DS}$ during S3, indicating stable data on the data bus. As long as at least one of the $\overline{DSACKx}$ signals is recognized by the end of S2 (meeting the asynchronous input setup time requirement), the cycle terminates one clock later. If $\overline{DSACKx}$ is not recognized by the start of S3, the MC68349 inserts wait states instead of

proceeding to S4 and S5. To ensure that wait states are inserted, both $\overline{\text{DSACK1}}$ and $\overline{\text{DSACK0}}$ must remain negated throughout the asynchronous input setup and hold times around the end of S2. If wait states are added, the MC68349 continues to sample $\overline{\text{DSACKx}}$ on the falling edges of the clock until one is recognized. The selected device uses R/$\overline{\text{W}}$, $\overline{\text{DS}}$, SIZ1/SIZ0, A1, and A0 to latch data from the appropriate section(s) of the data bus (D31–D24, D23–D16, D15–D8, and D7–D0). SIZ1/SIZ0, A1, and A0 select the data bus sections. If it has not already done so, the device asserts $\overline{\text{DSACKx}}$ when it has successfully stored the data.

State 4—The MC68349 issues no new control signals during S4.

State 5—The MC68349 negates $\overline{\text{AS}}$ and $\overline{\text{DS}}$ during S5. It holds the address and data valid during S5 to provide address hold time for memory systems. R/$\overline{\text{W}}$ and FC3–FC0 also remain valid throughout S5. If more than one write cycle is required, states S0–S5 are repeated for each write cycle. The external device keeps $\overline{\text{DSACKx}}$ asserted until it detects the negation of $\overline{\text{AS}}$ or $\overline{\text{DS}}$ (whichever it detects first). The device must remove its data and negate $\overline{\text{DSACKx}}$ within approximately one clock period after sensing the negation of $\overline{\text{AS}}$ or $\overline{\text{DS}}$.

3.4 CPU SPACE CYCLES

FC3–FC0 select user and supervisor program and data areas. The area selected by FC3–FC0 = $7 is classified as the CPU space. The breakpoint acknowledge, LPSTOP broadcast, module base address register access, and interrupt acknowledge cycles described in the following paragraphs use CPU space. The CPU space type, which is encoded on A19–A16 during a CPU space operation, indicates the function that the MC68349 is performing. On the MC68349, four of the encodings are implemented as shown in Figure 3-22. All unused values are reserved by Motorola for additional CPU space types.

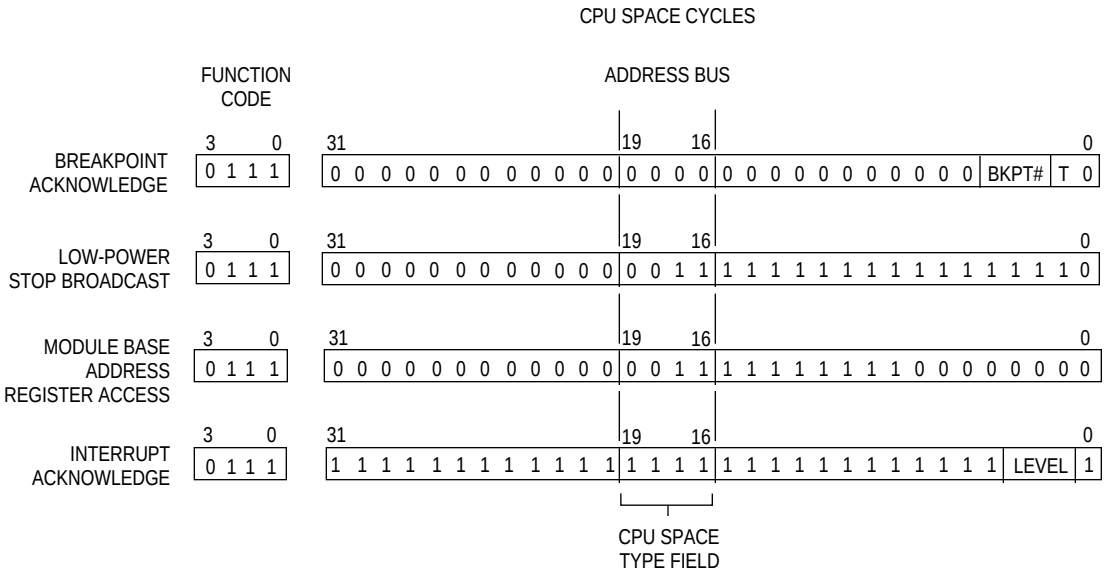


Figure 3-22. CPU Space Address Encoding

3.4.1 Breakpoint Acknowledge Cycle

The breakpoint acknowledge cycle allows external hardware to insert an instruction directly into the instruction pipeline as the program executes. The breakpoint acknowledge cycle is generated by the execution of a breakpoint instruction (BKPT) or the assertion of the $\overline{\text{BKPT}}$ pin. The T-bit state (shown in Figure 3-10) differentiates a software breakpoint cycle (T = 0) from a hardware breakpoint cycle (T = 1).

When a BKPT instruction is executed (software breakpoint), the MC68349 performs a word read from CPU space, type 0, at an address corresponding to the breakpoint number (bits [2–0] of the BKPT opcode) on A4–A2, and the T-bit (A1) is cleared. If this bus cycle is terminated with $\overline{\text{BERR}}$ (i.e., no instruction word is available), the MC68349 then performs illegal instruction exception processing. If the bus cycle is terminated by $\overline{\text{DSACKx}}$, the MC68349 uses the data on D31–D16 (for 32-bit or 16-bit ports) or two reads from D31–D24 (for 8-bit ports) to replace the BKPT instruction in the internal instruction pipeline and then begins execution of that instruction.

When the CPU32+ acknowledges a $\overline{\text{BKPT}}$ pin assertion (hardware breakpoint) with background mode disabled, the CPU32+ performs a word read from CPU space, type 0, at an address corresponding to all ones on A4–A2 (BKPT#7), and the T-bit (A1) is set. If this bus cycle is terminated by $\overline{\text{BERR}}$, the MC68349 performs hardware breakpoint exception processing. If this bus cycle is terminated by $\overline{\text{DSACKx}}$, the MC68349 ignores data on the data bus and continues execution of the next instruction.

NOTE

The $\overline{BKPT}$ pin is sampled on the same clock phase as data and is latched with data as it enters the CPU32+ pipeline. If $\overline{BKPT}$ is asserted for only one bus cycle and a pipeline flush occurs before $\overline{BKPT}$ is detected by the CPU32+, $\overline{BKPT}$ is ignored. To ensure detection of $\overline{BKPT}$ by the CPU32+, $\overline{BKPT}$ can be asserted until a breakpoint acknowledge cycle is recognized.

When the MC68349 is configured for a 32-bit bus, the CPU32+ can fetch two instructions simultaneously. Since there is only one $\overline{BKPT}$ pin, the external user cannot break individually on those instructions, but rather must break on both, causing the BKPT exception to be taken after the first instruction and before the second instruction.

The breakpoint operation flowchart is shown in Figure 3-23. Figures 3-24 and 3-25 show the timing diagrams for the breakpoint acknowledge cycle with instruction opcodes supplied on the cycle and with an exception signaled, respectively.

3.4.2 LPSTOP Broadcast Cycle

The low power stop (LPSTOP) broadcast cycle is generated by the CPU32+ executing the LPSTOP instruction. Since the external bus interface must get a copy of the interrupt mask level from the CPU32+, the CPU32+ performs a CPU space type 3 write with the mask level encoded on the data bus, as shown in the following figure. The CPU space type 3 cycle waits for the bus to be available, and is shown externally to indicate to external devices that the MC68349 is going into LPSTOP mode. If an external device requires additional time to prepare for entry into LPSTOP mode, entry can be delayed by asserting $\overline{HALT}$. The SIM49 provides internal $\overline{DSACKx}$ response to this cycle. For more information on how the SIM49 responds to LPSTOP mode, see **Section 4 System Integration Module**.

31 … 19	18	17	16
—	I2	I1	I0

15 … 3	2	1	0
—	I2	I1	I0

I2–I0—Interrupt Mask Level

The interrupt mask level is encoded on bits 2–0 of the data bus during an LPSTOP broadcast. For convenience when decoding, the interrupt mask level is replicated on data bus bits 18–16 during an LPSTOP broadcast.

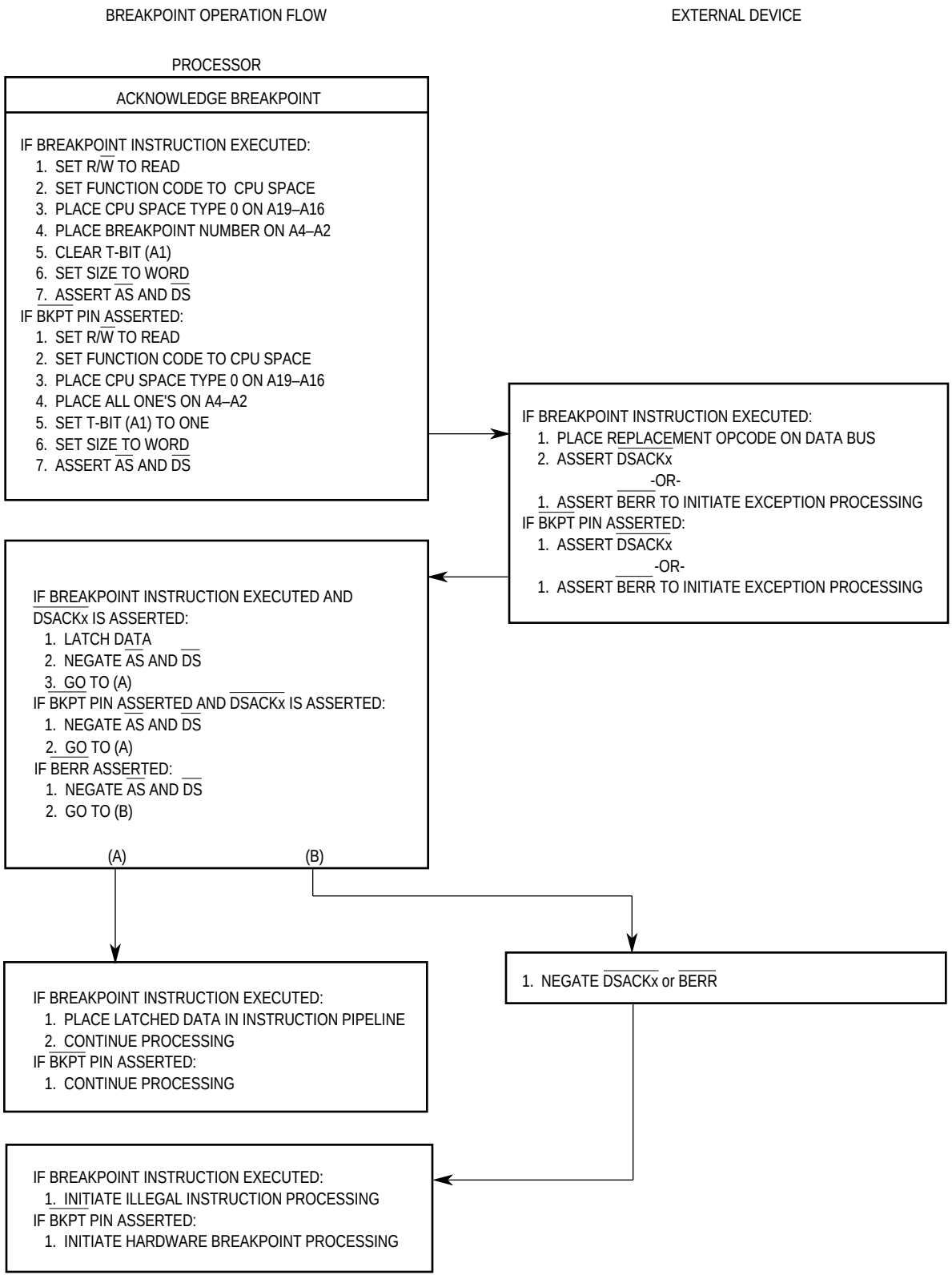


Figure 3-23. Breakpoint Operation Flowchart

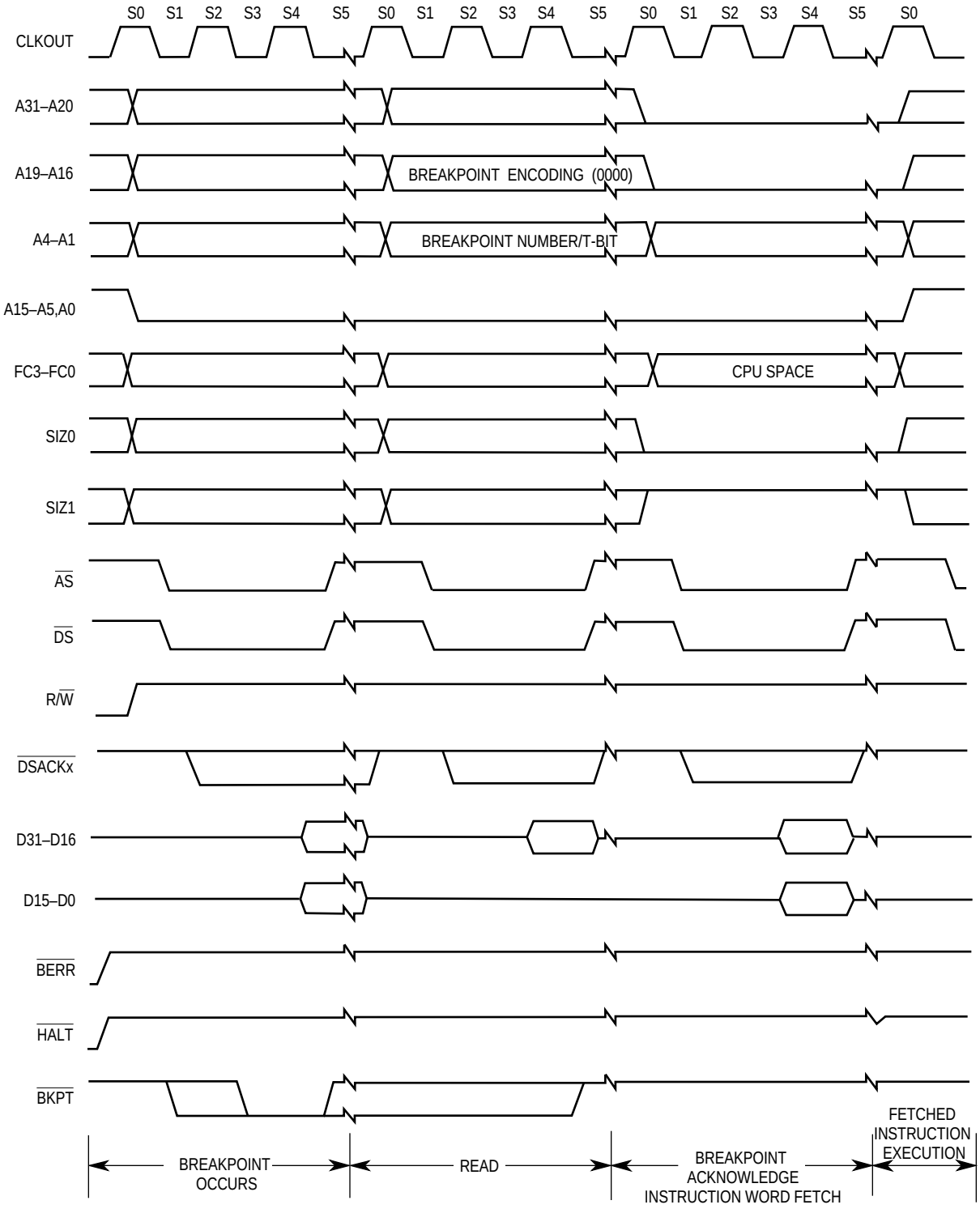


Figure 3-24. Breakpoint Acknowledge Cycle Timing (Opcode Returned)

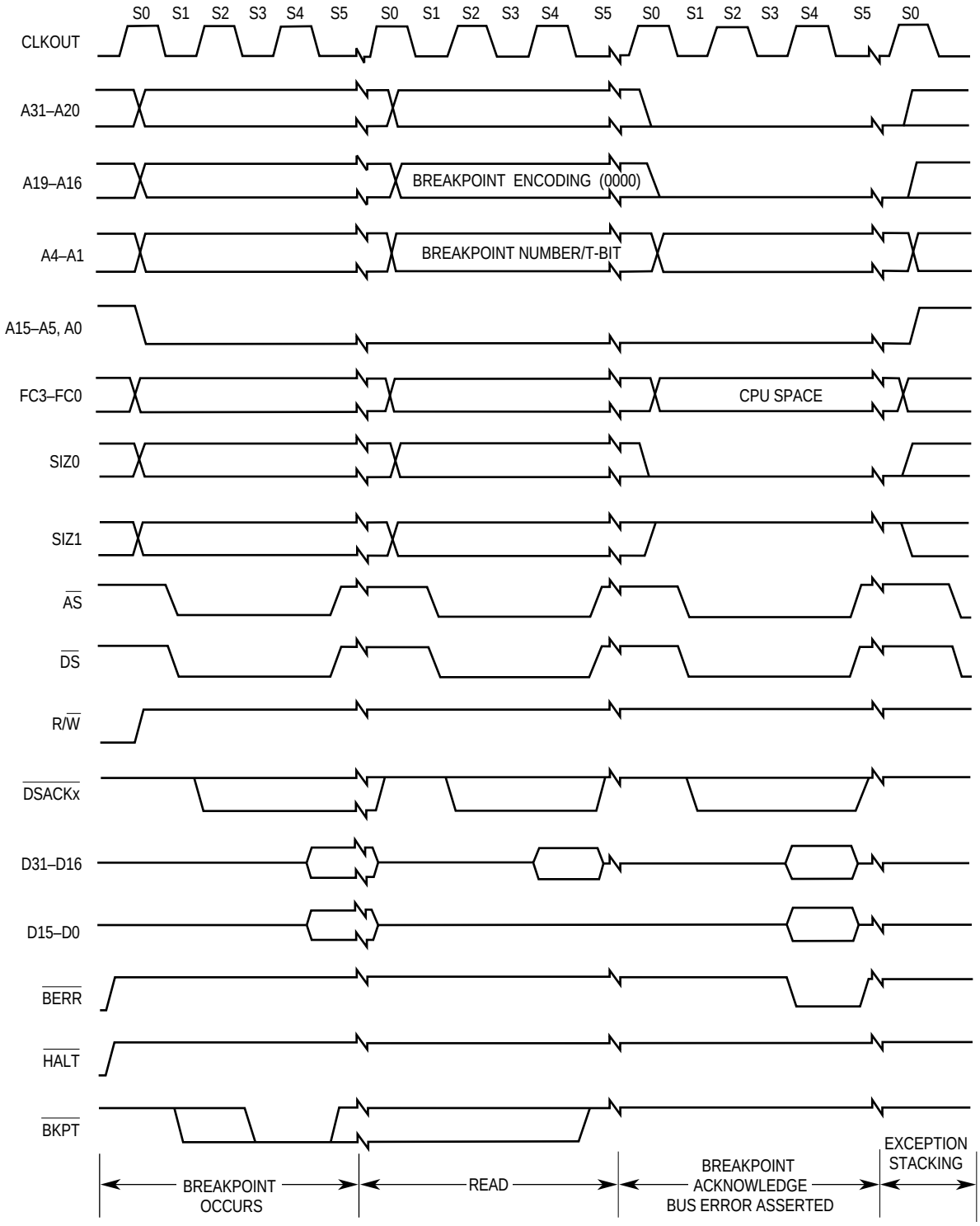


Figure 3-25. Breakpoint Acknowledge Cycle Timing (Exception Signaled)

3.4.3 Module Base Address Register Access

All internal module registers, including the SIM49, occupy a single 4-Kbyte block that is relocatable along 4-Kbyte boundaries. The location is fixed by writing the desired base address of the SIM49 block to the module base address register using the MOVES instruction. The module base address register is only accessible in CPU space at address $0003FF00. The source function code (SFC) or destination function code (DFC) register must indicate CPU space (FC3–FC0 = $7), using the MOVEC instruction, before accessing the module base address register. Refer to **Section 4 System Integration Module** for additional information on the module base address register.

3.4.4 Interrupt Acknowledge Bus Cycles

The CPU32+ makes an interrupt pending in three cases. The first case occurs when a peripheral device signals the CPU32+ (with $\overline{IRQ7}$–$\overline{IRQ1}$) that the device requires service and the internally synchronized value on these signals indicates a higher priority than the interrupt mask in the status register. The second case occurs when a transition has occurred in the case of a level 7 interrupt. A recognized level 7 interrupt must be removed for one clock cycle before a second level 7 can be recognized. The third case occurs if, upon returning from servicing a level 7 interrupt, the request level stays at 7 and the processor mask level changes from 7 to a lower level, a second level 7 is recognized. The CPU32+ takes an interrupt exception for a pending interrupt within one instruction boundary (after processing any other pending exception with a higher priority). The following paragraphs describe the types of interrupt acknowledge bus cycles that can be executed as part of interrupt exception processing.

3.4.4.1 INTERRUPT ACKNOWLEDGE CYCLE—TERMINATED NORMALLY. When the CPU32+ processes an interrupt exception, it performs an interrupt acknowledge cycle to obtain the number of the vector that contains the starting location of the interrupt service routine. Some interrupting devices have programmable vector registers that contain the interrupt vectors for the routines they use. The following paragraphs describe the interrupt acknowledge cycle for these devices. Other interrupting conditions or devices that cannot supply a vector number will use the autovector cycle described in **3.4.4.2 Autovector Interrupt Acknowledge Cycle**.

The interrupt acknowledge cycle is a read cycle. It differs from the read cycle described in **3.3.1 Read Cycle** in that it accesses the CPU address space. Specifically, the differences are as follows:

1. FC3–FC0 are set to $7 (FC3/FC2/FC1/FC0 = 0111) for CPU address space.
2. A3, A2, and A1 are set to the interrupt request level, and the $\overline{\text{IACKx}}$ strobe corresponding to the current interrupt level is asserted. (Either the function codes and address signals or the $\overline{\text{IACKx}}$ strobes can be monitored to determine that an interrupt acknowledge cycle is in progress and the current interrupt level.)
3. The CPU32+ space type field (A19–A16) is set to $F (interrupt acknowledge).
4. Other address signals (A31–A20, A15–A4, and A0) are set to one.
5. The SIZ0/SIZ1 and R/$\overline{\text{W}}$ signals are driven to indicate a single-byte read cycle. The responding device places the vector number on the least significant byte of its data port (for an 8-bit port, the vector number must be on D31–D24; for a 16-bit port, the vector must be on D23–D16; for a 32-bit port, the vector number must be on D7–D0) during the interrupt acknowledge cycle. The cycle is then terminated normally with $\overline{\text{DSACKx}}$.

Figure 3-26 is a flowchart of the interrupt acknowledge cycle; Figure 3-27 shows the timing for an interrupt acknowledge cycle terminated with $\overline{\text{DSACKx}}$.

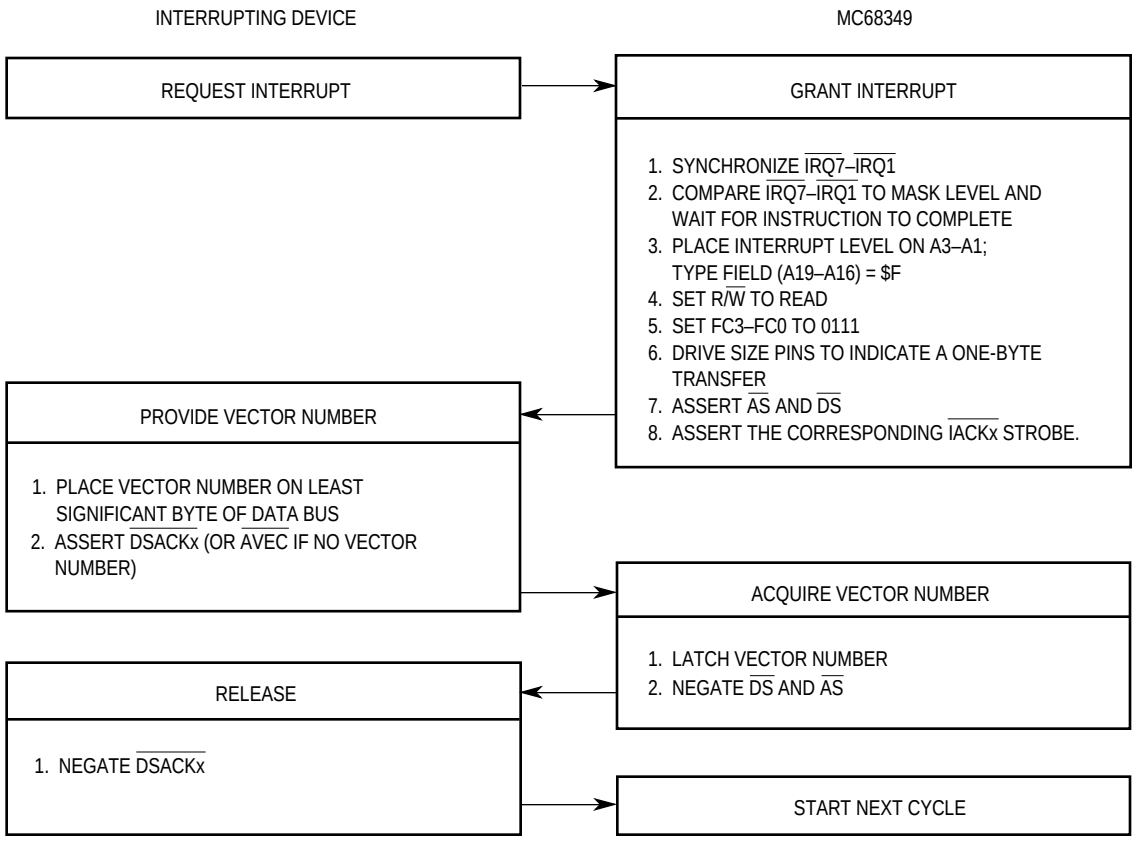
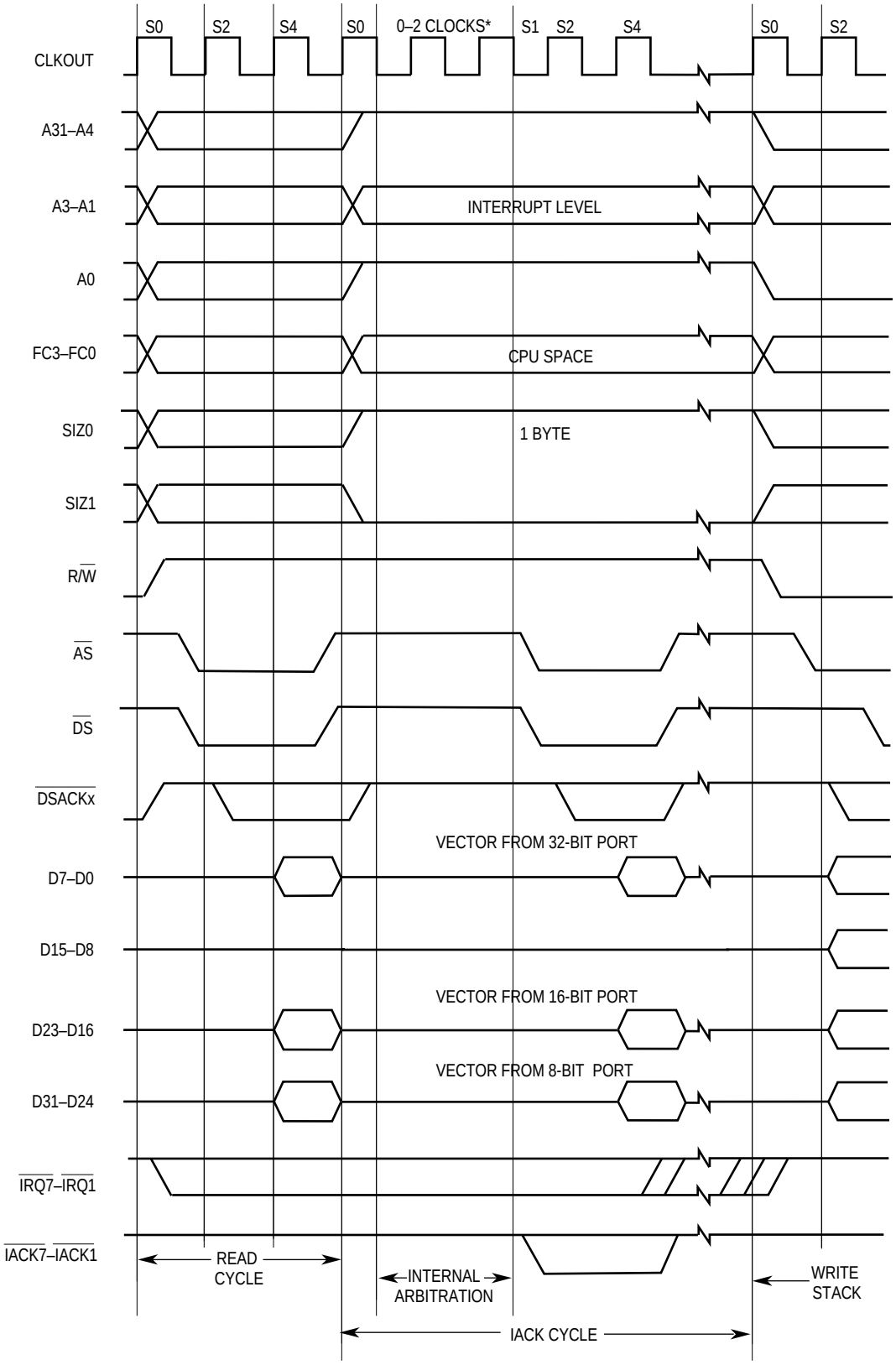


Figure 3-26. Interrupt Acknowledge Cycle Flowchart



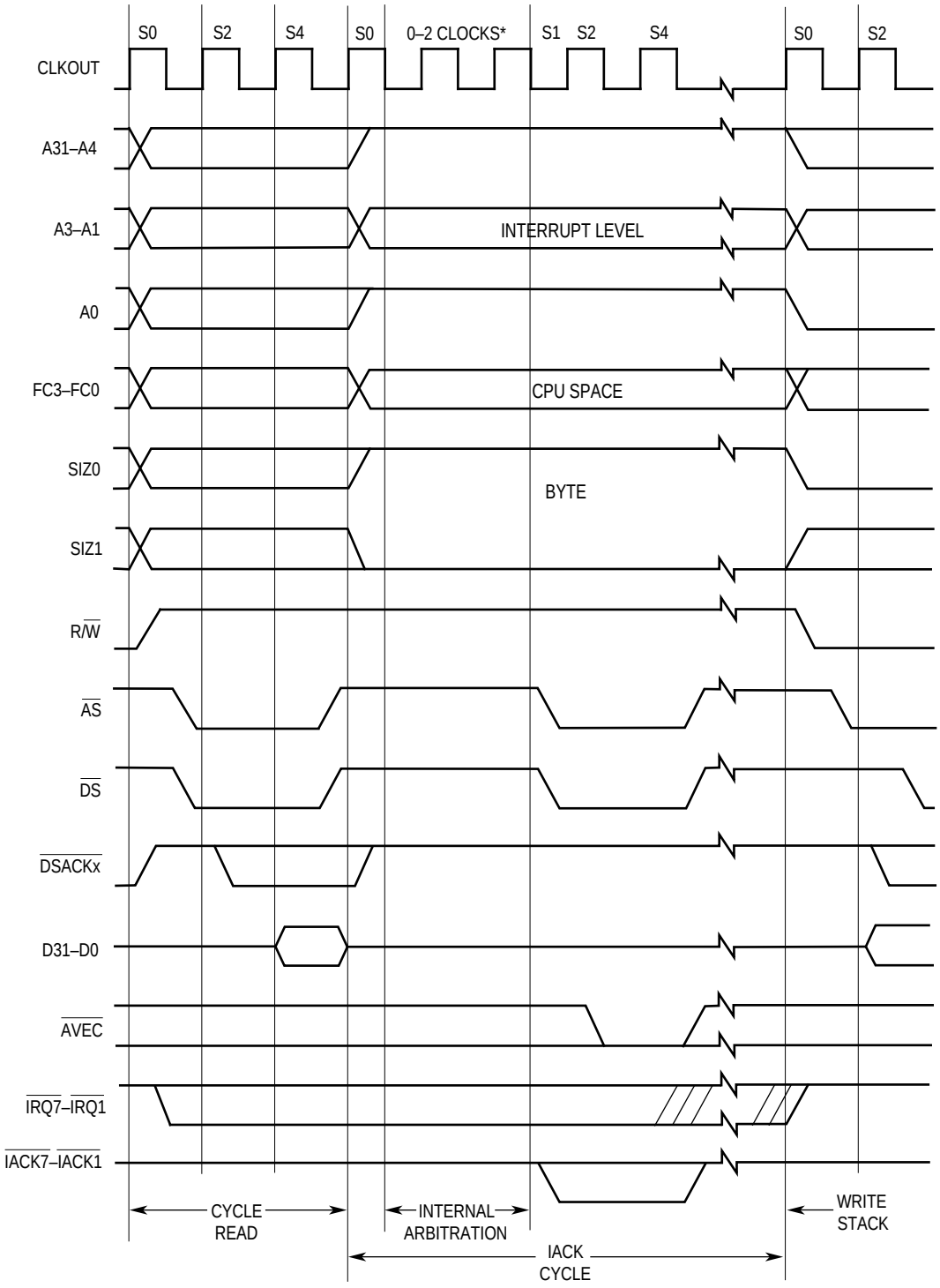
*Internal Arbitration may take between 0–2 clock cycles.

Figure 3-27. Interrupt Acknowledge Cycle Timing

3.4.4.2 AUTOVECTOR INTERRUPT ACKNOWLEDGE CYCLE. When the interrupting device cannot supply a vector number, it requests an automatically generated vector (autovector). Instead of placing a vector number on the data bus and asserting $\overline{\text{DSACKx}}$, the device asserts $\overline{\text{AVEC}}$ to terminate the cycle. If the $\overline{\text{DSACKx}}$ signals are asserted during an interrupt acknowledge cycle terminated by $\overline{\text{AVEC}}$, the $\overline{\text{DSACKx}}$ signals and data will be ignored if $\overline{\text{AVEC}}$ is asserted before or at the same time as the $\overline{\text{DSACKx}}$ signals. The vector number supplied in an autovector operation is derived from the interrupt level of the current interrupt. When $\overline{\text{AVEC}}$ is asserted instead of $\overline{\text{DSACKx}}$ during an interrupt acknowledge cycle, the MC68349 ignores the state of the data bus and internally generates the vector number (the sum of the interrupt level plus 24 ($18)).

$\overline{\text{AVEC}}$ is multiplexed with $\overline{\text{CS0}}$. The FIRQ bit in the SIM49 module configuration register controls whether the $\overline{\text{AVEC}}/\overline{\text{CS0}}$ pin is used as an autovector input or as $\overline{\text{CS0}}$ (refer to **Section 4 System Integration Module** for additional information). $\overline{\text{AVEC}}$ is only sampled during an interrupt acknowledge cycle. During all other cycles, $\overline{\text{AVEC}}$ is ignored. Additionally, $\overline{\text{AVEC}}$ can be internally generated for external devices by programming the autovector register. Seven distinct autovectors can be used, corresponding to the seven levels of interrupt available with signals $\overline{\text{IRQ7}}$–$\overline{\text{IRQ1}}$. Figure 3-28 shows the timing for an autovector operation.

3.4.4.3 SPURIOUS INTERRUPT CYCLE. Requested interrupts, whether internal or external, are arbitrated internally. When no internal module (including the SIM49, which responds for external requests) responds during an interrupt acknowledge cycle by arbitrating for the interrupt acknowledge cycle internally, the spurious interrupt monitor generates an internal bus error signal to terminate the vector acquisition. The MC68349 automatically generates the spurious interrupt vector number (24) instead of the interrupt vector number in this case. When an external device does not respond to an interrupt acknowledge cycle with $\overline{\text{AVEC}}$ or $\overline{\text{DSACKx}}$, a bus monitor must assert $\overline{\text{BERR}}$, which results in the CPU32+ taking the spurious interrupt vector. If $\overline{\text{HALT}}$ is also asserted, the MC68349 retries the interrupt acknowledge cycle instead of using the spurious interrupt vector.



* Internal Arbitration may take between 0–2 clocks.

Figure 3-28. Autovector Operation Timing

3.5 BUS EXCEPTION CONTROL CYCLES

The bus architecture requires assertion of $\overline{\text{DSACKx}}$ from an external device to signal that a bus cycle is complete. Neither $\overline{\text{DSACKx}}$ nor $\overline{\text{AVEC}}$ is asserted in the following cases:

- $\overline{\text{DSACKx}}$/$\overline{\text{AVEC}}$ is programmed to respond internally.
- The external device does not respond.
- Various other application-dependent errors occur.

The MC68349 provides $\overline{\text{BERR}}$ when no device responds by asserting $\overline{\text{DSACKx}}$/$\overline{\text{AVEC}}$ within an appropriate period of time after the MC68349 asserts $\overline{\text{AS}}$. This mechanism allows the cycle to terminate and the MC68349 to enter exception processing for the error condition. $\overline{\text{HALT}}$ is also used for bus exception control. This signal can be asserted by an external device for debugging purposes to cause single bus cycle operation, or, in combination with $\overline{\text{BERR}}$, a retry of a bus cycle in error. To properly control termination of a bus cycle for a retry or a bus error condition, $\overline{\text{DSACKx}}$, $\overline{\text{BERR}}$, and $\overline{\text{HALT}}$ can be asserted and negated with the rising edge of the MC68349 clock. This assures that when two signals are asserted simultaneously, the required setup and hold time for both is met for the same falling edge of the MC68349 clock. This or an equivalent precaution should be designed into the external circuitry to provide these signals. Alternatively, the internal bus monitor could be used. The acceptable bus cycle terminations for asynchronous cycles are summarized in relation to $\overline{\text{DSACKx}}$ assertion as follows (case numbers refer to Table 3-7):

- Normal Termination: $\overline{\text{DSACKx}}$ is asserted; $\overline{\text{BERR}}$ and $\overline{\text{HALT}}$ remain negated (case 1).
- Halt Termination: $\overline{\text{HALT}}$ is asserted at the same time as or before $\overline{\text{DSACKx}}$, and $\overline{\text{BERR}}$ remains negated (case 2).
- Bus Error Termination: $\overline{\text{BERR}}$ is asserted in lieu of, at the same time as, or before $\overline{\text{DSACKx}}$ (case 3) or after $\overline{\text{DSACKx}}$ (case 4), and $\overline{\text{HALT}}$ remains negated; $\overline{\text{BERR}}$ is negated at the same time as or after $\overline{\text{DSACKx}}$.
- Retry Termination: $\overline{\text{HALT}}$ and $\overline{\text{BERR}}$ are asserted in lieu of, at the same time as, or before $\overline{\text{DSACKx}}$ (case 5) or after $\overline{\text{DSACKx}}$ (case 6); $\overline{\text{BERR}}$ is negated at the same time as or after $\overline{\text{DSACKx}}$, and $\overline{\text{HALT}}$ may be negated at the same time as or after $\overline{\text{BERR}}$.

Table 3-8 lists various combinations of control signal sequences and the resulting bus cycle terminations. To ensure predictable operation, $\overline{\text{BERR}}$ and $\overline{\text{HALT}}$ should be negated according to the specifications given in **Section 11 Electrical Characteristics**. $\overline{\text{DSACKx}}$, $\overline{\text{BERR}}$, and $\overline{\text{HALT}}$ may be negated after $\overline{\text{AS}}$. If $\overline{\text{DSACKx}}$ or $\overline{\text{BERR}}$ remain asserted into S2 of the next bus cycle, that cycle may be terminated prematurely.

EXAMPLE A: A system uses a bus monitor timer to terminate accesses to an unpopulated address space. The timer asserts $\overline{\text{BERR}}$ after timeout (case 3).

EXAMPLE B: A system uses error detection and correction on random access memory (RAM) contents. The designer may:

1. Delay $\overline{\text{DSACKx}}$ until data is verified and assert $\overline{\text{BERR}}$ and $\overline{\text{HALT}}$ simultaneously to indicate to the MC68349 to automatically retry the error cycle (case 5), or if data is valid, assert $\overline{\text{DSACKx}}$ (case 1).
2. Delay $\overline{\text{DSACKx}}$ until data is verified and assert $\overline{\text{BERR}}$ with or without $\overline{\text{DSACKx}}$ if data is in error (case 3). This initiates exception processing for software handling of the condition.
3. Return $\overline{\text{DSACKx}}$ prior to data verification; if data is invalid, $\overline{\text{BERR}}$ is asserted on the next clock cycle (case 4). This initiates exception processing for software handling of the condition.
4. Return $\overline{\text{DSACKx}}$ prior to data verification; if data is invalid, assert $\overline{\text{BERR}}$ and $\overline{\text{HALT}}$ on the next clock cycle (case 6). The memory controller can then correct the RAM prior to or during the automatic retry.

Table 3-8. $\overline{\text{DSACKx}}$, $\overline{\text{BERR}}$, and $\overline{\text{HALT}}$ Assertion Results

Case Num	Control Signal	Asserted on Rising Edge of State		Result
		N	N + 2	
1	$\overline{\text{DSACKx}}$ $\overline{\text{BERR}}$ $\overline{\text{HALT}}$	A NA NA	S NA X	Normal cycle terminate and continue.
2	$\overline{\text{DSACKx}}$ $\overline{\text{BERR}}$ $\overline{\text{HALT}}$	A NA A/S	S NA S	Normal cycle terminate and halt; continue when $\overline{\text{HALT}}$ negated.
3	$\overline{\text{DSACKx}}$ $\overline{\text{BERR}}$ $\overline{\text{HALT}}$	NA/A A NA	X S X	Terminate and take bus error exception, possibly deferred.
4	$\overline{\text{DSACKx}}$ $\overline{\text{BERR}}$ $\overline{\text{HALT}}$	A NA NA	X A NA	Terminate and take bus error exception, possibly deferred.
5	$\overline{\text{DSACKx}}$ $\overline{\text{BERR}}$ $\overline{\text{HALT}}$	NA/A A A/S	X S S	Terminate and retry when $\overline{\text{HALT}}$ negated.
6	$\overline{\text{DSACKx}}$ $\overline{\text{BERR}}$ $\overline{\text{HALT}}$	A NA NA	X A A	Terminate and retry when $\overline{\text{HALT}}$ negated.

NOTES:
- N — Number of the current even bus state (e.g., S2, S4, etc.)
- A — Signal is asserted in this bus state
- NA — Signal is not asserted in this state
- X — Don't care
- S — Signal was asserted in previous state and remains asserted in this state

3.5.1 Bus Errors

$\overline{\text{BERR}}$ can be used to abort the bus cycle and the instruction being executed. $\overline{\text{BERR}}$ takes precedence over $\overline{\text{DSACKx}}$ provided it meets the timing constraints described in **Section 11 Electrical Characteristics**. If $\overline{\text{BERR}}$ does not meet these constraints, it may cause unpredictable operation of the MC68349. If $\overline{\text{BERR}}$ remains asserted into the next bus cycle, it may cause incorrect operation of that cycle. When $\overline{\text{BERR}}$ is issued to terminate a bus cycle, the MC68349 can enter exception processing immediately following the bus cycle, or it can defer processing the exception.

The instruction prefetch mechanism requests instruction words from the bus controller before it is ready to execute them. If a bus error occurs on an instruction fetch, the MC68349 does not take the exception until it attempts to use that instruction word. Should an intervening instruction cause a branch or should a task switch occur, the bus error exception does not occur. The bus error condition is recognized during a bus cycle in any of the following cases:

- $\overline{\text{DSACKx}}$ and $\overline{\text{HALT}}$ are negated, and $\overline{\text{BERR}}$ is asserted.
- $\overline{\text{HALT}}$ and $\overline{\text{BERR}}$ are negated, and $\overline{\text{DSACKx}}$ is asserted. $\overline{\text{BERR}}$ is then asserted within one clock cycle ($\overline{\text{HALT}}$ remains negated).
- $\overline{\text{BERR}}$ and $\overline{\text{HALT}}$ are asserted simultaneously, indicating a retry.

When the MC68349 recognizes a bus error condition, it terminates the current bus cycle in the normal way. Figure 3-29 shows the timing of a bus error for the case in which $\overline{\text{DSACKx}}$ is not asserted. Figure 3-30 shows the timing for a bus error that is asserted after $\overline{\text{DSACKx}}$. Exceptions are taken in both cases. Refer to **Section 5 CPU030** for details of bus error exception processing.

In the second case, in which $\overline{\text{BERR}}$ is asserted after $\overline{\text{DSACKx}}$ is asserted, $\overline{\text{BERR}}$ must be asserted within the time specified for purely asynchronous operation, or it must be asserted and remain stable during the sample window around the next falling edge of the clock after $\overline{\text{DSACKx}}$ is recognized. If $\overline{\text{BERR}}$ is not stable at this time, the MC68349 may exhibit erratic behavior. $\overline{\text{BERR}}$ has priority over $\overline{\text{DSACKx}}$. In this case, data may be present on the bus, but it may not be valid. This sequence can be used by systems that have memory error detection and correction logic and by external cache memories.

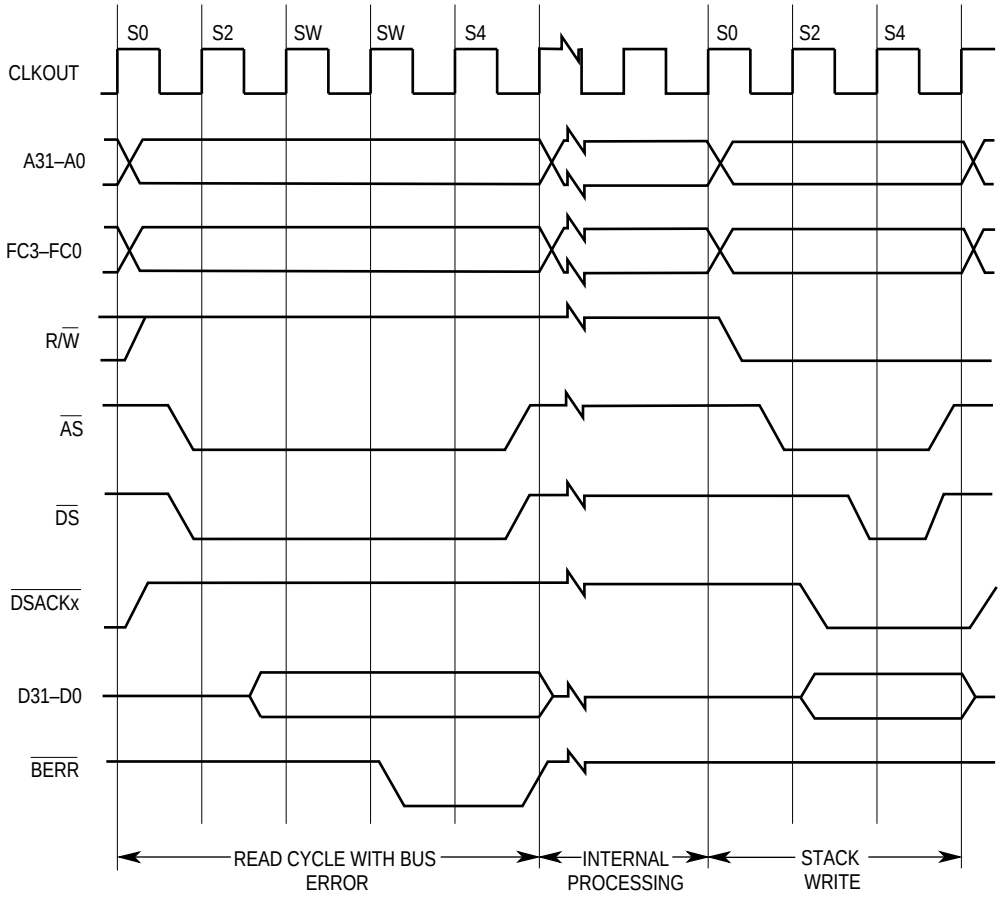


Figure 3-29. Bus Error without DSACKx Timing

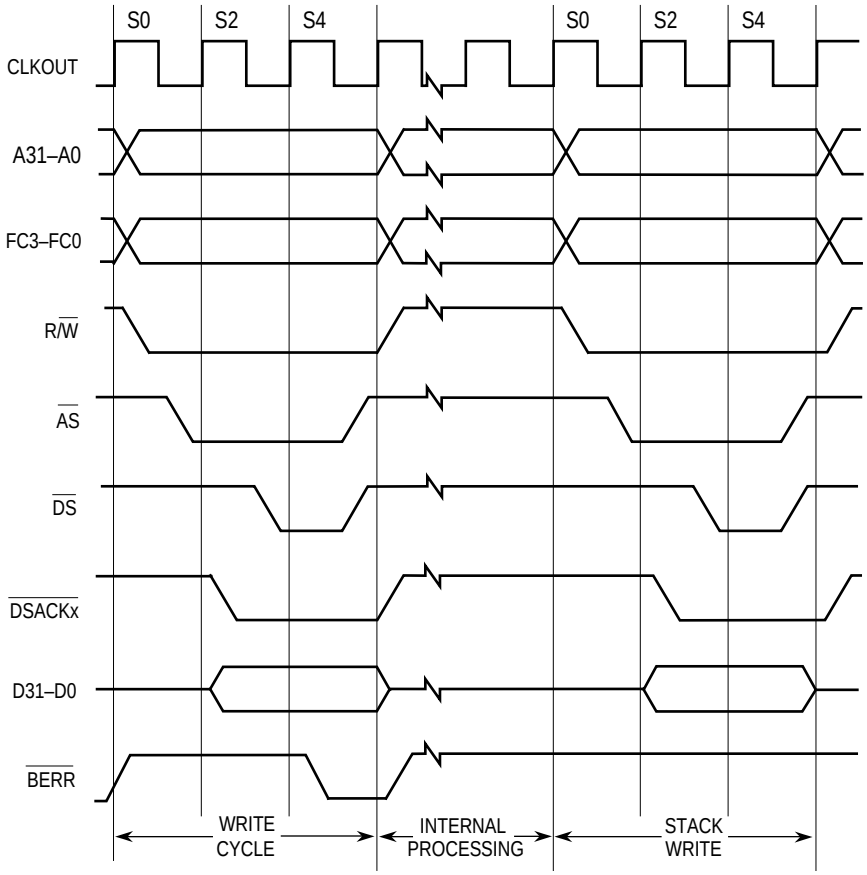


Figure 3-30. Late Bus Error with DSACKx Timing

3.5.2 Retry Operation

When both BERR and HALT are asserted by an external device during a bus cycle, the MC68349 enters the retry sequence shown in Figure 3-31. A delayed retry, which is similar to the delayed BERR signal described previously, can also occur (see Figure 3-32). The MC68349 terminates the bus cycle, places the control signals in their inactive state, and does not begin another bus cycle until the BERR and HALT signals are negated by external logic. After a synchronization delay, the MC68349 retries the previous cycle using the same access information (address, function code, size, etc.). BERR should be negated before S2 of the retried cycle to ensure correct operation of the retried cycle.

The MC68349 retries any read or write cycle of a read-modify-write operation separately; RMC remains asserted during the entire retry sequence. Asserting BR along with BERR and HALT provides a relinquish and retry operation. The MC68349 does not relinquish the bus during a read-modify-write operation. Any device that requires the MC68349 to give up the bus and retry a bus cycle during a read-modify-write cycle must assert only BERR and BR (HALT must not be included). The bus error handler software should examine the read-modify-write bit in the special status word (see **Section 5 CPU030**) and take the appropriate action to resolve this type of fault when it occurs.

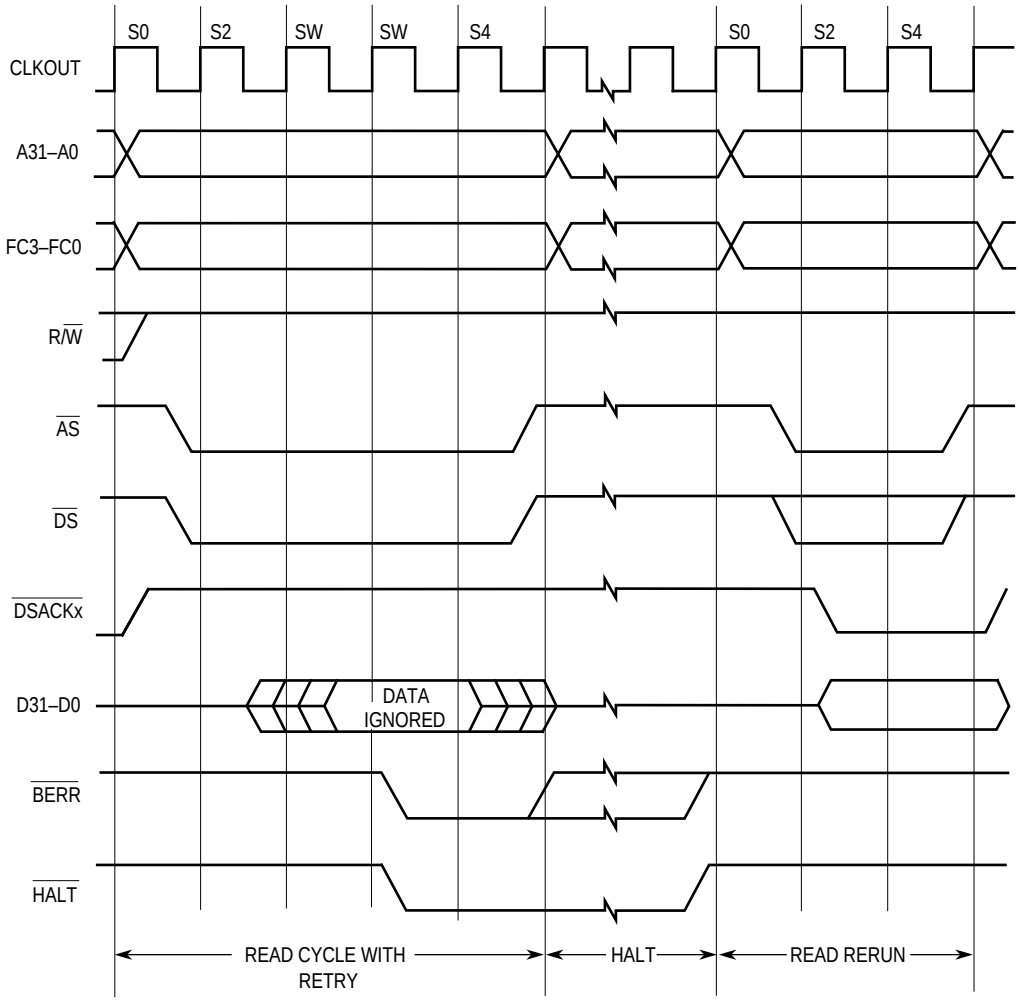


Figure 3-31. Retry Sequence Timing

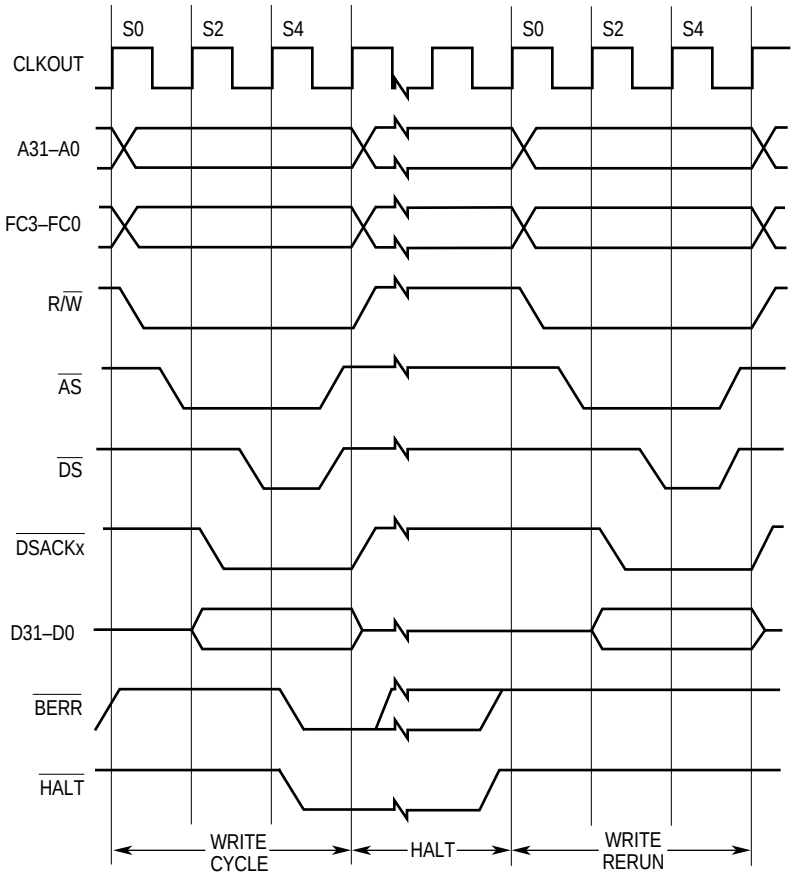


Figure 3-32. Late Retry Sequence Timing

3.5.3 Halt Operation

When $\overline{\text{HALT}}$ is asserted and $\overline{\text{BERR}}$ is not asserted, the MC68349 halts external bus activity at the next bus cycle boundary (see Figure 3-33). $\overline{\text{HALT}}$ by itself does not terminate a bus cycle. Negating and reasserting $\overline{\text{HALT}}$ in accordance with the correct timing requirements provides a single-step (bus cycle to bus cycle) operation. Since $\overline{\text{HALT}}$ affects external bus cycles only, a program that does not require use of the external bus may continue executing. The single-cycle mode allows the user to proceed through (and debug) external MC68349 operations, one bus cycle at a time. Since the occurrence of a bus error while $\overline{\text{HALT}}$ is asserted causes a retry operation, the user must anticipate retry cycles while debugging in the single-cycle mode. The single-step operation and the software trace capability allow the system debugger to trace single bus cycles, single instructions, or changes in program flow.

When the MC68349 completes a bus cycle with $\overline{\text{HALT}}$ asserted, D31–D0 is placed in the high-impedance state, and bus control signals are negated (not high-impedance state); the A31–A0, FCx, SIZx, and R/$\overline{\text{W}}$ signals remain in the same state. The halt operation has no effect on bus arbitration (see **3.6 Bus Arbitration**). When bus arbitration occurs while the MC68349 is halted, the address and control signals are also placed in the high-impedance state. Once bus mastership is returned to the MC68349, if $\overline{\text{HALT}}$ is still asserted, the A31–A0, FCx, SIZx, and R/$\overline{\text{W}}$ signals are again driven to their previous states. The MC68349 does not service interrupt requests while it is halted.

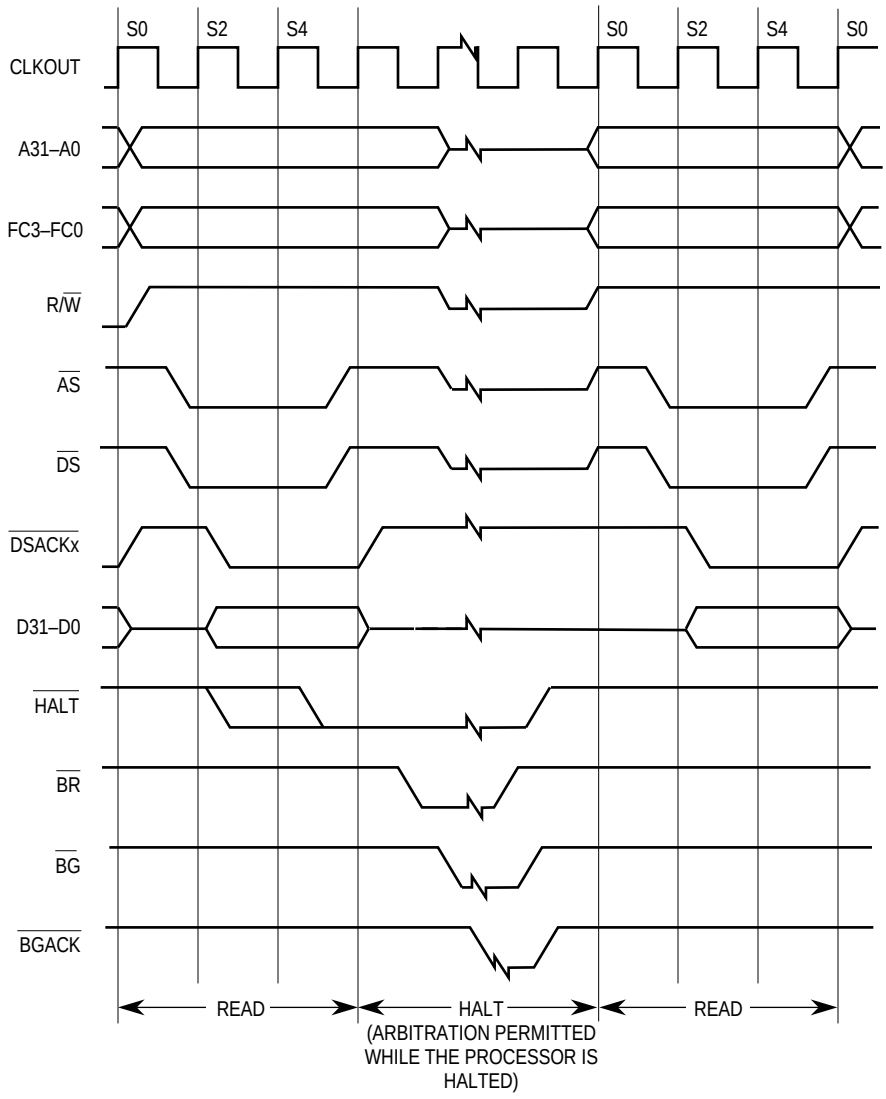


Figure 3-33. HALT Timing

3.5.4 Double Bus Fault

A double bus fault results when a bus error or an address error occurs during the exception processing sequence for any of the following:

- A previous bus error
- A previous address error
- A reset

For example, the MC68349 attempts to stack several words containing information about the state of the machine while processing a bus error exception. If a bus error exception occurs during the stacking operation, the second error is considered a double bus fault. When a double bus fault occurs, the MC68349 halts and asserts $\overline{\text{HALT}}$. Only a reset operation can restart a halted MC68349. However, bus arbitration can still occur (see **3.6 Bus Arbitration**). A second bus error or address error that occurs after exception processing has completed (during the execution of the exception handler routine or later) does not cause a double bus fault. A bus cycle that is retried does not constitute a bus error or contribute to a double bus fault. The MC68349 continues to retry the same bus cycle as long as the external hardware requests it.

Reset can also be generated internally by the halt monitor (see **Section 5 CPU030**).

3.6 BUS ARBITRATION

The bus design of the MC68349 provides for a single bus master at any one time, either the MC68349 or an external device. One or more of the external devices on the bus can have the capability of becoming bus master for the external bus, but not the MC68349 internal bus. Bus arbitration is the protocol by which an external device becomes bus master; the bus controller in the MC68349 manages the bus arbitration signals so that the MC68349 has the lowest priority. External devices that need to obtain the bus must assert the bus arbitration signals in the sequences described in the following paragraphs. Systems having several devices that can become bus master require external circuitry to assign priorities to the devices so that, when two or more external devices attempt to become bus master at the same time, the one having the highest priority becomes bus master first. The sequence of the protocol is as follows:

1. An external device asserts $\overline{\text{BR}}$.
2. The MC68349 asserts $\overline{\text{BG}}$ to indicate that the bus is available.
3. The external device asserts $\overline{\text{BGACK}}$ to indicate that it has assumed bus mastership.

NOTE

The MC68349 does not place $\overline{\text{CS3}}$–$\overline{\text{CS0}}$ in a high-impedance state after reset or when the bus is granted to an external master.

$\overline{BR}$ may be issued any time during a bus cycle or between cycles. $\overline{BG}$ is asserted in response to $\overline{BR}$. To guarantee operand coherency, $\overline{BG}$ is only asserted at the end of an operand transfer. Additionally, $\overline{BG}$ is not asserted until the end of a read-modify-write operation (when $\overline{RMC}$ is negated) in response to a $\overline{BR}$ signal. When the requesting device receives $\overline{BG}$ and more than one external device can be bus master, the requesting device should begin whatever arbitration is required. When the external device assumes bus mastership, it asserts $\overline{BGACK}$ and maintains $\overline{BGACK}$ during the entire bus cycle (or cycles) for which it is bus master. The following conditions must be met for an external device to assume mastership of the bus through the normal bus arbitration procedure: 1) it must have received $\overline{BG}$ through the arbitration process, and 2) $\overline{BGACK}$ must be inactive, indicating that no other bus master has claimed ownership of the bus.

Figure 3-34 is a flowchart showing bus arbitration for a single device. This technique allows processing of bus requests during data transfer cycles. Refer to Figures 3-35 and 3-36 for bus arbitration timing diagrams.

$\overline{BR}$ is negated at the time that $\overline{BGACK}$ is asserted. This type of operation applies to a system consisting of the MC68349 and one device capable of bus mastership. In a system having a number of devices capable of bus mastership, $\overline{BR}$ from each device can be wire-ORed to the MC68349. In such a system, more than one bus request could be asserted simultaneously. $\overline{BG}$ is negated a few clock cycles after the transition of $\overline{BGACK}$. However, if bus requests are still pending after the negation of $\overline{BG}$, the MC68349 asserts another $\overline{BG}$ within a few clock cycles after it was negated. This additional assertion of $\overline{BG}$ allows external arbitration circuitry to select the next bus master before the current bus master has finished using the bus. The following paragraphs provide additional information about the three steps in the arbitration process. Bus arbitration requests are recognized during normal processing, $\overline{HALT}$ assertion, and a CPU32+ halt caused by a double bus fault.

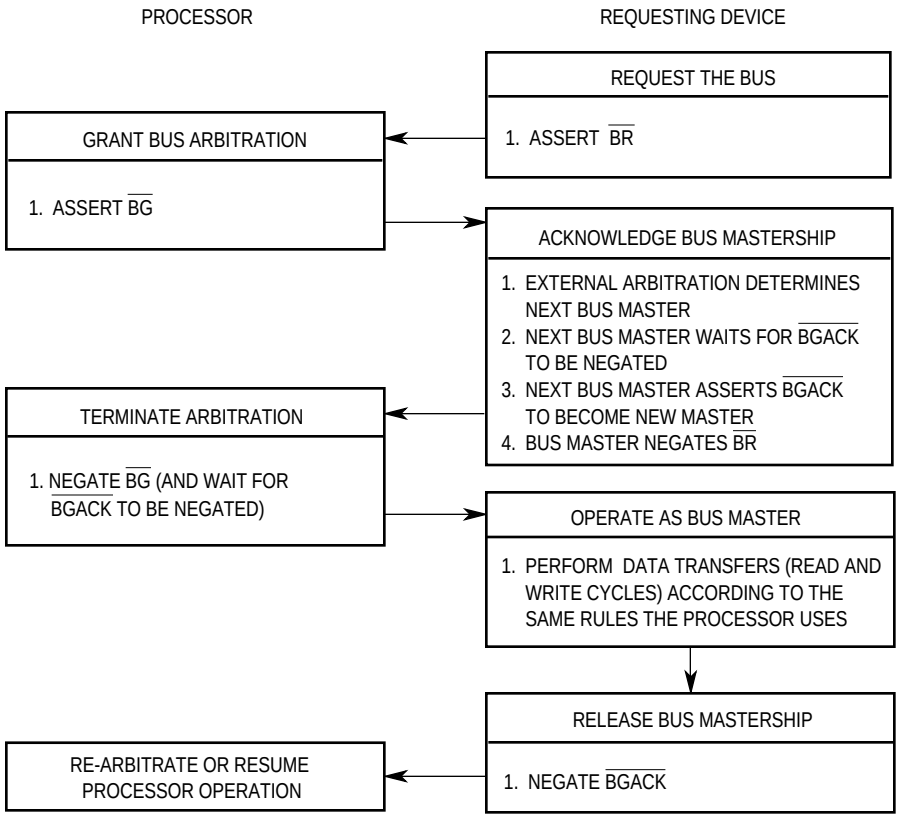


Figure 3-34. Bus Arbitration Flowchart for Single Request

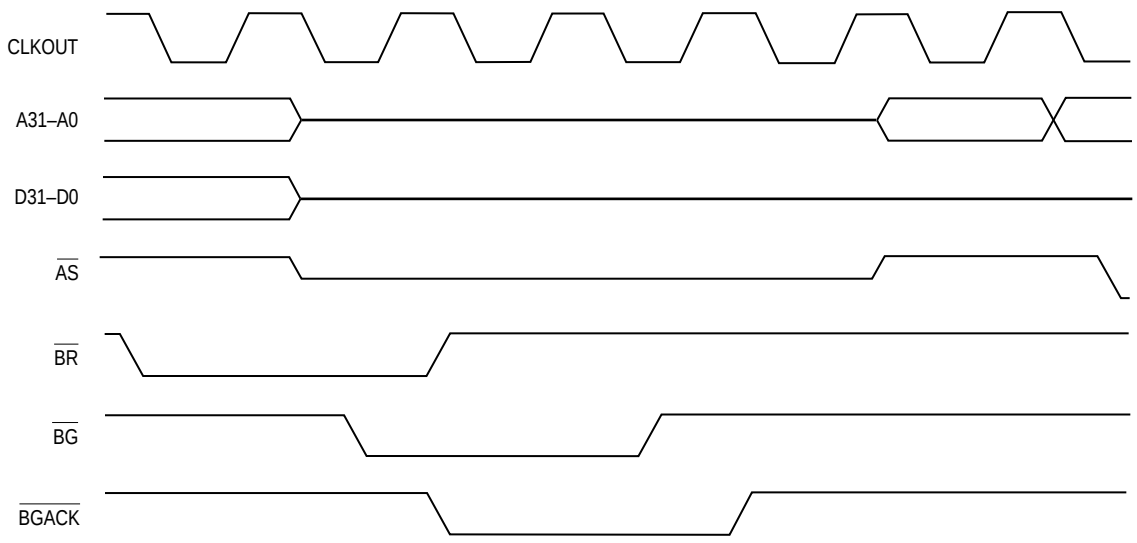


Figure 3-35. Bus Arbitration Timing—Idle Bus Case

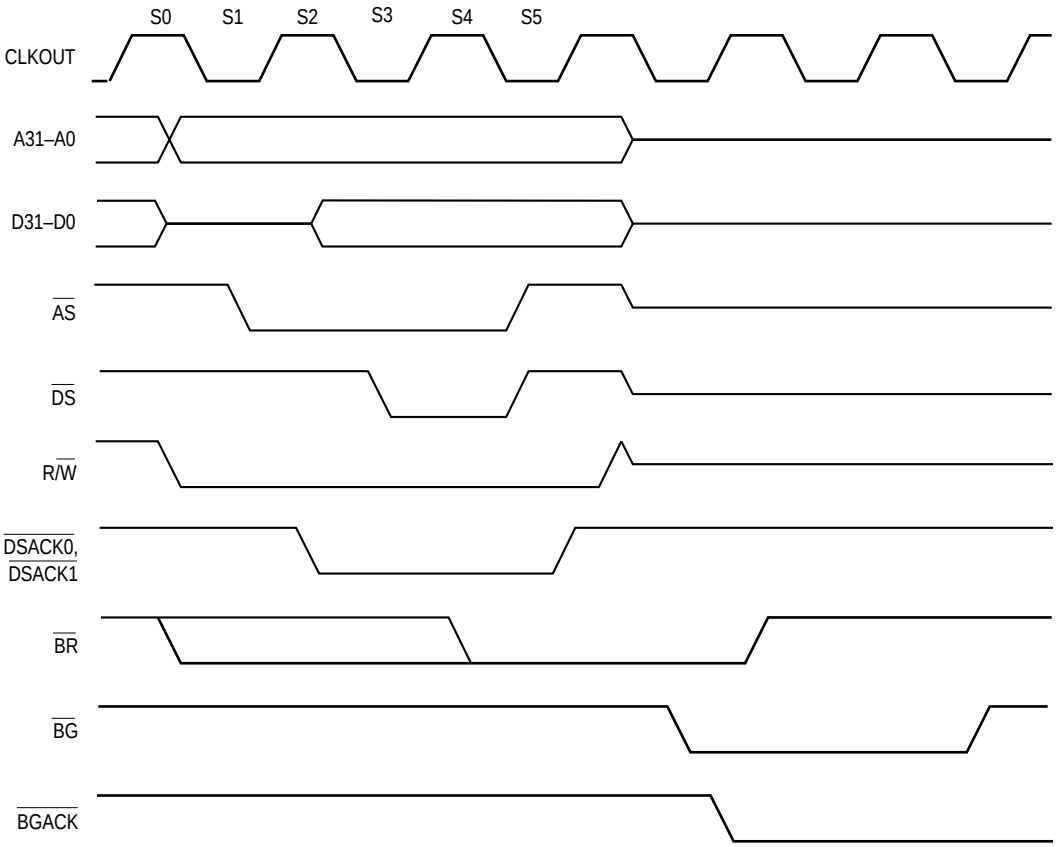


Figure 3-36. Bus Arbitration Timing—Active Bus Case

3.6.1 Bus Request

External devices capable of becoming bus masters request the bus by asserting $\overline{BR}$. This signal can be wire-ORed to indicate to the MC68349 that some external device requires control of the bus. The MC68349 is effectively at a lower bus priority level than the external device and relinquishes the bus after it has completed the current bus cycle (if one has started). If no $\overline{BGACK}$ is received while the $\overline{BR}$ is active, the MC68349 remains bus master once $\overline{BR}$ is negated. This prevents unnecessary interference with ordinary processing if the arbitration circuitry inadvertently responds to noise or if an external device determines that it no longer requires use of the bus before it has been granted mastership.

3.6.2 Bus Grant

The MC68349 supports operand coherency; thus, if an operand transfer requires multiple bus cycles, the MC68349 does not release the bus until the entire transfer is complete. Therefore, assertion of $\overline{BG}$ is subject to the following constraints:

- The minimum time for $\overline{BG}$ assertion after $\overline{BR}$ is asserted depends on internal synchronization (see **Section 11 Electrical Characteristics**).
- During an external operand transfer, the MC68349 does not assert $\overline{BG}$ until after the last cycle of the transfer (determined by SIZx and $\overline{DSACKx}$).
- During an external operand transfer, the MC68349 does not assert $\overline{BG}$ as long as $\overline{RMC}$ is asserted.
- If the show cycle bits SHEN1–SHEN0 = 01, the MC68349 does not assert $\overline{BG}$ to an external master.

Externally, the $\overline{BG}$ signal can be routed through a daisy-chained network or a priority-encoded network. The MC68349 is not affected by the method of arbitration as long as the protocol is obeyed.

3.6.3 Bus Grant Acknowledge

An external device cannot request and be granted the external bus while another device is the active bus master. A device that asserts $\overline{BGACK}$ remains the bus master until it negates $\overline{BGACK}$. $\overline{BGACK}$ should not be negated until all required bus cycles are completed. Bus mastership is terminated at the negation of $\overline{BGACK}$.

Once an external device receives the bus and asserts $\overline{BGACK}$, it should negate $\overline{BR}$. If $\overline{BR}$ remains asserted after $\overline{BGACK}$ is asserted, the MC68349 assumes that another device is requesting the bus and prepares to issue another $\overline{BG}$.

3.6.4 Bus Arbitration Control

The bus arbitration control unit in the MC68349 is implemented with a finite state machine. As discussed previously, all asynchronous inputs to the MC68349 are internally synchronized in a maximum of two cycles of the clock. As shown in Figure 3-37 input signals labeled R and A are internally synchronized versions of $\overline{BR}$ and $\overline{BGACK}$ respectively. The $\overline{BG}$ output is labeled G, and the internal high-impedance control signal is labeled T. If T is true, the address, data, and control buses are placed in the high-impedance state after the next rising edge following the negation of $\overline{AS}$ and $\overline{RMC}$. All signals are shown in positive logic (active high) regardless of their true active voltage level. The state machine shown in Figure 3-37 does not have a state 1 or state 4.

State changes occur on the next rising edge of the clock after the internal signal is valid. The $\overline{BG}$ signal transitions on the falling edge of the clock after a state is reached during which G changes. The bus control signals (controlled by T) are driven by the MC68349 immediately following a state change, when bus mastership is returned to the MC68349. State 0, in which G and T are both negated, is the state of the bus arbiter while the MC68349 is bus master. R and A keep the arbiter in state 0 as long as they are both negated.

The MC68349 does not allow arbitration of the external bus during the $\overline{RMC}$ sequence. For the duration of this sequence, the MC68349 ignores the $\overline{BR}$ input. If mastership of the bus is required during an $\overline{RMC}$ operation, $\overline{BERR}$ must be used to abort the $\overline{RMC}$ sequence.

3.6.5 Show Cycles

The MC68349 can perform data transfers with its internal modules without using the external bus, but, when debugging, it is desirable to have address and data information appear on the external bus. These external bus cycles, called show cycles, are distinguished by the fact that $\overline{AS}$ is not asserted externally. $\overline{DS}$ is used to signal address strobe timing in show cycles.

After reset, show cycles are disabled and must be enabled by writing to the SHENx bits in the module configuration register (see **Section 4 System Integration Module**). When show cycles are disabled, the A31–A0, FCx, SIZx, and R/$\overline{W}$ signals continue to reflect internal bus activity. However, $\overline{AS}$ and $\overline{DS}$ are not asserted externally, and the external data bus remains in a high-impedance state. When show cycles are enabled, $\overline{DS}$ indicates address strobe timing and the external data bus contains data. The following paragraphs are a state-by-state description of show cycles, and Figure 3-38 illustrates a show cycle timing diagram. Refer to **Section 11 Electrical Characteristics** for specific timing information.

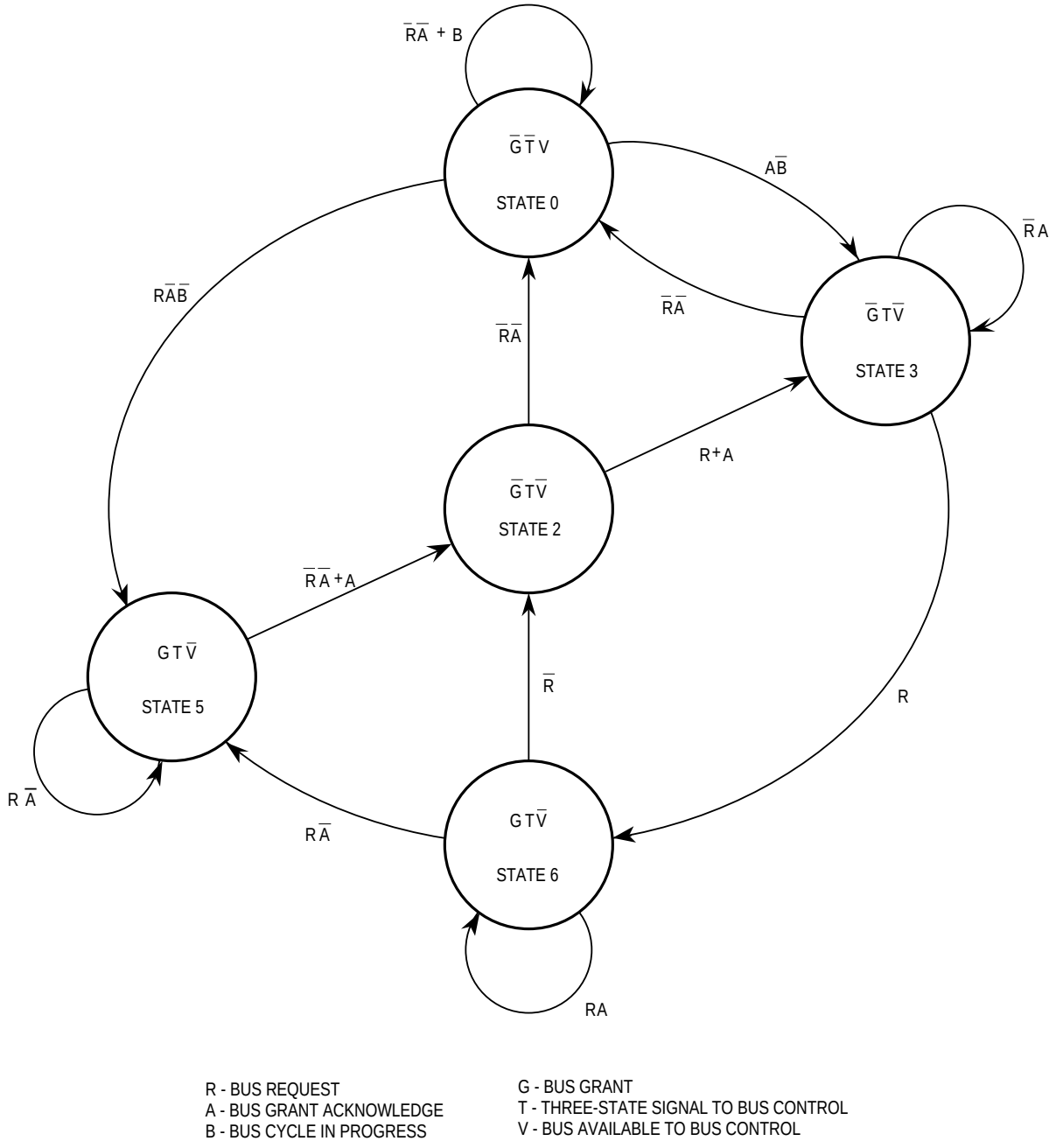


Figure 3-37. Bus Arbitration State Diagram

State 0—During state 0, the A31–A0 and FCx become valid, R/$\overline{W}$ is driven to indicate a show read or write cycle, and the SIZx pins indicate the number of bytes to transfer. During a read, the addressed peripheral is driving the data bus, and the user must take care to avoid bus conflicts.

State 41—One-half clock cycle later, $\overline{DS}$ (rather than $\overline{AS}$) is asserted to indicate that address information is valid.

State 42—No action occurs in state 42. The bus controller remains in state 42 (wait states will be inserted) until the internal read cycle is complete.

State 43—When $\overline{DS}$ is negated, show data is valid on the next falling edge of the system clock. The external data bus drivers are enabled so that data becomes valid on the external bus as soon as it is available on the internal bus.

State 0—The A31–A0, FCx, R/$\overline{W}$, and SIZx pins change to begin the next cycle. Data from the preceding cycle is valid through state 0.

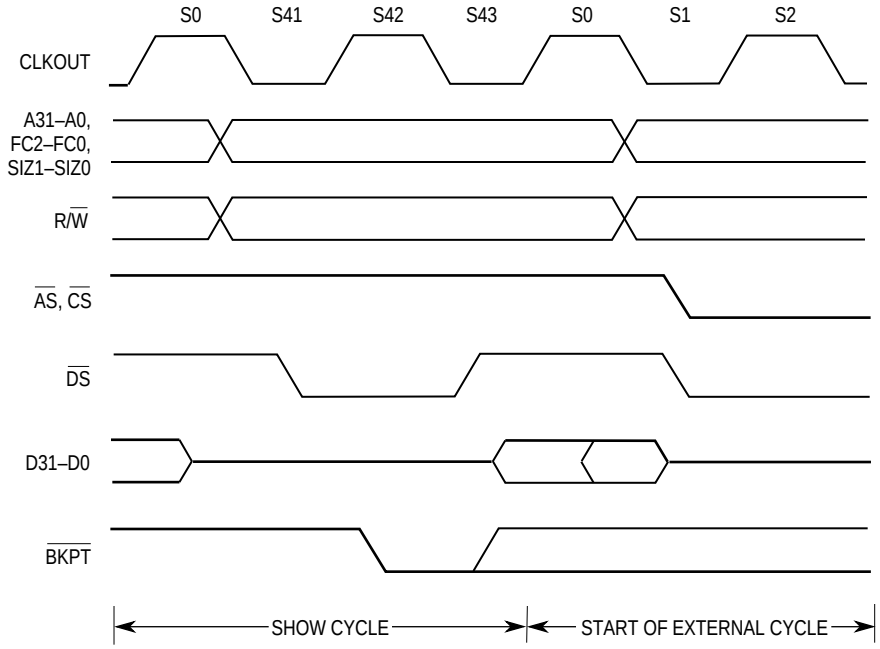


Figure 3-38. Show Cycle Timing

3.7 RESET OPERATION

The MC68349 has reset control logic to determine the cause of reset, synchronize it if necessary, and assert the appropriate reset lines. The reset control logic can independently drive three different lines:

1. EXTRST (external reset) drives the external $\overline{RESET}$ pin.
2. CLKRST (clock reset) resets the clock module.
3. INTRST (internal reset) goes to all other internal circuits.

Synchronous reset sources are not asserted until the end of the current bus cycle, whether or not $\overline{RMC}$ is asserted. The internal bus monitor is automatically enabled for synchronous resets; therefore, if the current bus cycle does not terminate normally, the bus monitor terminates it. Only single-byte or word transfers are guaranteed valid for synchronous resets. An external or clock reset is a synchronous reset source.

Asynchronous reset sources indicate a catastrophic failure, and the reset controller logic immediately resets the system. Resetting the MC68349 causes any bus cycle in progress to terminate as if $\overline{\text{DSACKx}}$ or $\overline{\text{BERR}}$ had been asserted. In addition, the MC68349 appropriately initializes registers for a reset exception. Asynchronous reset sources include power-up, software watchdog, double bus fault resets, and execution of the $\overline{\text{RESET}}$ instruction.

If an external device drives $\overline{\text{RESET}}$ low, $\overline{\text{RESET}}$ should be asserted for at least 590 clock periods to ensure that the MC68349 resets. The reset control logic holds reset asserted internally until the external $\overline{\text{RESET}}$ is released. When the reset control logic detects that external $\overline{\text{RESET}}$ is no longer being driven, it drives both internal and external reset low for an additional 512 cycles to guarantee this length of reset to the entire system. Figure 3-39 shows the $\overline{\text{RESET}}$ timing.

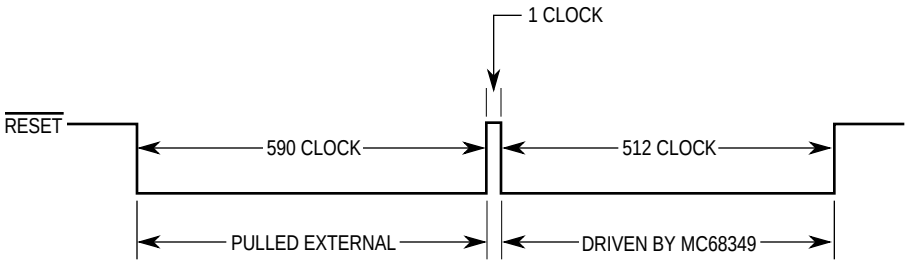
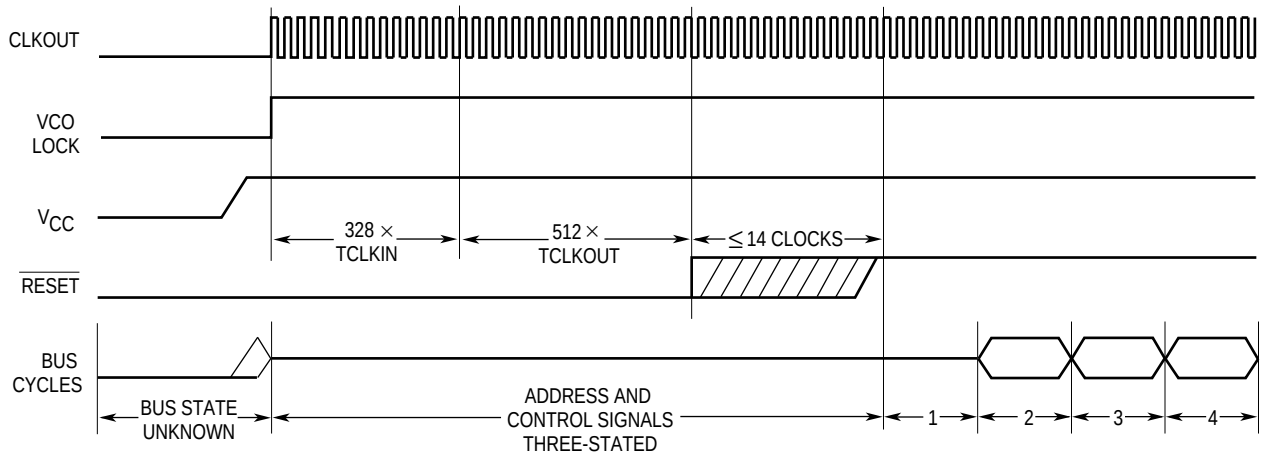


Figure 3-39. Timing for External Devices Driving $\overline{\text{RESET}}$

If reset is asserted from any other source, the reset control logic asserts $\overline{\text{RESET}}$ for 328 input clock periods plus 512 output clock periods, and until the source of reset is negated.

After any internal reset occurs, a 14-cycle rise time is allowed before testing for the presence of an external reset. If no external reset is detected, the CPU32+ begins its vector fetch.

Figure 3-40 is a timing diagram of the power-up reset operation, showing the relationships between $\overline{\text{RESET}}$, V_{CC}, and bus signals. During the reset period, the entire bus three-states except for non-three-statable signals, which are driven to their inactive state. Once $\overline{\text{RESET}}$ negates, all control signals are driven to their inactive state, the data bus is in read mode, and the address bus is driven. After this, the first bus cycle for $\overline{\text{RESET}}$ exception processing begins.



NOTES:
1. Internal start-up time.
2. SSP read here.
3. PC read here.
4. First instruction fetched here.

Figure 3-40. Power-Up Reset Timing

When a RESET instruction is executed, the MC68349 drives the $\overline{\text{RESET}}$ signal for 512 clock cycles. The SIM49 registers and the module control registers in each internal peripheral module (DMA, timers, and serial modules) are not affected. All other peripheral module registers are reset the same as for a hardware reset. The external devices connected to the $\overline{\text{RESET}}$ signal are reset at the completion of the RESET instruction.

SECTION 4
SYSTEM INTEGRATION MODULE

The MC68349 system integration module (SIM49) consists of several functions that control the system start-up, initialization, configuration, and the external bus with a minimum of external devices. It also provides the IEEE 1149.1 boundary scan capabilities. The SIM49 includes the following functions:

- System Configuration and Protection
- Clock Synthesizer
- Chip Selects and Wait States
- External Bus Interface
- Bus Arbitration
- Dynamic Bus Sizing
- IEEE 1149.1 Test Access Port

4.1 MODULE OVERVIEW

The SIM49 has similar features to the SIM in the MC68330, MC68331, MC68332, MC68333, and MC68340. The periodic interrupt timer, double bus fault monitor, software watchdog, internal bus monitor, and spurious interrupt monitor are identical. However, many of the other features in the SIM's differ in their use and details.

The system configuration and protection function controls system configuration and provides various monitors and timers, including the internal bus monitor, double bus fault monitor, spurious interrupt monitor, software watchdog timer, and the periodic interrupt timer.

The clock synthesizer generates the clock signals used by the SIM49 and the other on-chip modules, as well as CLKOUT used by external devices.

The programmable chip select function provides four chip select signals that can enable external memory and peripheral circuits, providing all handshaking and timing signals. Each chip select signal has an associated base address register and an address mask register that contain the programmable characteristics of that chip select. Up to six wait states can be programmed by setting bits in the base address and address mask registers.

The external bus interface (EBI) handles the transfer of information between the internal CPU32+ and memory, peripherals, or other processing elements in the external address space. See **Section 3 Bus Operation** for further information.

The MC68349 dynamically interprets the port size of an addressed device during each bus cycle, allowing operand transfers to or from 8-, 16-, and 32-bit ports. The device signals its port size and indicates completion of the bus cycle through the use of the $\overline{\text{DSACKx}}$ inputs. Dynamic bus sizing allows a programmer to write code that is not bus-width specific. For a discussion on dynamic bus sizing, see **Section 3 Bus Operation.**

The MC68349 includes dedicated user-accessible test logic that is fully compliant with the IEEE 1149.1 *Standard Test Access Port and Boundary Scan Architecture*. Problems associated with testing high-density circuit boards have led to the development of this standard under the sponsorship of the IEEE Test Technology Committee and Joint Test Action Group (JTAG). The MC68349 implementation supports circuit-board test strategies based on this standard. Refer to **Section 9 IEEE 1149.1 Test Access Port** for additional information.

4.2 MODULE OPERATION

The following paragraphs describe the operation of the module base address register, system configuration and protection, clock synthesizer, chip select functions, and the external bus interface.

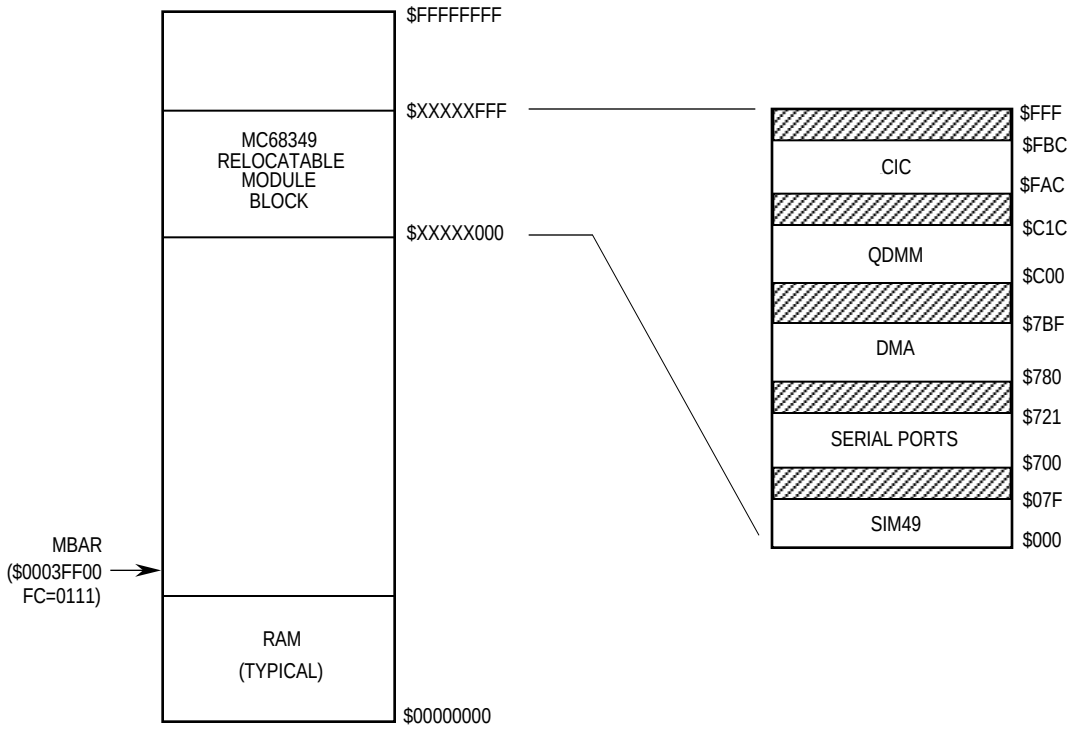
NOTE

The terms *assert* and *negate* are used throughout this section to avoid confusion when dealing with a mixture of active-low and active-high signals. The term *assert* or *assertion* indicates that a signal is active or true, independent of the level represented by a high or low voltage. The term *negate* or *negation* indicates that a signal is inactive or false.

4.2.1 Module Base Address Register Operation

The module base address register (MBAR) controls the location of all internal module registers (see **4.3.1 Module Base Address Register (MBAR)**). The address stored in this register is the base address (starting location) for all internal registers. All internal module registers are contained in a single 4-Kbyte block (see Figure 4-1) that is relocatable along 4-Kbyte boundaries.

The location of the internal registers is fixed by writing the desired base address of the 4-Kbyte block to the MBAR using the MOVES instruction to address $0003FF00 in CPU space. The source function code (SFC) and destination function code (DFC) registers contain the address space values (FC3–FC0) for the read or write operand of the MOVES instruction (see **Section 5 CPU030** or M68000PM/AD, *Programmer's Reference Manual*). Therefore, the SFC or DFC register must indicate CPU space (FC3–FC0 = $7), using the MOVEC instruction, before accessing MBAR. The offset from the base address is shown above each register diagram.



NOTE: $XXXXX is the value contained in the MBAR bits BA31-BA12.

Figure 4-1. SIM49 Module Register Block

4.2.2 System Configuration and Protection Operation

The SIM49 allows the user to control certain features of system configuration by writing bits in the module configuration register (MCR). This register also contains read-only status bits that show the state of the SIM49.

All M68000 family members are designed to provide maximum system safeguards. As an extension of this family, the MC68349 promotes the same basic concepts of safeguarded design present in all M68000 members. In addition, many functions that normally must be provided by external circuits are incorporated in this device. The following features are provided in the system configuration and protection function:

SIM49 Module Configuration

The SIM49 allows the user to configure the system to the particular requirements. The functions include control of FREEZE and show cycle operation, the function of the $\overline{\text{CSx}}$ signals, the access privilege of the supervisor/user registers, the level of interrupt arbitration, and automatic vectoring for external interrupts.

Reset Status

The reset status register provides the user with information on the cause of the most recent reset. The possible causes of reset include: external, power-up, software watchdog, double bus fault, loss of clock, and RESET instruction.

Internal Bus Monitor

The SIM49 provides an internal bus monitor to monitor the $\overline{\text{DSACKx}}$ response time for all internal bus accesses. An option allows the monitoring of external bus accesses. For external bus accesses, four selectable response times are provided to allow for variations in response speed of memory and peripherals used in the system. A bus error signal is asserted internally if the $\overline{\text{DSACKx}}$ response limit is exceeded. $\overline{\text{BERR}}$ is not asserted externally. This monitor can be disabled for external bus cycles only.

Double Bus Fault Monitor

The double bus fault monitor causes a reset to occur if the internal $\overline{\text{HALT}}$ is asserted by the CPU32+, indicating a double bus fault. A double bus fault results when a bus or address error occurs during the exception processing sequence for a previous bus or address error, a reset, or while the CPU32+ is loading information from a bus error stack frame during an RTE instruction. This function can be disabled. See **Section 3 Bus Operation** for more information.

Spurious Interrupt Monitor

If no interrupt arbitration occurs during an interrupt acknowledge (IACK) cycle, the bus error signal is asserted internally. This function cannot be disabled.

Software Watchdog

The software watchdog asserts $\overline{\text{RESET}}$ or a level 7 interrupt (as selected by the system protection and control register) if the software fails to service the software watchdog for a designated period of time (i.e., because it is trapped in a loop or lost). There are eight selectable timeout periods. This function can be disabled.

Periodic Interrupt Timer

The SIM49 provides a timer to generate periodic interrupts. The periodic interrupt time period can vary from 122 μs to 15.94 s (with a 32.768-kHz crystal used to generate the system clock). This function can be disabled.

Figure 4-2 shows a block diagram of the system configuration and protection function.

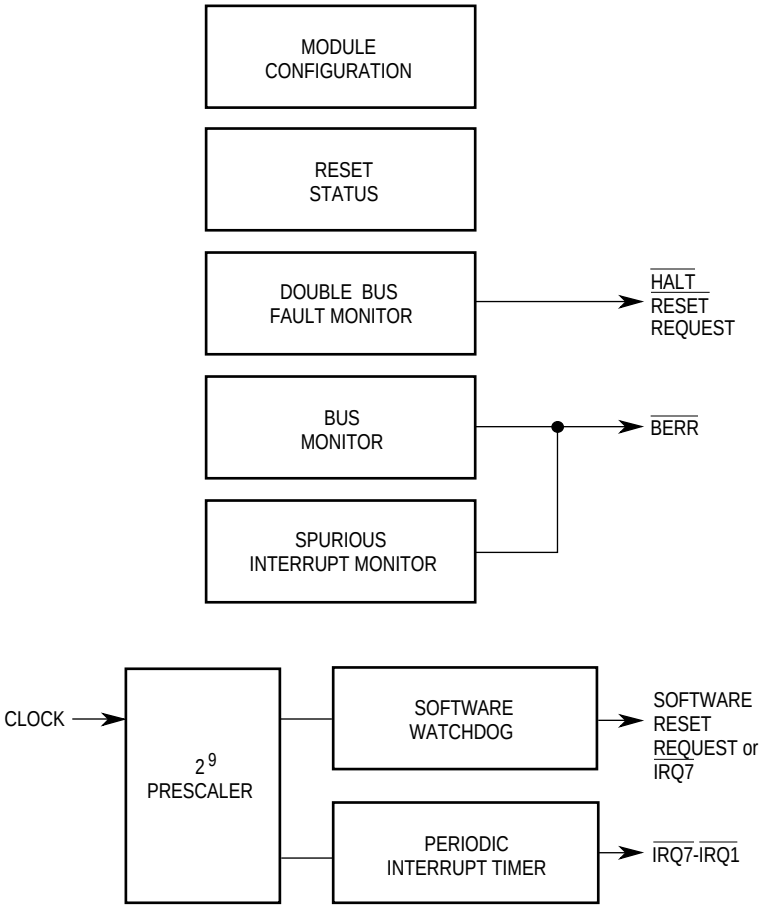


Figure 4-2. System Configuration and Protection Function

4.2.2.1 SYSTEM CONFIGURATION. Aspects of the system configuration are controlled by the MCR and the autovector register (AVR).

The configuration of port B is controlled by the combination of the FIRQ bit in the MCR and the port B pin assignment register (PPARB). Port B pins can function as dedicated I/O lines, chip selects, interrupts, or autovector input.

For debug purposes, internal bus accesses can be shown on the external bus. This function is called show cycles. The SHEN1, SHEN0 bits in the MCR control show cycles. Bus arbitration can be either enabled or disabled during show cycles.

Arbitration for servicing interrupts is controlled by the value programmed into the interrupt arbitration (IARB) field of the MCR. Each module that generates interrupts, including the SIM49, has an IARB field. The value of the IARB field allows arbitration during an IACK cycle among modules that simultaneously generate the same interrupt level. No two modules should share the same IARB value. The IARB must contain a value other than $0 for all modules that can generate interrupts; interrupts with IARB = 0 are discarded as extraneous. The SIM49 arbitrates for both its own interrupts and externally generated interrupts.

There are eight arbitration levels for access to the intermodule bus (IMB). The SIM49 is fixed at the highest level (above the programmable level 7), and the CPU32+ is fixed at the lowest level (below level 0). The direct memory access (DMA) module is the only other module that can become bus master and arbitrate for the bus. It must be initialized with a level other than 0 or 7.

The AVR contains bits that correspond to external interrupt levels that require an autovector response. The SIM49 supports up to seven discrete external interrupt requests. If the bit corresponding to an interrupt level is set in the AVR, the SIM49 returns an autovector in response to the IACK cycle servicing that external interrupt request. Otherwise, external circuitry must either return an interrupt vector or assert the external $\overline{\text{AVEC}}$ signal.

4.2.2.2 INTERNAL BUS MONITOR. The internal bus monitor continually checks for the bus cycle termination response time by checking the $\overline{\text{DSACK}}$x, $\overline{\text{BERR}}$, and $\overline{\text{HALT}}$ status or the $\overline{\text{AVEC}}$ status during an IACK cycle. The monitor initiates a bus error if the response time is excessive. The bus monitor feature cannot be disabled for internal accesses to an internal module. The internal bus monitor cannot check the $\overline{\text{DSACK}}$x response on the external bus unless the MC68349 is the bus master. The BME bit in the system protection control register (SYPCR) enables the internal bus monitor for internal-to-external bus cycles. If the system contains external bus masters whose bus cycles must be monitored, an external bus monitor must be implemented. In this case, the internal-to-external bus monitor option must be disabled.

The bus cycle termination response time is measured in clock cycles, and the maximum-allowable response time is programmable. The bus monitor response time period ranges from 64 to 512 system clocks (see Table 4-9). These options are provided to allow for different response times of peripherals that might be used in the system.

4.2.2.3 DOUBLE BUS FAULT MONITOR. A double bus fault is caused by a bus error or address error during the exception processing sequence. The double bus fault monitor responds to an assertion of $\overline{\text{HALT}}$ on the internal bus. Refer to **Section 3 Bus Operation** for more information. The DBF bit in the reset status register (RSR) indicates that the last reset was caused by the double bus fault monitor. The double bus fault monitor reset can be enabled by the DBFE bit in the SYPCR.

4.2.2.4 SPURIOUS INTERRUPT MONITOR. The spurious interrupt monitor issues $\overline{\text{BERR}}$ if no interrupt arbitration occurs during an IACK cycle. Normally, during an IACK cycle, one or more internal modules recognize that the CPU32+ is responding to interrupt request(s) and arbitrate for the privilege of returning a vector or asserting $\overline{\text{AVEC}}$. (The SIM49 reports and arbitrates for externally generated interrupts.) This feature cannot be disabled.

4.2.2.5 SOFTWARE WATCHDOG. The SIM49 provides a software watchdog option to prevent system lock-up in case the software becomes trapped in loops with no controlled exit. Once enabled by the SWE bit in the SYPCR, the software watchdog requires a special service sequence to be executed on a periodic basis. If this periodic servicing action does not occur, the software watchdog times out and issues a reset or a level 7 interrupt (as programmed by the SWRI bit in the SYPCR). The address of the interrupt service routine for the software watchdog interrupt is stored in the software interrupt vector register (SWIV). Figure 4-3 shows a block diagram of the software watchdog as well as the clock control circuits for the periodic interrupt timer.

The watchdog clock rate is determined by the SWP bit in the periodic interrupt timing register (PITR) and the SWT bits in the SYPCR. See Table 4-7 for a list of watchdog timeout periods.

The software watchdog service sequence consists of the following steps: 1) write $55 to the software service register (SWSR) and 2) write $AA to the SWSR. Both writes must occur in the order listed prior to the watchdog timeout, but any number of instructions or accesses to the SWSR can be executed between the two writes.

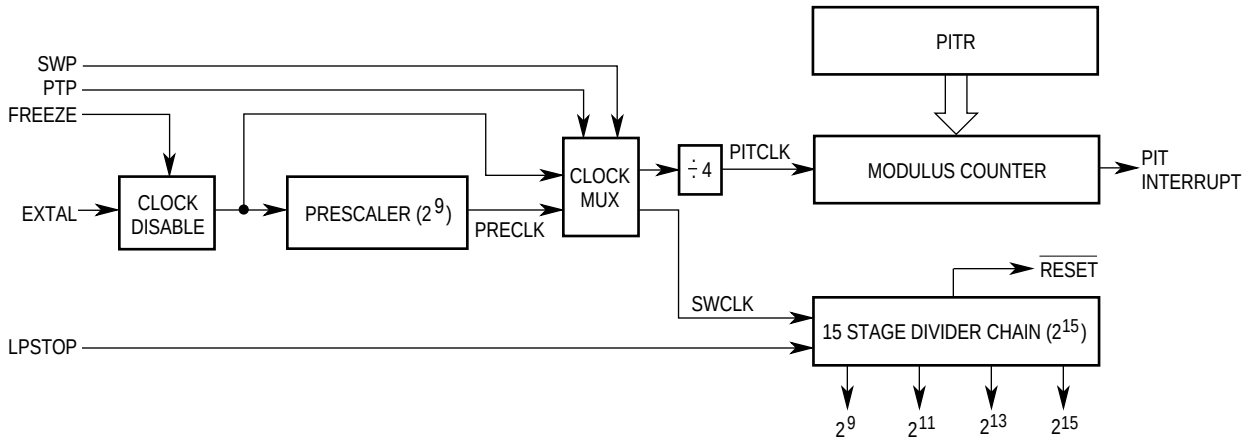


Figure 4-3. Software Watchdog Block Diagram

4.2.2.6 PERIODIC INTERRUPT TIMER. The periodic interrupt timer consists of an 8-bit modulus counter that is loaded with the value contained in the periodic interrupt timing register (see Figure 4-3). The modulus counter is clocked by a signal derived from the EXTAL input pin unless an external frequency source is used. When an external frequency source is used (MODCK low during reset), the default state of the prescaler control bits (SWP and PTP) in the PITR is changed to enable both prescalers.

Either clock source (EXTAL or EXTAL ÷ 512) is divided by 4 before driving the modulus counter (PITCLK). When the modulus counter value reaches zero, an interrupt is generated. The level of the generated interrupt is programmed into the PIRQL bits in the periodic interrupt control register (PICR). During the IACK cycle, the SIM49 places the periodic interrupt vector, programmed into the PIV bits in the PICR, onto the internal bus. The value of bits 7–0 in the PITR is then loaded again into the modulus counter, and the

counting process starts over. If a new value is written to the PITR, this value is loaded into the modulus counter when the current count is completed.

4.2.2.6.1 Periodic Timer Period Calculation. The period of the periodic timer can be calculated using the following equation:

$$\text{periodic interrupt timer period} = \frac{\text{PITR count value}}{\frac{\text{EXTAL frequency/prescaler value}}{2^2}}$$

Solving the equation using a crystal frequency of 32.768-kHz with the prescaler disabled gives:

$$\text{periodic interrupt timer period} = \frac{\text{PITR count value}}{\frac{32768/1}{2^2}}$$

$$\text{periodic interrupt timer period} = \frac{\text{PITR count value}}{8192}$$

This gives a range from 122 μs, with a PITR value of $01 (00000001 binary), to 31.128 ms, with a PITR value of $FF (11111111 binary).

Solving the equation with the prescaler enabled (PTP=1 in the PITR) gives the following values:

$$\text{periodic interrupt timer period} = \frac{\text{PITR count value}}{\frac{32768/512}{2^2}}$$

$$\text{periodic interrupt timer period} = \frac{\text{PITR count value}}{16}$$

This gives a range from 62.5 ms, with a PITR value of $01, to 15.94 s, with a PITR value of $FF.

For fast calculation of periodic timer period using a 32.768-kHz crystal, the following equations can be used:

With prescaler disabled:

programmable interrupt timer period = PITR (122 μs)

With prescaler enabled:

programmable interrupt timer period = PITR (62.5 ms)

4.2.2.6.2 Using the Periodic Timer as a Real-Time Clock. The periodic interrupt timer can be used as a real-time clock interrupt by setting it up to generate an interrupt with a one-second period. Rearranging the periodic timer period equation to solve for the desired count value:

$$\text{PITR count value} = \frac{\text{(PIT period) (EXTAL frequency)}}{\text{(Prescaler value) } (2^2)}$$

$$\text{PITR count value} = \frac{\text{(1) (32768)}}{\text{(512) } (2^2)}$$

$$\text{PITR count value} = 16 \text{ (decimal)}$$

Therefore, when using a 32.768-kHz crystal, the PITR should be loaded with a value of $10 with the prescaler enabled to generate interrupts at a one-second rate.

4.2.2.7 SIMULTANEOUS INTERRUPTS BY SOURCES IN THE SIM49. If multiple interrupt sources at the same interrupt level are simultaneously asserted in the SIM49, it will prioritize and service the interrupts in the following order: 1) software watchdog, 2) periodic interrupt timer, and 3) external interrupts.

4.2.3 Clock Synthesizer Operation

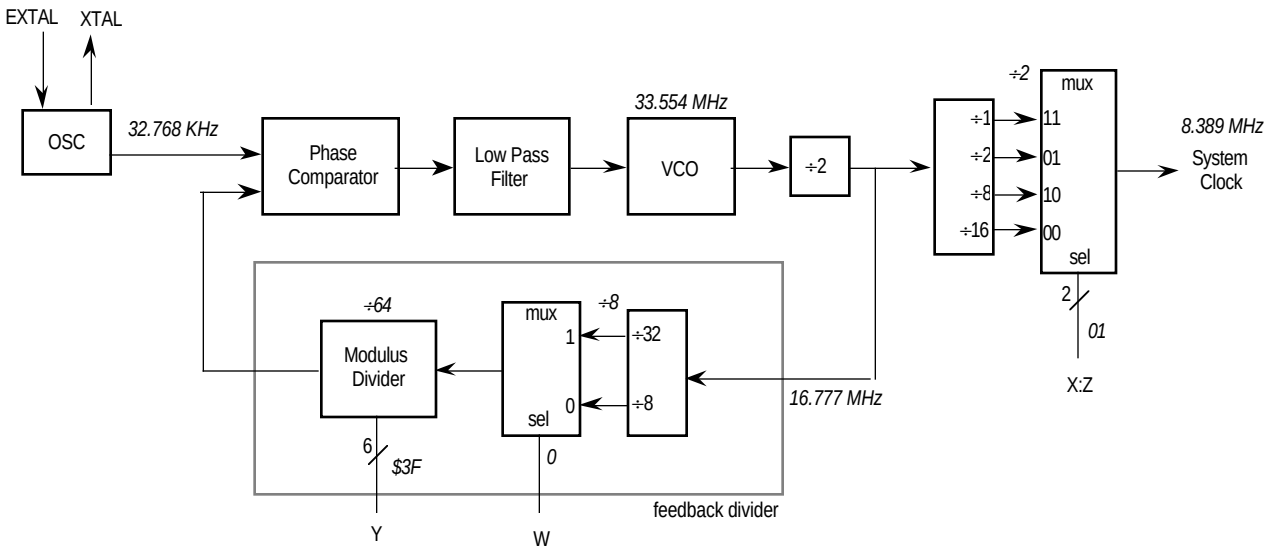
The clock synthesizer can operate with either an external crystal or an external oscillator for reference, using the internal phase-locked loop (PLL) and voltage-controlled oscillator (VCO), or an external clock can directly drive the clock signal at the operating frequency. The four modes of clock operation are listed in Table 4-1.

Table 4-1. Clock Operating Modes

Mode	Description	MODCK Reset Value (Logic Level)	V_{CCSYN} Operating Value (Voltage Level)
Crystal Mode	External crystal or oscillator used with the on-chip PLL and VCO to generate a system clock and CLKOUT of programmable rates.	1	V_{CC}
External Clock Mode without PLL	The desired operating frequency is driven into EXTAL resulting in a system clock and CLKOUT of the same frequency, not tightly coupled.	0	0 V
External Clock Mode with PLL	The desired operating frequency is driven into EXTAL, resulting in a system clock and CLKOUT of the same frequency, with a tight skew between input and output signals.	0	V_{CC}
Limp Mode	Upon input signal loss for either clock mode using the PLL, operation continues at approximately one-half operating speed (affected by the value of the X-bit in the SYNCR).	X	V_{CC}

In crystal mode (see Figure 4-4), the clock synthesizer can operate from the on-chip PLL and VCO, using a parallel resonant crystal connected between the EXTAL and XTAL pins, or an external oscillator connected to EXTAL as a reference frequency source. The oscillator circuit is shown in Figure 4-5. A 32.768-kHz watch crystal provides an inexpensive reference, but the reference crystal or external oscillator frequency can be any frequency in the range specified in **Section 11 Electrical Characteristics**. When using crystal mode, the system clock frequency is programmable (using the W, X, Y, and Z bits in the clock synthesizer control register (SYNCR)) over the range specified in **Section 11 Electrical Characteristics** (see Table 4-2.).

A separate power pin (V_{CCSYN}) is used to provide increased noise immunity for the clock circuits. The source for V_{CCSYN} should be a quiet power supply with adequate external bypass capacitors placed as close as possible to the V_{CCSYN} pin to ensure a stable operating frequency. Figure 4-4 shows typical values for the bypass and PLL external capacitors. The crystal manufacturer's documentation should be consulted for specific recommendations for external components.



Note: Default bit values after reset (and resulting frequencies) are shown in italics for VCO operation with a 32.768KHz crystal.

Figure 4-4. Clock Block Diagram for Crystal Operation

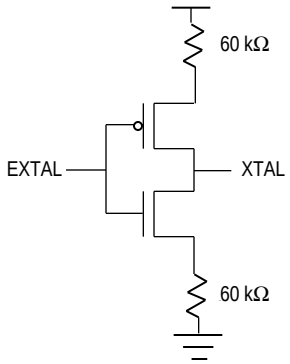


Figure 4-5. MC68349 Crystal Oscillator

To use an external clock source (see Figure 4-6), the operating clock frequency can be driven directly into the EXTAL pin (the XTAL pin must be left floating for this case). This approach results in a system clock and CLKOUT that are the same as the input signal frequency, but not tightly coupled to it. To enable this mode, MODCK must be held low during reset, and V_{CCSYN} held at 0 V while the chip is in operation.

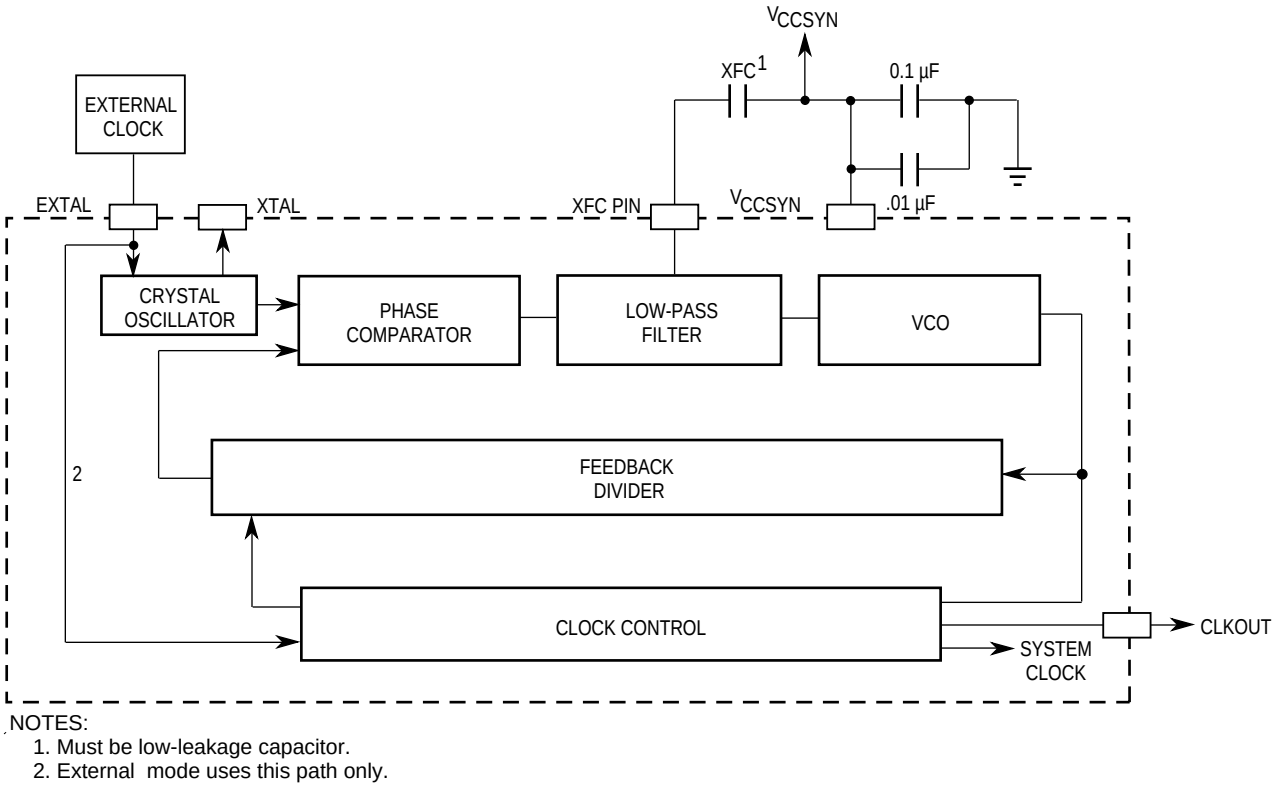


Figure 4-6. Clock Block Diagram for External Oscillator Operation

Alternatively, an external clock signal can be directly driven into EXTAL (with XTAL left floating) using the on-chip PLL. This configuration results in an internal clock and CLKOUT signal of the same frequency as the input signal, with a tight skew between the external clock and the internal clock and CLKOUT signals. To enable this mode, MODCK must be held low during reset, and V_{CCSYN} should be connected to a quiet 5-V source (or 3.3-V source for MC68349V).

If an input signal loss for either of the clock modes utilizing the PLL occurs, chip operation can continue in limp mode with the VCO running at approximately one-half the operating speed (affected by the values of the X- and Z-bits in the SYNCR), using an internal voltage reference. The SLIMP bit in the SYNCR indicates that a loss of input signal reference has been detected. The RSTEN bit in the SYNCR controls whether an input signal loss causes a system reset or causes the device to operate in limp mode. The SLOCK bit in the SYNCR indicates when the VCO has locked onto the desired frequency or if an external clock is being used.

4.2.3.1 PHASE COMPARATOR AND FILTER. The phase comparator takes the output of the frequency divider and compares it to an external input signal reference. The result of this compare is low-pass filtered and used to control the VCO. The comparator also detects when the external crystal or oscillator stops running to initiate the limp mode for the system clock.

The PLL requires an external low-leakage filter capacitor, typically in the range from 0.01 to 0.1 μF, connected between the XFC and V_{CCSYN} pins. The XFC capacitor should provide 50-MΩ insulation but should not be electrolytic. Smaller values of the external filter capacitor provide a faster response time for the PLL, and larger values provide greater frequency stability. For external clock mode without PLL, the XFC pin can be left open.

4.2.3.2 FREQUENCY DIVIDER. The frequency divider circuits divide the VCO frequency down to the reference frequency for the phase comparator. The frequency divider consists of 1) a 2-bit prescaler controlled by the W-bit in the SYNCR and 2) a 6-bit modulo downcounter controlled by the Y-bits in the SYNCR.

Several factors are important to the design of the system clock. The resulting system clock frequency must be within the limits specified for the device. The frequency of the system clock is given by the following equation:

$$F_{SYSTEM} = F_{CRYSTAL} [2^{(2W+X+3Z-1)}] \times (Y+1)$$

The maximum VCO frequency limit must also be observed. The VCO frequency is given by the following equations:

$$F_{VCO} = F_{CRYSTAL} [2^{(4+2W)}] \times (Y+1)$$

$$= F_{SYSTEM} [2^{(5-X-3Z)}]$$

Since setting the X-bit causes the VCO to run at twice the system frequency, the VCO upper frequency limit must be considered when programming the SYNCR. Both the system clock and VCO frequency limits are given in **Section 11 Electrical Characteristics**. Table 4-2 lists frequencies available from various combinations of SYNCR bits with a reference frequency of 32.768-KHz.

Table 4-2. System Frequencies from 32.768-kHz Reference

	CLKOUT (kHz)[1]				VCO (kHz)[1]	CLKOUT (kHz)[1]				VCO (kHz)[1]
	W = 0[2]				W = 0[2]	W = 1[2]				W = 1[2]
	Z = 0		Z = 1		Z = x	Z = 0		Z = 1		Z = x
Y[2]	X = 0	X = 1	X = 0	X = 1	X = x	X = 0	X = 1	X = 0	X = 1	X = x
0	16	33	131	262	524	66	1049	524	1049	2097
1	33	66	262	524	1049	131	2097	1049	2097	4194
2	49	98	393	786	1573	197	3146	1573	3146	6291
3	66	131	524	1049	2097	262	4194	2097	4194	8389
4	82	164	655	1311	2621	328	5243	2621	5243	10486
5	98	197	786	1573	3146	393	6291	3146	6291	12583
6	115	229	918	1835	3670	459	7340	3670	7340	14680
7	131	262	1049	2097	4194	524	8389	4194	8389	16777
8	147	295	1180	2359	4719	590	9437	4719	9437	18874
9	164	328	1311	2621	5243	655	10486	5243	10486	20972
10	180	360	1442	2884	5767	721	11534	5767	11534	23069
11	197	393	1573	3146	6291	786	12583	6291	12583	25166
12	213	426	1704	3408	6816	852	13631	6816	13631	27263
13	229	459	1835	3670	7340	918	14680	7340	14680	29360
14	246	492	1966	3932	7864	983	15729	7864	15729	31457
15	262	524	2097	4194	8389	1049	16777	8389	16777	33554
16	279	557	2228	4456	8913	1114	17826	8913	17826	35652
17	295	590	2359	4719	9437	1180	18874	9437	18874	37749
18	311	623	2490	4981	9961	1245	19923	9961	19923	39846
19	328	655	2621	5243	10486	1311	20972	10486	20972	41943
20	344	688	2753	5505	11010	1376	22020	11010	22020	44040
21	360	721	2884	5767	11534	1442	23069	11534	23069	46137
22	377	754	3015	6029	12059	1507	24117	12059	24117	48234
23	393	786	3146	6291	12583	1573	25166	12583	25166	50332
24	410	819	3277	6554	13107	1638	26214	13107	26214	52429
25	426	852	3408	6816	13631	1704	27263	13631	27263	54526
26	442	885	3539	7078	14156	1769	28312	14156	28312	56623
27	459	918	3670	7340	14680	1835	29360	14680	29360	58720
28	475	950	3801	7602	15204	1901	30409	15204	30409	60817
29	492	983	3932	7864	15729	1966	31457	15729	31457	62915
30	508	1016	4063	8126	16253	2032	32506	16253	32506	65012
31	524	1049	4194	8389	16777	2097	33554	16777	33554	67109

Table 4-2. System Frequencies from 32.768-kHz Reference (Continued)

	CLKOUT (kHz)				VCO (kHz)	CLKOUT (kHz)				VCO (kHz)
	W = 0				W = 0	W = 1				W = 1
	Z = 0		Z = 1		Z = x	Z = 0		Z = 1		Z = x
Y	X = 0	X = 1	X = 0	X = 1	X = x	X = 0	X = 1	X = 0	X = 1	X = x
32	541	1081	4325	8651	17302	2163	34603	17302	34603	69206
33	557	1114	4456	8913	17826	2228	35652	17826	35652	71303
34	573	1147	4588	9175	18350	2294	36700	18350	36700	73400
35	590	1180	4719	9437	18874	2359	37749	18874	37749	75497
36	606	1212	4850	9699	19399	2425	38797	19399	38797	77595
37	623	1245	4981	9961	19923	2490	39846	19923	39846	79692
38	639	1278	5112	10224	20447	2556	40894	20447	40894	81789
39	655	1311	5243	10486	20972	2621	41943	20972	41943	83886
40	672	1343	5374	10748	21496	2687	42992	21496	42992	85983
41	688	1376	5505	11010	22020	2753	44040	22020	44040	88080
42	705	1409	5636	11272	22544	2818	45089	22544	45089	90178
43	721	1442	5767	11534	23069	2884	46137	23069	46137	92275
44	737	1475	5898	11796	23593	2949	47186	23593	47186	94372
45	754	1507	6029	12059	24117	3015	48234	24117	48234	96469
46	770	1540	6160	12321	24642	3080	49283	24642	49283	98566
47	786	1573	6291	12583	25166	3146	50332	25166	50332	100663
48	803	1606	6423	12845	25690	3211	51380	25690	51380	102760
49	819	1638	6554	13107	26214	3277	52429	26214	52429	104858
50	836	1671	6685	13369	26739	3342	53477	26739	53477	106955
51	852	1704	6816	13631	27263	3408	54526	27263	54526	109052
52	868	1737	6947	13894	27787	3473	55575	27787	55575	111149
53	885	1769	7078	14156	28312	3539	56623	28312	56623	113246
54	901	1802	7209	14418	28836	3604	57672	28836	57672	115343
55	918	1835	7340	14680	29360	3670	58720	29360	58720	117441
56	934	1868	7471	14942	29884	3736	59769	29884	59769	119538
57	950	1901	7602	15204	30409	3801	60817	30409	60817	121635
58	967	1933	7733	15466	30933	3867	61866	30933	61866	123732
59	983	1966	7864	15729	31457	3932	62915	31457	62915	125829
60	999	1999	7995	15991	31982	3998	63963	31982	63963	127926
61	1016	2032	8126	16253	32506	4063	65012	32506	65012	130023
62	1032	2064	8258	16515	33030	4129	66060	33030	66060	132121
63	1049	2097	8389	16777	33554	4194	67109	33554	67109	134218

NOTES:

1. Some W/X/Y bit combinations shown may select a CLKOUT or VCO frequency higher than spec. Refer to **Section 11 Electrical Characteristics** for CLKOUT and VCO frequency limits.
2. Any change to W or Y results in a change in the VCO frequency - the VCO should be allowed to relock if necessary

4.2.3.3 CLOCK CONTROL. The clock control circuits determine the source used for both internal and external clocks during special circumstances, such as low-power stop (LPSTOP) execution.

Table 4-3 summarizes the clock activity during LPSTOP in crystal mode operation. Any clock in the off state is held low. The STEXT and STSIM bits in the SYNCR control clock activity during LPSTOP. Refer to **4.2.6 Low-Power Stop** for additional information.

Table 4-3. Clock Control Signals

Control Bits		Clock Outputs	
STSIM	**STEXT**	**SIMCLK**	**CLKOUT**
0	0	EXTAL	Off
0	1	EXTAL	EXTAL
1	0	VCO	Off
1	1	VCO	VCO

NOTE: SIMCLK runs the periodic interrupt $\overline{\text{RESET}}$ and $\overline{\text{IRQx}}$ pin synchronizers in LPSTOP mode.

4.2.4 Chip Select Operation

Typical microprocessor systems require external hardware to provide select signals to external memory and peripherals. The MC68349 integrates these functions on chip to provide the cost, speed, and reliability benefits of a higher level of integration. The chip select function contains register pairs for each external chip select signal. The pair consists of a base address register and an address mask register that define the characteristics of a single chip select. The register pair provides flexibility for a wide variety of chip select functions.

4.2.4.1 PROGRAMMABLE FEATURES. The chip select function supports the following programmable features:

Four Programmable Chip Select Circuits

All four chip select circuits are independently programmable from the same list of selectable features. Each chip select circuit has an individual base address register and address mask register that contain the programmed characteristics of that chip select. The base address register selects the starting address for the address block in 256-byte increments. The address mask register specifies the size of the address block range. The base address register V-bit indicates that the register information for that chip select is valid. A global chip select allows address decode for a boot ROM before system initialization occurs (see **4.2.4.2 Global Chip Select Operation**).

Variable Block Sizes

The block size, starting from the specified base address, can vary in size from 256 bytes up to 4 Gbytes in 2^n increments. The specified base address must be on a multiple of the block size. The block size is specified in the address mask register.

8-, 16-. and 32-Bit Ports Supported

The 8-bit ports are accessible on both odd and even addresses when connected to D31–D24. The 16-bit ports can be accessed as odd bytes, even bytes, or even words when connected to D31–D16. The 32-bit ports can be accessed as odd or even bytes, odd or even words, or odd or even longwords. The port size is specified by the PS bits in the address mask register.

Write Protect Capability

The WP bit in each base address register can restrict write access to its range of addresses.

Fast Termination Option

Programming the EDS bit (base address register) and the DD1 and DD2 bits (address mask register) to a value of 111 selects the fast termination option. The fast termination option causes the chip select to terminate the cycle by asserting the internal $\overline{\text{DSACKx}}$ early, providing a two-cycle external access.

Internal $\overline{\text{DSACKx}}$ Generation for External Accesses with Programmable Wait States

$\overline{\text{DSACKx}}$ can be generated internally with up to six wait states for a particular device using the EDS bit in the base address register along with the DDx bits in the address mask register.

Full 32-Bit Address Decode with Address Space Checking

The FC bits in the base address register and FCM bits in the address mask register are used to select address spaces for which the chip selects will be asserted.

4.2.4.2 GLOBAL CHIP SELECT OPERATION. Global chip select operation allows address decode for a boot ROM before system initialization occurs. The size of the global chip select is programmable and is determined by the state of data bus signals D31 and D30 at reset (see Table 4-4).

Table 4-4. D31, D30 Encoding for Global Chip Select Size

D31	D30	Global Chip Select Size
0	0	32-Bit
0	1	16-Bit
1	0	8-Bit
1	1	External $\overline{\text{DSACKx}}$ Response

4.2.5 External Bus Interface Operation

This section describes port A and port B functions. Refer to **Section 3 Bus Operation** for more information about the EBI.

4.2.5.1 PORT A. Port A pins can be independently programmed to function as either addresses A31–A24, discrete I/O pins, or $\overline{\text{IACK}}$x pins. The port A pin assignment registers (PPARA1 and PPARA2) control the function of the port A pins as listed in Table 4-5. Upon reset, the port A pins are initialized as either input pins or address outputs, dependent on the state of the D29 pin at reset. If D29 = 1 at reset, the port A pins are initialized as address outputs; if D29 = 0 at reset, the port A pins are initialized as inputs.

Table 4-5. Port A Pin Assignment Register

Signal	Pin Function		
	PPARA1 = 0 PPARA2 = 0 (D29 = 1 At Reset)	PPARA1 = 1 PPARA2 = X (D29 = 0 At Reset)	PPARA1 = 0 PPARA2 = 1
A31	A31	PORT A7	$\overline{\text{IACK7}}$
A30	A30	PORT A6	$\overline{\text{IACK6}}$
A29	A29	PORT A5	$\overline{\text{IACK5}}$
A28	A28	PORT A4	$\overline{\text{IACK4}}$
A27	A27	PORT A3	$\overline{\text{IACK3}}$
A26	A26	PORT A2	$\overline{\text{IACK2}}$
A25	A25	PORT A1	$\overline{\text{IACK1}}$
A24	A24	PORT A0	—

4.2.5.2 PORT B. Port B pins can be independently programmed to function as chip selects, $\overline{IRQx}$ and MODCK pins, or discrete I/O pins. These pins are multiplexed as shown in Figure 4-7. Selection of a pin function is accomplished by a combination of the PPARB and the FIRQ bit of the MCR. See Table 4-6 for port B combinations. By changing the value of the FIRQ bit and the corresponding bits in the PPARB for a particular signal, the port B pins can be configured for different pin functions. The MODCK signal is used only during reset. After reset, port B is configured as PORTB7, PORTB6, PORTB5, PORTB3, and $\overline{CS3}$–$\overline{CS0}$, and PORTB0.

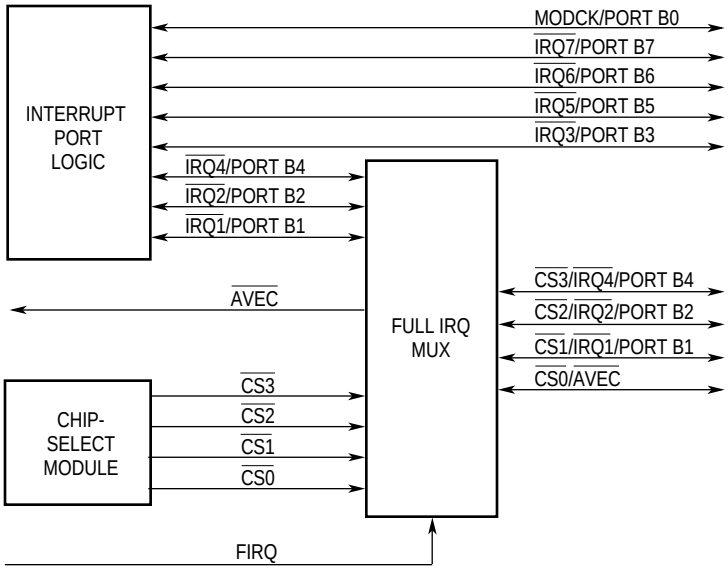


Figure 4-7. Full Interrupt Request Multiplexer

Table 4-6. Port B Pin Assignment Register

Signal	Pin Function			
	FIRQ = 0 PPARB = 0	FIRQ = 0 PPARB = 1	FIRQ = 1 PPARB = 0	FIRQ = 1 PPARB = 1
$\overline{IRQ7}$	PORTB7	$\overline{IRQ7}$	PORTB7	$\overline{IRQ7}$
$\overline{IRQ6}$	PORTB6	$\overline{IRQ6}$	PORTB6	$\overline{IRQ6}$
$\overline{IRQ5}$	PORTB5	$\overline{IRQ5}$	PORTB5	$\overline{IRQ5}$
$\overline{IRQ3}$	PORTB3	$\overline{IRQ3}$	PORTB3	$\overline{IRQ3}$
$\overline{CS3}$	$\overline{CS3}$	$\overline{CS3}$	PORTB4	$\overline{IRQ4}$
$\overline{CS2}$	$\overline{CS2}$	$\overline{CS2}$	PORTB2	$\overline{IRQ2}$
$\overline{CS1}$	$\overline{CS1}$	$\overline{CS1}$	PORTB1	$\overline{IRQ1}$
$\overline{CS0}$	$\overline{CS0}$	$\overline{CS0}$	$\overline{AVEC}$	$\overline{AVEC}$
MODCK	PORTB0	—	PORTB0	—

NOTE: MODCK is used only during reset.

The number of wait states programmed into the internal wait state generation logic by a chip select can be used even though the pin is not used as a $\overline{CSx}$ signal. The programmed number of wait states in the $\overline{CSx}$ signal applies to the port B pins configured as $\overline{IRQx}$ or I/O pins. This is done by programming the chip select with the number of wait states to be added, as though it were to be used. The EDS bit in the base address register along with the DD1/DD0 and PS1/PS0 bits in the chip select address mask register must be set to add the desired number of wait states (the V-bit in the module base address register must also be set).

4.2.6 Low-Power Stop

Executing the LPSTOP instruction provides reduced power consumption when the MC68349 is idle; only the SIM49 remains active. Operation of the SIM49 clock and CLKOUT during LPSTOP is controlled by the STSIM and STEXT bits in the SYNCR (see Table 4-3). LPSTOP disables the clock to the software watchdog in the low state. The software watchdog remains stopped until the LPSTOP mode ends; it begins to run again on the next rising clock edge.

NOTE

When the CPU32+ executes the STOP instruction (as opposed to LPSTOP), the software watchdog continues to run. If the software watchdog is enabled, it issues a reset or interrupt when timeout occurs.

The periodic interrupt timer does not respond to an LPSTOP instruction; thus, it can be used to exit LPSTOP as long as the interrupt request level is higher than the CPU32+ interrupt mask level. To stop the periodic interrupt timer while in LPSTOP, the PITR must be loaded with a zero value before LPSTOP is executed. The bus monitor, double bus fault monitor, and spurious interrupt monitor are all inactive during LPSTOP.

The STP bit in the MCR of each on-chip module (DMA, quad data memory, and serial modules) should be set prior to executing the LPSTOP instruction. Setting the STP bit stops all clocks within each of the modules, except for the clock from the IMB. The clock from the IMB remains active to allow the CPU32+ access to the MCR of each module. The system clock stops on the low phase of the clock and remains stopped until the STP bit is cleared by the CPU32+ or until reset. For more information, see the description of the MCR STP bit for each module.

If an external device requires additional time to prepare for entry into LPSTOP mode, entry can be delayed by asserting $\overline{HALT}$ (see **Section 3 Bus Operation**).

4.2.7 Freeze

The FREEZE signal is asserted by the CPU32+ if a breakpoint is encountered with background mode enabled. Refer to **Section 5 CPU030** for more information on the background mode. When FREEZE is asserted, the double bus fault monitor and spurious interrupt monitor continue to operate normally. However, the software watchdog, the periodic interrupt timer and the internal bus monitor will be affected. When FREEZE is asserted, setting the FRZ1 bit in the MCR disables the software watchdog and periodic interrupt timer and setting the FRZ0 bit in the MCR disables the bus monitor.

4.3 PROGRAMMING MODEL

Figure 4-8 is a programming model (register map) of all the registers in the SIM49. For more information about a particular register, refer to the description of the module or function indicated in the right column. The ADDR (address) column indicates the offset of the register from the address stored in the MBAR. The FC (function code) column indicates whether a register is restricted to supervisor access (S) or programmable to exist in either supervisor or user space (S/U).

The SIM49 registers are discussed in the following pages In the register diagrams, the number in the upper right-hand corner indicates the offset of the register from the address stored in the MBAR. The numbers on the top line of the register represent the bit position in the register. The second line contains the mnemonic for the bit. The numbers below the register represent the bit values after a hardware reset. The access privilege is indicated in the lower right-hand corner.

NOTE:

A CPU32+ RESET instruction will not affect any of the SIM49 registers.

The IMB data port to the SIM49 registers is 16 bits wide. Long-word accesses to registers are allowed, but are dynamically sized to two 16-bit accesses.

ADDR	FC	15 — 8	7 — 0	
000	S	MODULE CONFIGURATION REGISTER (MCR)		SYSTEM PROTECTION
002	S/U	IDENTIFICATION REGISTER (IDR)		IDENTIFICATION
004	S	CLOCK SYNTHESIZER CONTROL REGISTER (SYNCR)		CLOCK
006	S	AUTOVECTOR REGISTER (AVR)	RESET STATUS REGISTER (RSR)	SYSTEM PROTECTION
010	S/U	RESERVED	PORT A DATA (PORTA)	EBI
012	S/U	RESERVED	PORT A DATA DIRECTION (DDRA)	EBI
014	S	RESERVED	PORT A PIN ASSIGNMENT 1 (PPRA1)	EBI
016	S	RESERVED	PORT A PIN ASSIGNMENT 2 (PPRA2)	EBI
018	S/U	RESERVED	PORT B DATA (PORTB)	EBI
01A	S/U	RESERVED	PORT B DATA (PORTB1)	EBI
01C	S/U	RESERVED	PORT B DATA DIRECTION (DDRB)	EBI
01E	S	RESERVED	PORT B PIN ASSIGNMENT (PPARB)	EBI
020	S	SW INTERRUPT VECTOR (SWIV)	SYSTEM PROTECTION CONTROL (SYPCR)	SYSTEM PROTECTION
022	S	PERIODIC INTERRUPT CONTROL REGISTER (PICR)		SYSTEM PROTECTION
024	S	PERIODIC INTERRUPT TIMING REGISTER (PITR)		SYSTEM PROTECTION
026	S	RESERVED	SOFTWARE SERVICE (SWSR)	SYSTEM PROTECTION
040 042	S S	CS0 ADDRESS MASK REGISTER		CHIP SELECT
044 046	S S	CS0 BASE ADDRESS REGISTER		CHIP SELECT
048 04A	S S	CS1 ADDRESS MASK REGISTER		CHIP SELECT
04C 04E	S S	CS1 BASE ADDRESS REGISTER		CHIP SELECT
050 052	S S	CS2 ADDRESS MASK REGISTER		CHIP SELECT
054 056	S S	CS2 BASE ADDRESS REGISTER		CHIP SELECT
058 05A	S S	CS3 ADDRESS MASK REGISTER		CHIP SELECT
05C 05E	S S	CS3 BASE ADDRESS REGISTER		CHIP SELECT

Figure 4-8. SIM49 Programming Model

4.3.1 Module Base Address Register (MBAR)

MBAR 1 $0003FF00

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
BA31	BA30	BA29	BA28	BA27	BA26	BA25	BA24	BA23	BA22	BA21	BA20	BA19	IBA18	BA17	BA16
RESET:															
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
BA15	BA14	BA13	BA12	0	0	AS8	AS7	AS6	AS5	AS4	AS3	AS2	AS1	AS0	V
RESET															
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

CPU Space Only

BA31–BA12—Base Address Bits 31–12

The base address field is the upper 20 bits of the MBAR and provides for block starting addresses in increments of 4-Kbytes.

Bits 11, 10—Reserved by Motorola

AS8–AS0—Address Space Bits 8–0

The address space field allows particular address spaces to be masked, placing the 4-Kbyte module block into a particular address space(s). If an address space is masked, an access to the register block location in that address space becomes an external access. The module block is not accessed. The address space bits are as follows:

AS8—mask DMA Space	address space (FC3–FC0 = 1xxx)
AS7—mask CPU Space	address space (FC3–FC0 = 0111)
AS6—mask Supervisor Program	address space (FC3–FC0 = 0110)
AS5—mask Supervisor Data	address space (FC3–FC0 = 0101)
AS4—mask Reserved [Motorola]	address space (FC3–FC0 = 0100)
AS3—mask Reserved [User]	address space (FC3–FC0 = 0011)
AS2—mask User Program	address space (FC3–FC0 = 0010)
AS1—mask User Data	address space (FC3–FC0 = 0001)
AS0—mask Reserved [Motorola]	address space (FC3–FC0 = 0000)

For each address space bit:

1 = Mask this address space from the internal module selection. The bus cycle goes external.
0 = Decode for the internal module block.

V—Valid Bit

This bit indicates when the contents of the MBAR are valid. The base address value is not used; therefore, all internal module registers are not accessible until the V-bit is set.

1 = Contents are valid.
0 = Contents are not valid.

NOTE

An access to this register does not affect external space since the cycle is not run externally.

Example code to read the MBAR value into CPU register D0:

```
MOVE.L    #7,D0           load D0 with the CPU space function code
MOVEC.L   D0,SFC          load SFC to indicate CPU space
LEA.L     $0003FF00,A0    load A0 with the address of MBAR
MOVES.L   (A0),D0         load D0 with the contents of MBAR
```

Example code for initializing the MBAR register:

```
MOVE.L    #7,D0           load D0 with the CPU space function code
MOVEC.L   D0,DFC          load DFC to indicate CPU space
LEA.L     $0003FF00,A0    load A0 with the address of MBAR
MOVE.L    #$FFFFF101,D0   MBAR value: base = FFFFF000, AS7, valid
MOVES.L   D0,(A0)         write the value contained in D0 into MBAR
```

4.3.2 System Configuration and Protection Registers

The following paragraphs provide descriptions of the system configuration and protection registers.

4.3.2.1 MODULE CONFIGURATION REGISTER (MCR). The MCR, which controls the SIM49 configuration, can be read or written at any time.

MCR $000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	FRZ1	FRZ0	FIRQ	0	0	SHEN1	SHEN0	SUPV	0	0	0	IARB3	IARB2	IARB1	IARB0
RESET:															
0	1	1	0	0	0	0	0	1	0	0	0	1	1	1	1

Supervisor Only

Bits 15, 11, 10, 6–4—Reserved by Motorola

FRZ1—Freeze Software Enable

1 = When FREEZE is asserted, the software watchdog and periodic interrupt timer counters are disabled, preventing interrupts from occurring during software debug.

0 = When FREEZE is asserted, the software watchdog and periodic interrupt timer counters continue to run. See **4.2.7 Freeze** for more information.

FRZ0—Freeze Bus Monitor Enable

1 = When FREEZE is asserted, the bus monitor is disabled.
0 = When FREEZE is asserted, the bus monitor continues to operate as programmed.

FIRQ—Full Interrupt Request Mode

1 = Configures port B for seven interrupt request lines, autovector, and no external chip selects.
0 = Configures port B for four interrupt request lines and four external chip selects.

See Table 4-5 for pin function selection.

SHEN1, SHEN0—Show Cycle Enable

These two control bits determine what the EBI does with the external bus during internal transfer operations (see Table 4-7). A show cycle allows internal transfers to be externally monitored. The address, data, and control signals (except for $\overline{AS}$) are driven externally. $\overline{DS}$ is used to signal address strobe timing for show cycles. Data is valid on the next falling clock edge after $\overline{DS}$ is negated. However, data is not driven externally, and $\overline{AS}$ and $\overline{DS}$ are not asserted externally for internal accesses unless show cycles are enabled.

If external bus arbitration is disabled, the EBI will not recognize an external bus request until arbitration is enabled again. To prevent bus conflicts, external peripherals must not attempt to initiate cycles during show cycles with arbitration disabled.

Table 4-7. SHENx Control Bits

SHEN1	SHEN0	ACTION
0	0	Show cycles disabled, external arbitration enabled
0	1	Show cycles enabled, external arbitration disabled
1	X	Show cycles enabled, external arbitration enabled

SUPV—Supervisor/User Data Space

The SUPV bit defines the SIM49 registers as either supervisor data space or user (unrestricted) data space.

1 = The SIM49 registers defined as supervisor/user are restricted to supervisor data access (FC3–FC0 = $5). An attempted user-space write is ignored and returns $\overline{BERR}$.
0 = The SIM49 registers defined as supervisor/user data are unrestricted (FC2 is a don't care).

IARB3–IARB0—Interrupt Arbitration Bits 3–0

These bits are used to arbitrate for the bus in the case that two or more modules simultaneously generate an interrupt at the same priority level. No two modules can share the same IARB value. The reset value of IARB is $F, allowing the SIM49 to arbitrate during an IACK cycle immediately after reset. The system software should initialize the IARB field to a value from $F (highest priority) to $1 (lowest priority). A value of $0 prevents arbitration and causes all SIM49 interrupts, including external interrupts, to be discarded as extraneous.

4.3.2.2 IDENTIFICATION REGISTER (IDR). The IDR differentiates the M68300 family parts. When read, a unique value is returned for the MC68340 and MC68349. The IDR in the MC68349 returns $31; the IDR in the MC68340 (also located at $002 offset from the MBAR) returns all zeroes.

IDR $002

7	6	5	4	3	2	1	0
0	0	1	1	0	0	0	1
RESET:							
0	0	1	1	0	0	0	1

Supervisor Only

4.3.2.3 AUTOVECTOR REGISTER (AVR). The AVR contains bits that correspond to external interrupt levels that require an autovector response. Setting a bit allows the SIM49 to assert an internal $\overline{\text{AVEC}}$ during the IACK cycle in response to the specified interrupt request level. This register can be read and written at any time.

AVR $006

7	6	5	4	3	2	1	0
AV7	AV6	AV5	AV4	AV3	AV2	AV1	0
RESET:							
0	0	0	0	0	0	0	0

Supervisor Only

NOTE:

The IARB field in the MCR must contain a value other than $0 for the SIM49 to autovector for external interrupts.

4.3.2.4 RESET STATUS REGISTER (RSR). The RSR contains a bit for each reset source to the SIM49. A set bit indicates the last type of reset that occurred, and only one bit can be set in the register. The RSR is updated by the reset control logic when the SIM49 comes out of reset. This register can be read at any time; a write has no effect. For more information, see **Section 3 Bus Operation.**

RSR $007

7	6	5	4	3	2	1	0
EXT	POW	SW	DBF	0	LOC	SYS	0

Supervisor Only

EXT—External Reset

1 = The last reset was caused by an external signal driving $\overline{\text{RESET}}$.
0 = The last reset was not caused by an external signal driving $\overline{\text{RESET}}$.

POW—Power-Up Reset

1 = The last reset was caused by the power-up reset circuit.
0 = The last reset was not caused by the power-up reset circuit.

SW—Software Watchdog Reset

1 = The last reset was caused by the software watchdog circuit.
0 = The last reset was not caused by the software watchdog circuit.

DBF—Double Bus Fault Monitor Reset

1 = The last reset was caused by the double bus fault monitor.
0 = The last reset was not caused by the double bus fault monitor.

Bits 3, 0—Reserved by Motorola

LOC—Loss of Clock Reset

1 = The last reset was caused by a loss of frequency reference to the clock synthesizer. This reset can only occur if the RSTEN bit in the SYNCR is set and the VCO is enabled.
0 = The last reset was not caused by a loss of frequency reference to the clock synthesizer.

SYS—System Reset

1 = The last reset was caused by the CPU32+ executing a RESET instruction. The system reset does not load a reset vector or affect any internal CPU32+ registers, SIM49 configuration registers, or the MCR in each internal peripheral module (DMA, quad data memory, and serial modules). It will, however, reset external devices and all other registers in the peripheral modules.
0 = The last reset was not caused by the CPU32+ executing a RESET instruction.

4.3.2.5 SOFTWARE INTERRUPT VECTOR REGISTER (SWIV). The SWIV contains the 8-bit vector that is returned by the SIM49 during an IACK cycle in response to an interrupt generated by the software watchdog. This register can be read or written at any time. This register is set to the uninitialized vector, $0F, at reset.

SWIV $020

7	6	5	4	3	2	1	0
SWIV7	SWIV6	SWIV5	SWIV4	SWIV3	SWIV2	SWIV1	SWIV0
RESET:							
0	0	0	0	1	1	1	1

Supervisor Only

4.3.2.6 SYSTEM PROTECTION CONTROL REGISTER (SYPCR). The SYPCR controls the system monitors, the prescaler for the software watchdog, and the bus monitor timing. This register can be read at any time, but can be written only once after reset.

SYPCR $021

7	6	5	4	3	2	1	0
SWE	SWRI	SWT1	SWT0	DBFE	BME	BMT1	BMT0
RESET:							
0	0	0	0	0	0	0	0

Supervisor Only

SWE—Software Watchdog Enable

1 = Software watchdog is enabled.
0 = Software watchdog is disabled.

See **4.2.2.5 Software Watchdog** for more information.

SWRI—Software Watchdog Reset/Interrupt Select

1 = Software watchdog causes a system reset.
0 = Software watchdog causes a level 7 interrupt to the CPU32+.

SWT1, SWT0—Software Watchdog Timing

These bits, along with the SWP bit in the PITR, control the divide ratio used to establish the timeout period for the software watchdog. The software watchdog timeout period is given by the following formula:

$$\frac{\text{divide count}}{\text{EXTAL frequency}}$$

The software watchdog timeout period, listed in Table 4-8, gives the formula to derive the software watchdog timeout for any clock frequency. The timeout periods are listed for a 32.768-kHz crystal used with the VCO and for a 16.777-MHz external oscillator.

Table 4-8. Deriving Software Watchdog Timeout

SWP	SWT1	SWT0	Software Timeout Period	32.768-kHz Crystal Period	16.777-MHz External Clock Period
0	0	0	2^9/EXTAL Input Frequency	15.6 ms	30 µs
0	0	1	2^{11}/EXTAL Input Frequency	62.5 ms	122 µs
0	1	0	2^{13}/EXTAL Input Frequency	250 ms	488 µs
0	1	1	2^{15}/EXTAL Input Frequency	1 s	1.95 ms
1	0	0	2^{18}/EXTAL Input Frequency	8 s	15.6 ms
1	0	1	2^{20}/EXTAL Input Frequency	32 s	62.5 ms
1	1	0	2^{22}/EXTAL Input Frequency	128 s	250 ms
1	1	1	2^{24}/EXTAL Input Frequency	512 s	1 s

NOTE: When the SWP and SWT bits are modified to select a software timeout other than the default, the software service sequence ($55 followed by $AA written to the software service register) must be performed before the new timeout period takes effect. Refer to **4.2.2.5 Software Watchdog** for more information.

DBFE—Double Bus Fault Monitor Enable

1 = Enable double bus fault monitor function.
0 = Disable double bus fault monitor function.

For more information, see **4.2.2.3 Double Bus Fault Monitor** and **Section 5 CPU030**.

BME—Bus Monitor External Enable

1 = Enable bus monitor function for an internal-to-external bus cycle.
0 = Disable bus monitor function for an internal-to-external bus cycle.

For more information see **4.2.2.2 Internal Bus Monitor**.

BMT1, BMT0—Bus Monitor Timing

These bits select the timeout period for the bus monitor (see Table 4-9). Upon reset, the bus monitor is set to 512 system clocks.

Table 4-9. BMTx Encoding

BMT1	BMT0	Bus Monitor Timeout Period
0	0	512 system clocks (CLKOUT)
0	1	256 system clocks
1	0	128 system clocks
1	1	64 system clocks

4.3.2.7 PERIODIC INTERRUPT CONTROL REGISTER (PICR). The PICR contains the interrupt level and the vector number for the periodic interrupt request. This register can be read or written at any time.

PICR $022

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	PIRQL2	PIRQL1	PIRQL0	PIV7	PIV6	PIV5	PIV4	PIV3	PIV2	PIV1	PIV0
RESET:															
0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1

Supervisor Only

Bits 15–11—Reserved by Motorola

PIRQL2–PIRQL0—Periodic Interrupt Request Level

These bits contain the periodic interrupt request level. Table 4-10 lists which interrupt request level is asserted during an IACK cycle when a periodic interrupt is generated. The periodic timer continues to run when the interrupt is disabled.

Table 4-10. PIRQL Encoding

PIRQL2	PIRQL1	PIRQL0	Interrupt Request Level
0	0	0	Periodic Interrupt Disabled
0	0	1	Interrupt Request Level 1
0	1	0	Interrupt Request Level 2
0	1	1	Interrupt Request Level 3
1	0	0	Interrupt Request Level 4
1	0	1	Interrupt Request Level 5
1	1	0	Interrupt Request Level 6
1	1	1	Interrupt Request Level 7

NOTE

Use caution with a level 7 interrupt encoding due to the SIM49's interrupt servicing order. See **4.2.2.7 Simultaneous Interrupts by Sources in the SIM49** for the servicing order.

PIV7–PIV0—Periodic Interrupt Vector Bits 7–0

These bits contain the value of the vector generated during an IACK cycle in response to an interrupt from the periodic timer. When the SIM49 responds to the IACK cycle, the periodic interrupt vector from the PICR is placed on the bus. This vector number is multiplied by four to form the vector offset, which is added to the vector base register to obtain the address of the vector.

4.3.2.8 PERIODIC INTERRUPT TIMER REGISTER (PITR). The PITR contains control for prescaling the software watchdog and periodic timer as well as the count value for the periodic timer. This register can be read or written at any time.

PITR $024

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	SWP	PTP	PITR7	PITR6	PITR5	PITR4	PITR3	PITR2	PITR1	PITR0
RESET:															
0	0	0	0	0	0	$\overline{\text{MODCK}}$	$\overline{\text{MODCK}}$	0	0	0	0	0	0	0	0

Supervisor Only

Bits 15–10—Reserved by Motorola

SWP—Software Watchdog Prescale

This bit controls the software watchdog clock source as shown in **4.3.2.6 System Protection Control Register (SYPCR)**.

1 = Software watchdog clock is prescaled by a value of 512.
0 = Software watchdog clock is not prescaled.

The SWP reset value is the inverse of the MODCK signal state on the rising edge of reset.

PTP—Periodic Timer Prescaler Control

This bit contains the prescaler control for the periodic timer.

1 = Periodic timer clock is prescaled by a value of 512.
0 = Periodic timer clock is not prescaled.

The PTP reset value is the inverse of the MODCK signal state on the rising edge of reset.

PITR7–PITR0—Periodic Interrupt Timer Register Bits 7–0

The remaining bits of the PITR contain the count value for the periodic timer. A zero value turns off the periodic timer.

4.3.2.9 SOFTWARE SERVICE REGISTER (SWSR). The SWSR is the location to which the software watchdog servicing sequence is written. The software watchdog can be enabled or disabled by the SWE bit in the SYPCR. The SWSR can be written at any time, but returns all zeros when read.

SWSR $027

7	6	5	4	3	2	1	0
SWSR7	SWSR6	SWSR5	SWSR4	SWSR3	SWSR2	SWSR1	SWSR0
RESET:							
0	0	0	0	0	0	0	0

Supervisor Only

4.3.3 Clock Synthesizer Control Register (SYNCR)

The SYNCR can be read or written only in supervisor mode. The reset state of SYNCR produces an operating frequency of 8.39 MHz when the PLL is referenced to a 32.768-kHz reference signal. The system frequency is controlled by the frequency control bits in the upper byte of the SYNCR as follows:

$$F_{SYSTEM} = F_{CRYSTAL}\,[2^{(2W+X+3Z-1)}] \times (Y+1)$$

SYNCR $004

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
W	X	Y5	Y4	Y3	Y2	Y1	Y0	Z	0	0	SLIMP	SLOCK	RSTEN	STSIM	STEXT
RESET:															
0	0	1	1	1	1	1	1	1	0	0	U	U	0	0	0

U = Unaffected by reset Supervisor Only

W—Frequency Control Bit

This bit controls the prescaler tap in the synthesizer feedback loop. Setting the bit increases the VCO speed by a factor of 4, requiring a time delay for the VCO to relock (see equation for determining system frequency).

X—Frequency Control Bit

This bit controls a divide-by-two prescaler, which is outside the synthesizer feedback loop. Setting the bit doubles the system clock speed without changing the VCO speed, as specified in the equation for determining system frequency; therefore, no delay is incurred to relock the VCO.

Y5–Y0—Frequency Control Bits

The Y-bits, with a value from 0–63, control the modulus downcounter in the synthesizer feedback loop, causing it to divide by the value of Y+1 (see the equation for determining system frequency). Changing these bits requires a time delay for the VCO to relock.

Z—Frequency Control Bit

This bit controls a divide-by-eight prescaler, which is outside the synthesizer feedback loop. Clearing the bit decreases the system clock speed by a factor of eight without changing the VCO speed, as specified in the equation for determining system frequency; therefore, no delay is incurred to relock the VCO.

Bits 6–5—Reserved by Motorola

SLIMP—Limp Mode

1 = A loss of input signal reference has been detected, and the VCO is running at approximately one-half the maximum speed (affected by the X-bit), determined from an internal voltage reference.
0 = External input signal frequency is at VCO reference.

SLOCK—Synthesizer Lock

1 = VCO has locked onto the desired frequency (or system clock is driven externally).
0 = VCO is enabled, but has not yet locked.

RSTEN—Reset Enable

1 = Loss of input signal causes a system reset.
0 = Loss of input signal causes the VCO to operate at a nominal speed without external reference (limp mode), and the device continues to operate at that speed.

STSIM—Stop Mode System Integration Clock

1 = When LPSTOP is executed, the SIM49 clock is driven from the VCO.
0 = When LPSTOP is executed, the SIM49 clock is driven from an external crystal or oscillator, and the VCO is turned off to conserve power.

STEXT—Stop Mode External Clock

1 = When the LPSTOP instruction is executed, the external clock pin (CLKOUT) is driven from the SIM49 clock as determined by the STSIM bit.
0 = When the LPSTOP instruction is executed, the external clock (CLKOUT) is held low to conserve power. No external clock will be driven in LPSTOP mode.

4.3.4 Chip Select Registers

The following paragraphs provide descriptions of the registers in the chip select function, and an example of how to program the registers. The chip select registers cannot be accessed until the register is initialized.

4.3.4.1 BASE ADDRESS REGISTERS. There are four 32-bit base address registers in the chip select function, one for each chip select signal.

Base Address $044 (CS0), $04C (CS1), $054 (CS2), $05C (CS3)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
BA31	BA30	BA29	BA28	BA27	BA26	BA25	BA24	BA23	BA22	BA21	BA20	BA19	BA18	BA17	BA16
RESET:															
U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
BA15	BA14	BA13	BA12	BA11	BA10	BA9	BA8	BFC3	BFC2	BFC1	BFC0	WP	EDS	NCS	V
RESET:															
U	U	U	U	U	U	U	U	U	U	U	U	U	U	0	0

U = Unaffected by reset Supervisor Only

BA31–BA8—Base Address Bits 31–8

The base address field, the upper 24 bits of each base address register, selects the starting address for the chip select. The specified base address must be on a multiple of the selected block size. The corresponding bits, AM31–AM8, in the address mask

register define the size of the block for the chip select. The base address field (and the base function code field) is compared to the address on the address bus to determine if a chip select should be generated.

BFC3–BFC0—Base Function Code Bits 3–0

The value programmed into this field causes a chip select to be asserted for a certain address space type. There are nine function code address spaces (see **Section 3 Bus Operation**) specified as either user or supervisor, program or data, CPU, and DMA. These bits should be used to allow access to one type of address space. If access to more than one type of address space is desired, the FCMx bits should be used in addition to the BFCx bits. To prevent access to CPU space, set the NCS bit.

WP—Write Protect

This bit can restrict write accesses to the address range in a base address register. An attempt to write to the range of addresses specified in a base address register that has this bit set returns $\overline{\text{BERR}}$.

1 = Only read accesses are allowed.
0 = Either read or write accesses are allowed.

EDS—Extended Delay Select

This bit is used along with the DDx bits of the corresponding address mask register to specify the number (zero to six) of wait sates or a fast termination (two-clock external access). See table 4-11 for the encodings of the EDS bit and DDx bits.

NCS—No CPU Space

This bit specifies whether or not a chip select will assert on a CPU space access cycle (FC3–FC0 = $7 or $F). If both supervisor data and program accesses are desired, while ignoring CPU space accesses, then this bit should be set.

1 = Suppress the chip select on a CPU space access.
0 = Assert the chip select on a CPU space access.

V—Valid Bit

This bit indicates that the contents of its base address register and address mask register pair are valid. The programmed chip selects do not assert until the V-bit is set. A reset clears the V-bit in each base address register, but does not change any other bits in the base address and address mask registers.

1 = Contents are valid.
0 = Contents are not valid.

4.3.4.2 ADDRESS MASK REGISTERS. There are four 32-bit address mask registers in the chip select function, one for each chip select signal.

Address Mask $040 (CS0), $048 (CS1), $050 (CS2), $058 (CS3)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
AM31	AM30	AM29	AM28	AM27	AM26	AM25	AM24	AM23	AM22	AM21	AM20	AM19	AM18	AM17	AM16
RESET:															
U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
AM15	AM14	AM13	AM12	AM11	AM10	AM9	AM8	FCM3	FCM2	FCM1	FCM0	DD1	DD0	PS1	PS0
RESET:															
U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U

U = Unaffected by reset Supervisor Only

AM31–AM8—Address Mask Bits 31–8

The address mask field, the upper 24 bits of each address mask register, defines the chip select block size. The block size is equal to 2^n, where n = (number of bits set in the address mask field) + 8.

Any set bit masks the corresponding base address register bit (the base address register bit becomes a don't care). By masking the address bits independently, external devices of different size address ranges can be used. Address mask bits can be set or cleared in any order in the field, allowing a resource to reside in more than one area of the address map. This field can be read or written at any time.

FCM3–FCM0—Function Code Mask Bits 3–0

This field can be used to mask certain function code bits, allowing more than one address space type to be assigned to a chip select. Any set bit masks the corresponding function code bit.

DD1, DD0—DSACK Delay Bits 1 and 0

This field, along with the EDS bit of the corresponding base address register, determines the number of wait states added before an internal $\overline{\text{DSACKx}}$ is returned for that entry. Table 4-11 lists the encoding for the EDS and DDx bits.

NOTE

The port size field must be programmed for an internal $\overline{\text{DSACKx}}$ response and the EDS and DDx bits to have significance. If external $\overline{\text{DSACKx}}$ signals are returned earlier than indicated by the EDS and DDx bits, the cycle will terminate sooner than programmed. See **4.2.5.2 PORT B** for a discussion on using the internal $\overline{\text{DSACKx}}$ generation without using the $\overline{\text{CSx}}$ signal.

Table 4-11. EDS and DDx Encoding

EDS	DD1	DD0	Response
0	0	0	Zero Wait State
0	0	1	One Wait State
0	1	0	Two Wait States
0	1	1	Three Wait States
1	0	0	Four Wait States
1	0	1	Five Wait States
1	1	0	Six Wait States
1	1	1	Fast Termination

PS1, PS0—Port Size Bits 1 and 0

This field determines whether a given chip select responds with $\overline{\text{DSACKx}}$ and, if so, what port size is returned. Table 4-12 lists the encoding for the PSx bits.

Table 4-12. PSx Encoding

PS1	PS0	Mode
0	0	32-Bit Port
0	1	16-Bit Port
1	0	8-Bit Port
1	1	External $\overline{\text{DSACKx}}$ Response

To use the external $\overline{\text{DSACKx}}$ response, PS1–PS0 = 11 should be selected to suppress internal $\overline{\text{DSACKx}}$ generation . The EDS bit in the corresponding base address register and the DDx bits then have no significance.

4.3.4.3 CHIP SELECT REGISTERS PROGRAMMING EXAMPLE. The following listing is an example of programming a chip select at starting address $00040000, for a block size of 256 Kbytes, accessing supervisor and user data spaces with a 16-bit port requiring two wait states. There will be no write protection, no fast termination, and no CPU space accesses.

base address = $00040013

address mask = $0003FF49

NOTE

If an access matches multiple chip selects, the lowest numbered chip select will have priority. For example, if $\overline{\text{CS0}}$ and $\overline{\text{CS2}}$ "overlap" for a certain range, $\overline{\text{CS0}}$ will assert when accessing the "overlapped" address range, and $\overline{\text{CS2}}$ will not.

4.3.5 External Bus Interface Control

The following paragraphs describe the registers that control the I/O pins used with the EBI. Refer to **Section 3 Bus Operation** for more information about the EBI. For a list of pin numbers used with port A and port B, see the pinout diagram in **Section 12 Ordering Information and Mechanical Data. Section 2 Signal Descriptions** shows a block diagram of the port control circuits.

4.3.5.1 PORT A PIN ASSIGNMENT REGISTER 1 (PPARA1). PPARA1 selects between an address and discrete I/O function for the port A pins. Any set bit defines the corresponding pin to be an I/O pin, controlled by the port A data and data direction registers. Any cleared bit defines the corresponding pin to be an address bit as defined in Table 4-5. The reset value of the PPARA1 bits is the inverse of the state of D29 at reset. Bits set in this register override the configuration setting of PPARA2. This register can be read or written at any time.

PPARA1 $015

7	6	5	4	3	2	1	0
PRTA7/ A31	PRTA6/ A30	PRTA5/ A29	PRTA4/ A28	PRTA3/ A27	PRTA2/ A26	PRTA1/ A25	PRTA0/ A24
RESET: $\overline{D29}$	$\overline{D29}$	$\overline{D29}$	$\overline{D29}$	$\overline{D29}$	$\overline{D29}$	$\overline{D29}$	$\overline{D29}$

Supervisor Only

4.3.5.2 PORT A PIN ASSIGNMENT REGISTER 2 (PPARA2). PPARA2 selects between an address and $\overline{IACKx}$ function for the port A pins. Any set bit defines the corresponding pin to be an $\overline{IACKx}$ output pin (as long as the corresponding PPARA1 bit is cleared). Any cleared bit defines the corresponding pin to be an address bit (as long as the corresponding PPARA1 bit is cleared) as defined in Table 4-5. Any set bits in PPARA1 override the configuration set in PPARA2. Since there is no level 0 interrupt, bit 0 has no function in this register and is reserved by Motorola. This register can be read or written at any time.

PPARA2 $017

7	6	5	4	3	2	1	0
IACK7/ A31	IACK6/ A30	IACK5/ A29	IACK4/ A28	IACK3/ A27	IACK2/ A26	IACK1/ A25	0
RESET: 0	0	0	0	0	0	0	0

Supervisor Only

The $\overline{IACKx}$ signals are asserted if a bit in PPARA2 is set and the CPU32+ services an external interrupt at the corresponding level. $\overline{IACKx}$ signals have the same timing as address strobes.

4.3.5.3 PORT A DATA DIRECTION REGISTER (DDRA). DDRA controls the direction of the pin drivers when the pins are configured as I/O. Any set bit configures the corresponding pin as an output. Any cleared bit configures the corresponding pin as an input. This register affects only pins configured as discrete I/O. This register can be read or written at any time.

DDRA $013

7	6	5	4	3	2	1	0
DD7	DD6	DD5	DD4	DD3	DD2	DD1	DD0
RESET:							
0	0	0	0	0	0	0	0

Supervisor/User

4.3.5.4 PORT A DATA REGISTER (PORTA). PORTA affects only pins configured as discrete I/O. A write to PORTA is stored in the internal data latch, and if any port A pin is configured as an output, the value stored for that bit is driven on the pin. A read of PORTA returns the value at the pin only if the pin is configured as discrete input. Otherwise, the value read is the value stored in the internal data latch. This register can be read or written at any time.

PORTA $011

7	6	5	4	3	2	1	0
P7	P6	P5	P4	P3	P2	P1	P0
RESET:							
U	U	U	U	U	U	U	U

U = Unaffected by reset Supervisor/User

4.3.5.5 PORT B PIN ASSIGNMENT REGISTER (PPARB). PPARB controls the function of each port B pin. Any set bit defines the corresponding pin to be an $\overline{\text{IRQx}}$ input or $\overline{\text{CSx}}$ as defined in Table 4-6. Any cleared bit defines the corresponding pin to be a discrete I/O pin (or $\overline{\text{CSx}}$ if the FIRQ bit of the MCR is zero) controlled by the port B data and data direction registers. The MODCK signal has no function after reset. PPARB is configured to all zeros at reset (along with the FIRQ bit of the MCR) to provide for MODCK, PPARB7, PPARB6, PPARB5, PPARB3, and $\overline{\text{CS3}}$–$\overline{\text{CS0}}$. This register can be read or written at any time.

PPARB $01F

7	6	5	4	3	2	1	0
PPARB7/ IRQ7	PPARB6/ IRQ6	PPARB5/ IRQ5	PPARB4/ IRQ4	PPARB3/ IRQ3	PPARB2/ IRQ2	PPARB1/ IRQ1	PPARB0
RESET:							
0	0	0	0	0	0	0	0

Supervisor Only

4.3.5.6 PORT B DATA DIRECTION REGISTER (DDRB). DDRB controls the direction of the pin drivers when the pins are configured as I/O. Any set bit configures the corresponding pin as an output; any cleared bit configures the corresponding pin as an input. This register affects only pins configured as discrete I/O. This register can be read or written at any time.

DDRB $01D

7	6	5	4	3	2	1	0
DD7	DD6	DD5	DD4	DD3	DD2	DD1	DD0
RESET:							
0	0	0	0	0	0	0	0

Supervisor/User

4.3.5.7 PORT B DATA REGISTER (PORTB, PORTB1). This is a single register that can be accessed at two different addresses. This register affects only those pins configured as discrete I/O. A write is stored in the internal data latch, and if any port B pin is configured as an output, the value stored for that bit is driven on the pin. A read of this register returns the value stored in the register only if the pin is configured as a discrete output. Otherwise, the value read is the value of the pin. This register can be read or written at any time.

PORTB, PORTB1 $019, 01B

7	6	5	4	3	2	1	0
P7	P6	P5	P4	P3	P2	P1	P0
RESET:							
U	U	U	U	U	U	U	U

Supervisor/User

4.4 MC68349 INITIALIZATION SEQUENCE

The following paragraphs discuss a suggested method for initializing the MC68349 after power-up.

4.4.1 Startup

$\overline{\text{RESET}}$ is asserted by the MC68349 during the time in which V_{CC} is ramping up, the VCO is locking onto the frequency, and the MC68349 is going through the reset operation. After $\overline{\text{RESET}}$ is negated, four bus cycles are run, with global $\overline{\text{CS0}}$ being asserted to fetch the 32-bit supervisor stack pointer (SSP) and the 32-bit program counter (PC) from the boot read-only memory (ROM). Until programmed differently, $\overline{\text{CS0}}$ is a global, six-wait-state chip select. Port termination size is determined at reset by the state of D31 and D30 (see Table 4-4). $\overline{\text{CS0}}$ can be programmed to continue decode for a range of addresses after the V-bit is set, provided the desired address range is first loaded into the $\overline{\text{CS0}}$ base address register. After the V-bit is set for $\overline{\text{CS0}}$, global chip select can only be restarted with a system reset.

After the SSP and the PC are fetched, the MBAR should be initialized, and the MBAR V-bit should be set (CPU space address $0003FF00) with the desired base address for the internal modules.

4.4.2 SIM49 Module Configuration

The order of the following SIM49 register initializations is not important; however, time can be saved by initializing the SYNCR first to quickly increase to the desired processor operating frequency. The MBAR must be initialized prior to any of the following steps.

Hardware Configuration

- D31 and D30 select 8-/16-/32-bit or external termination for CS0 global chip select
- D29 configures port A as either address outputs or an input port

Clock Synthesizer Control Register (SYNCR):

- Set frequency control bits (W, X, Y, Z) to specify frequency.
- Select action taken during loss of crystal (RSTEN bit): activate a system reset or operate in limp mode.
- Select system clock and CLKOUT during LPSTOP (STSIM and STEXT bits).

Module Configuration Register (MCR)

- If using the software watchdog, periodic interrupt timer, and/or the bus monitor, select action taken when FREEZE is asserted (FRZx bits).
- Select port B configuration (FIRQ bit). Note that this bit is used in combination with the bits in the PPARB to program the function of the port B pins.
- Select the access privilege for the supervisor/user registers (SUPV bit).
- Select the interrupt arbitration level for the SIM49 (IARBx bits).

Autovector Register (AVR)

- Select the desired external interrupt levels for internal autovectoring.

System Protection Control Register (SYPCR) (Note that this register can only be written once after reset.)

- Enable the software watchdog, if desired (SWE bit).
- If the watchdog is enabled, select whether a system reset or a level 7 interrupt is desired at timeout (SWRI bit).
- If the watchdog is enabled, select the timeout period (SWTx bits).
- Enable the double bus fault monitor, if desired (DBFE bit).
- Enable the external bus monitor, if desired (BME bit).
- Select timeout period for bus monitor (BMTx bits).

Software Watchdog Interrupt Vector Register (SWIV)

- If using the software watchdog, program the vector number for a software watchdog interrupt.

Periodic Interrupt Timer Register (PITR)

- If using the software watchdog, select whether or not to prescale (SWP bit).
- If using the periodic interrupt timer, select whether or not to prescale (PTP bit).
- Program the count value for the periodic timer, or program a zero value to turn off the periodic timer (PITRx bits).

Periodic Interrupt Control Register (PICR)

- If using the periodic timer, program the desired interrupt level for the periodic interrupt timer (PIRQLx bits).
- If using the periodic timer, program the vector number for a periodic timer interrupt.

Chip Select Base Address and Address Mask Registers

- Initialize and set the V-bits in the necessary chip select base address and address mask registers. Following this step, other system resources requiring the $\overline{CSx}$ signals can be accessed. Care must be exercised when changing the address for $\overline{CS0}$.

Port A and B Registers

- Program the desired function of the port A signals (PPARA1 and PPARA2 registers).
- Program the desired function of the port B signals (PPARB register).

4.4.3 SIM49 Example Configuration Code

The following code is an example configuration sequence for the SIM49 module.

```
*******************************************************************************
* MC68349 basic SIM49 register initialization example code:
* This code is used to initialize the MC68349's internal SIM49 registers,
* providing basic functions for operation.
* It includes chip select programming for external devices.
* This code would be programmed beginning at offset $0 into ROM which is
* relocated to address $60000 by the initialization code.
* The SSP_VEC and RST_VEC vectors used to initialize the system stack
* pointer and initial PC, respectively, are located at offset $0 after
* reset.
*******************************************************************************
* equates
*******************************************************************************
SSP_INIT        EQU     $10000          Stack pointer initial value - top of RAM
MBAR   EQU      $0003FF00       Address of Module Base Address Reg.
MODBASE         EQU     $FFFFF000       Default Module Base address value

****************************************
* SIM49 register offsets from MBAR base address
MCR             EQU     $00
SYNCR           EQU     $04
SYPCR           EQU     $21
CSAM0           EQU     $40
CSBAR0          EQU     $44
CSAM1           EQU     $48
CSBAR1          EQU     $4c
CSAM2           EQU     $50
CSBAR2          EQU     $54
CSAM3           EQU     $58
CSBAR3          EQU     $5c
*******************************************************************************
* Reset vectors
* These two vectors should be located at addresses $0 and $4 after a processor
* hardware reset.
*******************************************************************************
          ORG     $60000
SSP_VEC         DC.L    SSP_INIT                Supervisor stack pointer - initial value
RST_VEC         DC.L    INIT340                 Reset vector pointing to initialization code

*******************************************************************************
* Initialization code
*******************************************************************************

* Start Chip Select Initialization:
INIT340         MOVE.W          #$2700,SR               Init SR - interrupts masked

* AS7 is set to prevent module blocked (mapped at $FFFFFxxx) from responding to
* an interrupt acknowledge access ($FFFFFFFx)—these accesses ate handled by
* dedicated IACK logic in the SIM49
```

```
*******************************************************************************
* Set up default module base address value
        MOVEQ.L     #7,D0                   MBAR is in CPU space
        MOVEC.L     D0,DFC                      load DFC to indicate CPU space
        MOVE.L      #MODBASE+$101,D0            Set address, AS7, and valid bit
        MOVES.L     D0,MBAR                     write to MBAR

*******************************************************************************
* Set up system protection register:
* Software watchdog disabled, double bus fault monitor disabled, bus
* monitor BERR after 64 clocks.
        MOVE.B      #7,SYPCR+MODBASE

*******************************************************************************
* Clock synthesizer control register:
* Switch from 8.3 to 16.7 MHZ
        MOVE.W      #$7F80,SYNCR+MODBASE    X>1 doubles the default speed

*******************************************************************************
* Module configuration register:
* When FREEZE is asserted, software watchdog and periodic interrupt timer
* are disabled, bus monitor is enabled.  Port B = 4 IRQs, 4 chip selects.
* Show Cycles enabled, external arbitration enabled. Supervisor/user
* SIM registers unrestricted, Interrupt Arbitration at priority $F
        MOVE.W      #$420F,MCR+MODBASE

*******************************************************************************
* Now, set up Address masks and base addresses for the chip selects:
        LEA         CSAM0+MODBASE,A0    Point to CS0 addr. mask location.
        MOVEQ       #7,D        Set up a loop counter.
        LEA         CSAM0$,A1       Point to addr mask memory location.
LOOP    MOVE.L      (A1)+,(A0)+     Init. addr mask and base addr reg
        DBRA        D0,LOOP
        JMP.L       USER_INIT

*******************************************************************************
*  Data table for chip select initialization
*******************************************************************************

* CS0 - EPROM - 00060000-0007ffff, 3-wait states, 16-bit, NCS, write protect
CSAM0$      DC.L    $0001FFFD
CSBAR0$     DC.L    $00060008
* CS1 - RAM - 00000000-0000ffff, 32-bit fast termination, NCS
CSAM1$      DC.L    $0000FFFC
CSBAR1$     DC.L    $00000007
* CS2 - external device - 00FFE8xx, external termination, NCS
CSAM2$      DC.L    $000000F3
CSBAR2$     DC.L    $00FFE803
* CS3 - secondary memory - 00000000-0003ffff, 1-wait states, 8-bit term, NCS
CSAM3$      DC.L    $0003FFF6
CSBAR3$     DC.L    $00000003

*******************************************************************************
        END
```

SECTION 5
CPU030

The MC68349 is based on the CPU030 processor, which combines a full 32-bit central processor (CPU32+) with a configurable instruction cache (CIC) and a quad data memory module (QDMM). This section discusses the CPU32+ and the CIC. The QDMM is discussed in **Section 6 Quad Data Memory Module**.

The CPU32+, the second instruction processing module of the M68300 family, is based on the industry-standard MC68000 core processor. Like the original CPU32, it has many features of the MC68010 and MC68020 as well as unique features suited for high-performance processor applications. The CPU32+ provides a significant performance increase over the MC68000 CPU, yet maintains source-code and binary-code compatibility with the M68000 family. The CPU32+ differs from the original CPU32 in two ways: it allows an option of a 32-bit data bus interface and allows byte-misaligned accesses to data operands.

The CIC improves system performance by reducing average memory access times for all instruction accesses (cache mode) and by providing fast static random access memory (SRAM) storage for critical data structures or code sequences (SRAM mode). The CPU32+ is connected directly to the CIC instead of through the intermodule bus (IMB); this direct connection reduces the impact of direct memory access (DMA) operations on system performance by allowing CPU32+ accesses to the CIC and DMA transfers on the IMB to occur simultaneously.

5.1 CPU32+ OVERVIEW

The CPU32+ is designed to interface to the intermodule bus, allowing interaction with other IMB submodules. In this manner, integrated processors can be developed that contain useful peripherals on chip. This integration provides high-speed accesses among the IMB submodules, increasing system performance.

The CPU32+ core is a CPU32 core with its bus interface unit modified to connect directly to the 32-bit IMB and take advantage of the larger bus width. Although the original CPU32 core already had a 32-bit internal data path and 32-bit arithmetic hardware, its external interface (i.e., to the internal IMB) was 16 bits. The CPU32+ core, however, can operate on 32-bit external operands with one bus cycle. This capability allows the CPU32+ core to fetch a long-word instruction or two word-length instructions in one bus cycle, allowing the internal instruction queue to be filled more quickly. The CPU32+ core can also read and write 32-bits of data in one bus cycle. The CPU32+ has an additional word in its

instruction pipeline when fetching from a 32-bit port. When fetching from a 16-bit port, this additional word is disabled. The performance of the CPU32+ on a 16-bit bus is the same as the CPU32 performance.

The CPU32+ supports byte-misaligned operands. Since operands can reside at any byte boundary, they may occasionally become misaligned. A byte operand is properly aligned at any address; a word operand is misaligned at a odd address; a long-word operand is misaligned at an address that is not evenly divisible by four. Devices such as the MC68302, MC68000/8, MC68010, and CPU32-based M68300 family processors allow long-word operand transfers at odd-word addresses, but force exceptions if word or long-word operand transfers are attempted at odd-byte addresses. Although the CPU32+ does not enforce any alignment restrictions for data operands (including program counter (PC) relative data addresses), some performance degradation occurs when additional bus cycles are required for long-word or word operands that are misaligned. For maximum performance, data items should be aligned on their natural boundaries. All instruction words and extension words must reside on word boundaries. Attempting to prefetch an instruction word at an odd address causes an address error exception.

The CPU32+ has four bits (SIZ1,SIZ0 and SZC1,SZC0) in the software status word (SSW) that are new or have changed definitions due to the 32-bit bus width and elimination of alignment restrictions.

The CPU32+ offers low power consumption. The CPU32+ is implemented in high-density complementary metal-oxide semiconductor (HCMOS) technology, providing low power use during normal operation. During periods of inactivity, the low-power stop (LPSTOP) instruction can be executed, shutting down the CPU32+ and other IMB modules, greatly reducing power consumption.

Ease of programming is an important consideration when using an integrated processor. The CPU32+ instruction format reflects a predominant register-memory interaction philosophy. All data resources are available to operations that require them. The programming model includes eight multifunction data registers and seven general-purpose addressing registers. The data registers support 8-bit (byte), 16-bit (word), and 32-bit (long-word) operand lengths for all operations. Address manipulation is supported by word and long-word operations. Although the PC and stack pointers (SP) are special-purpose registers, they are also available for most data addressing activities. Ease of program checking and diagnosis is enhanced by trace and trap capabilities at the instruction level.

As processor applications become more complex and programs become larger, high-level languages (HLLs) become the system designer's choice in programming languages. HLLs aid in the rapid development of complex algorithms with less error and are readily portable. The CPU32+ instruction set efficiently supports HLLs.

5.1.1 CPU32+ Features

Features of the CPU32+ are as follows:

- Fully Upward Object-Code Compatible with M68000 Family
- Virtual Memory Implementation
- Loop Mode of Instruction Execution
- Fast Multiply, Divide, and Shift Instructions
- Fast Bus Interface with Dynamic Bus Port Sizing
- Improved Exception Handling
- Additional Addressing Modes
  - — Scaled Index
  - — Address Register Indirect with Base Displacement and Index
  - — Expanded PC Relative Modes
  - — 32-Bit Branch Displacements
- Instruction Set Additions
  - — High-Precision Multiply and Divide
  - — Trap on Condition Codes
  - — Upper and Lower Bounds Checking
- Enhanced Breakpoint Instruction
- Trace on Change of Flow
- Table Lookup and Interpolate (TBL) Instruction
- LPSTOP Instruction
- Hardware $\overline{\text{BKPT}}$ Signal, Background Mode
- Fully Static Implementation

A block diagram of the CPU32+ is shown in Figure 5-1. The major blocks depicted operate in a highly independent fashion that maximizes concurrences of operation while managing the essential synchronization of instruction execution and bus operation. The bus controller loads instructions from the data bus into the decode unit. The sequencer and control unit provide overall chip control by managing the internal buses, registers, and functions of the execution unit.

5.1.2 Virtual Memory

A system that supports virtual memory has a limited amount of high-speed physical memory that can be accessed directly by the processor and maintains an image of a much larger virtual memory on a secondary storage device. When the processor attempts to access a location in the virtual memory map that is not resident in physical memory, a page fault occurs. The access to that location is temporarily suspended while the necessary data is fetched from secondary storage and placed in physical memory. The CPU32+ uses instruction restart, which requires that only a small portion of the internal

machine state be saved. After correcting the page fault, the machine state is restored, and the instruction is refetched and restarted. This process is completely transparent to the application program.

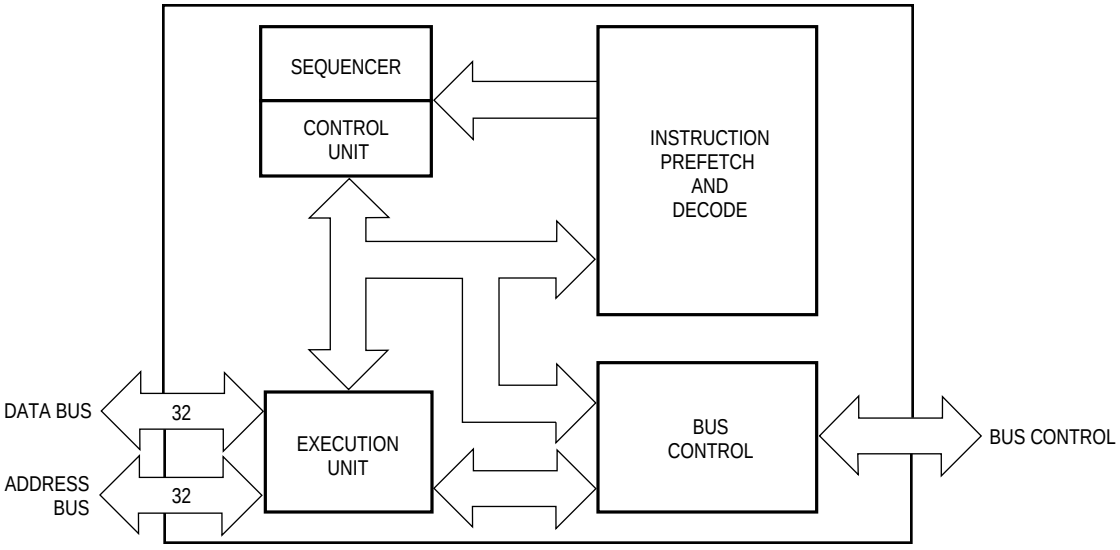


Figure 5-1. CPU32+ Block Diagram

5.1.3 Loop Mode Instruction Execution

The CPU32+ has several features that provide efficient execution of program loops. One of these features is the DBcc looping primitive instruction. To increase the performance of the CPU32+, a loop mode has been added to the processor. The loop mode is used by any single-word instruction that does not change the program flow. Loop mode is implemented in conjunction with the DBcc instruction. Figure 5-2 shows the required form of an instruction loop for the processor to enter loop mode.

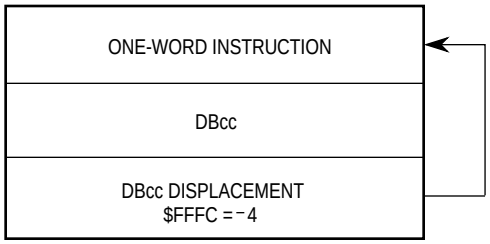


Figure 5-2. Loop Mode Instruction Sequence

The loop mode is entered when the DBcc instruction is executed and the loop displacement is –4. Once in loop mode, the processor performs only the data cycles associated with the instruction and suppresses all instruction fetches. The termination

condition and count are checked after each execution of the data operations of the looped instruction. The CPU32+ automatically exits the loop mode during interrupts or other exceptions.

5.1.4 Vector Base Register

The vector base register (VBR) contains the base address of the 1024-byte exception vector table, which consists of 256 exception vectors. Exception vectors contain the memory addresses of routines that begin execution at the completion of exception processing. These routines perform a series of operations appropriate for the corresponding exceptions. Because the exception vectors contain memory addresses, each vector consists of one long word, except the reset vector. The reset vector consists of two long words: the address used to initialize the supervisor stack pointer (SSP) and the address used to initialize the PC.

The address of an interrupt exception vector is derived from an 8-bit vector number and the VBR. The vector numbers for some exceptions are obtained from an external device; other numbers are supplied automatically by the processor. The processor multiplies the vector number by 4 to calculate the vector offset, which is added to the VBR. The sum is the memory address of the vector. All exception vectors are located in supervisor data space, except the reset vector, which is located in supervisor program space. Only the initial reset vector is fixed in the processor's memory map; once initialization is complete, there are no fixed assignments. Since the VBR provides the base address of the vector table, the vector table can be located anywhere in memory; it can even be dynamically relocated for each task that is executed by an operating system. Refer to **5.5 Exception Processing** for additional details.

5.1.5 Exception Handling

The processing of an exception occurs in four steps, with variations for different exception causes. During the first step, a temporary internal copy of the status register (SR) is made, and the SR is set for exception processing. During the second step, the exception vector is determined. During the third step, the current processor context is saved. During the fourth step, a new context is obtained, and the processor then proceeds with instruction processing.

Exception processing saves the most volatile portion of the current context by pushing it on the supervisor stack. This context is organized in a format called the exception stack frame. This information always includes the SR and PC context of the processor when the exception occurred. To support generic handlers, the processor places the vector offset in the exception stack frame. The processor also marks the frame with a frame format. The format field allows the return-from-exception (RTE) instruction to identify what information is on the stack so that it may be properly restored.

5.1.6 Addressing Modes

Addressing in the CPU32+ is register oriented. Most instructions allow the results of the specified operation to be placed either in a register or directly in memory; this flexibility eliminates the need for extra instructions to store register contents in memory.

The seven basic addressing modes are as follows:

- Register Direct
- Register Indirect
- Register Indirect with Index
- Program Counter Indirect with Displacement
- Program Counter Indirect with Index
- Absolute
- Immediate

Included in the register indirect addressing modes are the capabilities to postincrement, predecrement, and offset. The PC relative mode also has index and offset capabilities. In addition to these addressing modes, many instructions implicitly specify the use of the SR, SP and/or PC. Addressing is explained fully in the M68000PM/AD, *M68000 Family Programmer's Reference Manual***.**

5.2 CONFIGURABLE INSTRUCTION CACHE OVERVIEW

The CIC contains four dual-purpose memory blocks which can be independently configured as either instruction cache or SRAM. When configured as cache, each block provides 256 bytes of instruction cache, for a maximum 1-kbyte cache. The instruction cache improves system performance by providing cached instructions to the CPU32+ with very low latency. Alternatively, each element can be used as 512 bytes of SRAM, for a maximum of 2-kbytes SRAM. When configured as SRAM, the CIC provides a fast memory for general storage of data and/or instructions. The functionality of the blocks is independently programmable to allow a mix of instruction cache and SRAM for different applications. A high-level system diagram is shown in Figure 5-3.

A key feature of the CIC is its direct data path to the CPU32+. CPU accesses to the CIC do not require use of the IMB, resulting in increased IMB bandwidth available to other bus masters such as the DMA controller. This direct connection helps minimize the impact of DMA activity on CPU performance. The direct data path prevents accesses to the CIC by other modules on the IMB; hence, the CIC cannot be accessed from the IMB by the DMA controller.

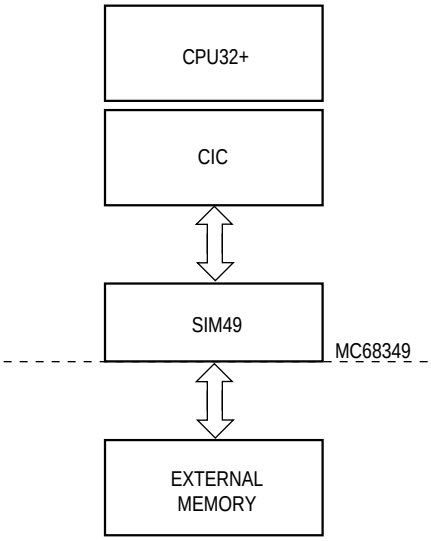


Figure 5-3. System Block Diagram

5.2.1 CIC Modes

The CIC blocks may be configured to operate in instruction cache mode or SRAM mode by programming the appropriate bits in the module configuration register (MCR). The following paragraphs describe these modes of operation.

5.2.1.1 INSTRUCTION CACHE MODE. When configured as cache, each CIC block functions as one of four possible sets of a set-associative instruction cache. Each set consists of 64 lines, which contain an address tag, status information, and four bytes of instruction information as shown in Figure 5-4. The address tag contains the upper 24 bits of the physical address. The status information consists of a valid bit for each of the two words in the cache line.

When the CPU32+ fetches an instruction, the CIC uses address bits A7–A2 as an offset to index into the cache and select one of the 64 lines in each set. The tags from the selected lines are compared against the physical address bits A31–A8. If one of the tags matches and the words corresponding to the access are valid, then the access hits in the cache and the requested instruction is transferred from the instruction cache to the CPU32+ in two clocks. Note that a hit on a cache line occurs only if all words referenced by the access are valid—if a long-word instruction fetch hits only on the second word of a cache line, the entire access is considered a miss.

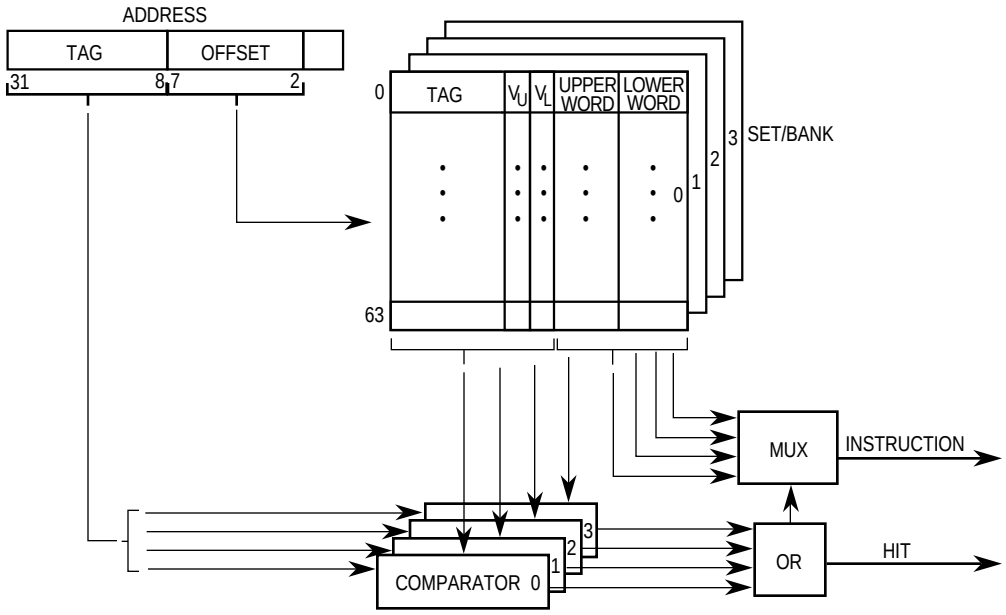


Figure 5-4. Instruction Cache

When a cache miss occurs, the instruction must be fetched from external memory across the IMB. A cache miss requires at least two clocks and is dependent on the speed of the external memory as well as IMB mastership at the time the cache miss occurs. If the CPU32+ has mastership of the IMB when a miss occurs, the instruction is fetched from external memory in (2+n) clocks, where n equals the number of wait states. If the CPU32+ does not have mastership of the IMB when a miss occurs, the instruction is fetched from external memory in (2+n+m) clocks, where m equals the number of clock cycles until the CPU32+ obtains mastership of the IMB. If the alternate bus master (DMA controller) is idle, then access times to the cache, QDMM, and 2-clock external memory are all 2 clocks. The performance benefits of the cache are more evident for slower external memory and/or high utilization of the IMB bandwidth by DMA activity.

For long-word instruction fetches, both words in the cache line are loaded and the two valid bits are set to flag the entire line as valid. Word accesses to 16-bit memory are stored into the word selected by address bit A1 of the fetch address; the other word in the line is invalidated unless the tag is not being changed (i.e., the cache line is being filled out). When a word fetch to a 32-bit memory port occurs, the cache controller uses the entire 32 bits from the IMB data bus to update both words in the cache line, requiring 32-bit external memory to always respond to a word instruction access with valid data for the full 32-bit port width (when at least one block of the CIC is configured as cache and enabled). Data accesses only require valid data for the bytes specifically requested.

Following a cache miss, the cache controller updates the cache with the new information in parallel with the bus access to fetch the required instruction(s) from main memory. Note that the cache does not allocate on instruction fetches from the QDMM, since accesses to the QDMM and the cache both take two clocks. For QDMM accesses, the instructions are

passed to the CPU32+ for execution, and the cache maintenance logic ignores the access. Only fetches from the external memory subsystem are cached, making maximum use of the internal memory elements of the processor.

If available, an invalid line is updated with the tag address and data from memory, and the line transitions from the INVALID state to the VALID state by setting the valid bit for each instruction word loaded. An invalid line is one in which one or both words are invalid—the replacement algorithm does not distinguish between lines which are completely invalid (neither valid bit set) and lines which are partially valid (one valid bit set). If all lines are already valid, a pseudo-random replacement technique is used to select one of the available lines and replace the tag and data contents of the line with the new line information. A cache block can be locked to prevent lines within it from being changed; however, the same function can be achieved for some applications by configuring the block as SRAM and preloading with instructions. Using a CIC block as SRAM also doubles the storage size from 256 bytes to 512 bytes (see **5.2.1.2 SRAM Mode**).

The replacement algorithm uses a 2-bit counter which is incremented on every access to the cache. When a miss occurs and all available lines in the cache are valid, the line in the block pointed to by the current counter value is replaced (provided that the block is not locked or in SRAM mode), after which the counter is incremented. If the block is locked or is in SRAM mode, the line in the next available block is replaced, where 0 is the lowest block and 3 is the highest block.

One or more cache sets can be selectively invalidated by setting the corresponding INV bit in the CIC module configuration register. All lines within a cache set are also invalidated when the block is switched from SRAM mode into instruction cache mode. The state diagram in Figure 5-5 shows the instruction cache word state transitions.

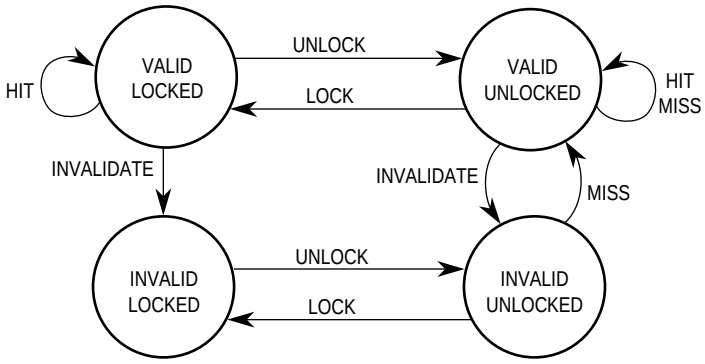


Figure 5-5. Instruction Cache Word State Diagram

The cache controller does not monitor data accesses by the CPU32+ or by alternate bus masters to maintain cache coherency. Any change in context that modifies memory previously fetched as instructions (such as changing code flow from a debugger or downloading new code) requires that the instruction cache be invalidated before executing the modified code.

5.2.1.2 SRAM MODE. Each of the four blocks in the CIC can be used as a 512-byte SRAM bank accessible only by the CPU32+, as shown in Figure 5-6. Each SRAM bank is relocatable on any 512-byte boundary. The memory contents can be write-protected on a bank basis by setting the corresponding write-protect (LWP) bit in the module configuration register (refer to **5.2.2 Programmer's Model** for more information).

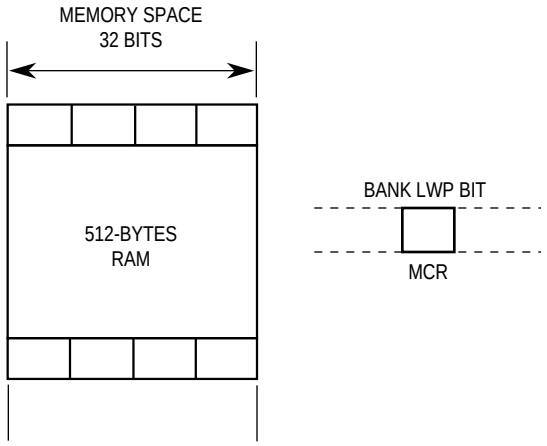


Figure 5-6. CIC SRAM Block Diagram

CIC SRAM banks are allowed to overlap external memory blocks defined by the SIM49 integrated chip selects and external address decode. An access which hits in the SRAM bank preempts an external bus cycle. However, all SRAM banks internal to the MC68349, whether in the CIC or the QDMM, should be mapped to unique memory locations relative to each other; data values returned for overlapping ranges are undefined. For more information about the QDMM, see **Section 6 Quad Data Memory Module**.

5.2.2 Programmer's Model

Figure 5-7 is a programmer's model (register map) of all registers in the CIC. The ADDR (address) column indicates the offset of the register from the address stored in the SIM49 module base address register (MBAR). The FC (function code) column indicates whether a register is restricted to supervisor access (S) or available to either supervisor or user space (S/U) accesses. All registers within the CIC are supervisor-only.

In the registers discussed in the following pages, the number in the upper right hand corner indicates the offset of the register from the address stored in the module base address register (MBAR). The numbers on the top line of the register represent the bit position in the register. The second line contains the mnemonic for the bit. The numbers below the register represent the bit values after reset. The access privilege is indicated in the lower right-hand corner.

ADDR	FC	31 … 0
FC0	S	Module Configuration Register (MCR)
FC4	S	SRAM Base Address Register 0 (BADDR0)
FC8	S	SRAM Base Address Register 1 (BADDR1)
FCC	S	SRAM Base Address Register 2 (BADDR2)
FD0	S	SRAM Base Address Register 3 (BADDR3)

Figure 5-7. CIC Programming Model

5.2.2.1 MODULE CONFIGURATION REGISTER (MCR). The MCR controls the CIC configuration. The register can be either read or written in supervisor state.

MCR $FC0

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
RESET: 0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved			LWP3	LWP2	LWP1	LWP0	INV3	INV2	INV1	INV0	MD3	MD2	MD1	MD0	ENAB
RESET 0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Supervisor Only

Bits 31–13—Reserved by Motorola

LWP3–0—Lock/write-protect corresponding CIC block

In blocks configured as instruction cache, the LWP bits indicate the lock status of the cache set.

1 = The set is locked. No lines will be replaced.
0 = The set is not locked. Lines may be replaced during misses or loads.

In blocks configured as SRAM, the LWP bits indicate the write-protect status of the SRAM banks.

1 = The SRAM bank is write-protected. An attempt to write to the bank returns $\overline{\text{BERR}}$.
0 = The SRAM bank is read/write.

INV3–0—Invalidate bits for corresponding instruction cache sets

The INV bits provide a way to invalidate the entire contents of sets in the instruction cache. These bits are write-only and any read attempts will return 0 (i.e., one-shot function for invalidating sets). Each set used for cache should be explicitly invalidated when first enabling the CIC.

1 = All lines in the set will be invalidated.
0 = Set line validity is not affected.

MD3–0—CIC block mode bits

The MD bits are used to specify the mode of the corresponding blocks.

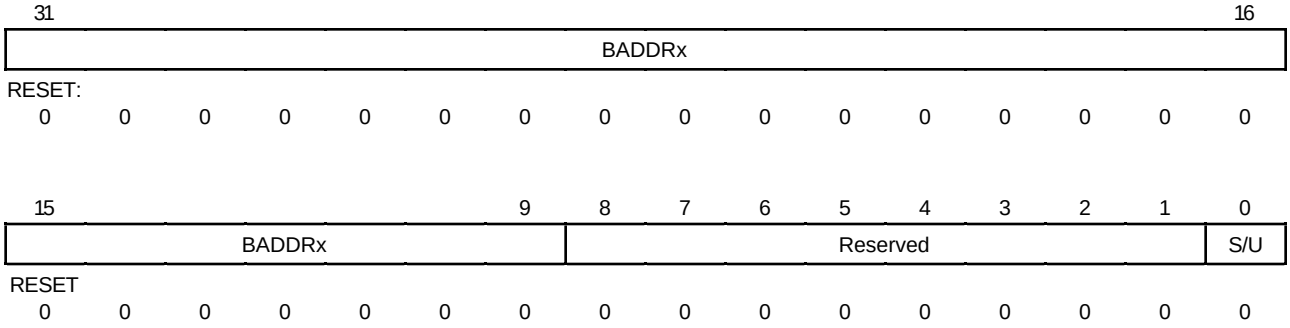
1 = The corresponding block is configured as an SRAM bank
0 = The corresponding block is configured as an instruction cache set

ENAB = CIC Module Enable Bit

1 = The CIC module is enabled.
0 = The CIC module is disabled.

5.2.2.2 SRAM BASE ADDRESS REGISTERS 3–0 (BADDR3–0). There are four SRAM base address registers, one corresponding to each of the CIC blocks.

BADDRx $FC4, $FC8, $FCC, $FD0



Supervisor Only

BADDR3–0—Base Address of corresponding SRAM bank

The BADDR field defines the upper 23 bits of the base address of SRAM banks. This field is only used when the corresponding CIC block is configured as SRAM.

Bits 8–1—Reserved by Motorola

S/U—Supervisor/User Space Bit

This bit identifies the SRAM bank as supervisor access only or supervisor/user space access. If a user space access is made to supervisor space, the SRAM returns $\overline{\text{BERR}}$.

1 = The bank is supervisor/user accessible.
0 = The bank allows supervisor access only.

5.3 ARCHITECTURE SUMMARY

The CPU32+ is upward source- and object-code compatible with the MC68000 and MC68010. It is downward source- and object-code compatible with the MC68020. Within the M68000 family, architectural differences are limited to the supervisory operating state. User programs can be executed unchanged on upward-compatible devices.

The major CPU32+ features are as follows:

- 32-Bit Internal Data Path and Arithmetic Hardware
- 32-Bit Address Bus Supported by 32-Bit Calculations
- Rich Instruction Set
- Eight 32-Bit General-Purpose Data Registers
- Seven 32-Bit General-Purpose Address Registers
- Separate User and Supervisor Stack Pointers (USP and SSP)
- Separate User and Supervisor Address Spaces
- Separate Program and Data Address Spaces
- Many Data Types
- Flexible Addressing Modes
- Full Interrupt Processing
- Expansion Capability

5.3.1 Programming Model

The CPU32+ programming model consists of two groups of registers that correspond to the user and supervisor privilege levels. User programs can only use the registers of the user model. The supervisor programming model, which supplements the user programming model, is used by CPU32+ system programmers who wish to protect sensitive operating system functions. The supervisor model is identical to that of MC68010 and later processors.

The CPU32+ has eight 32-bit data registers, seven 32-bit address registers, a 32-bit PC, separate 32-bit SSP and USP, a 16-bit status register (SR), two alternate function code registers, and a 32-bit VBR (see Figures 5-8 and 5-9).

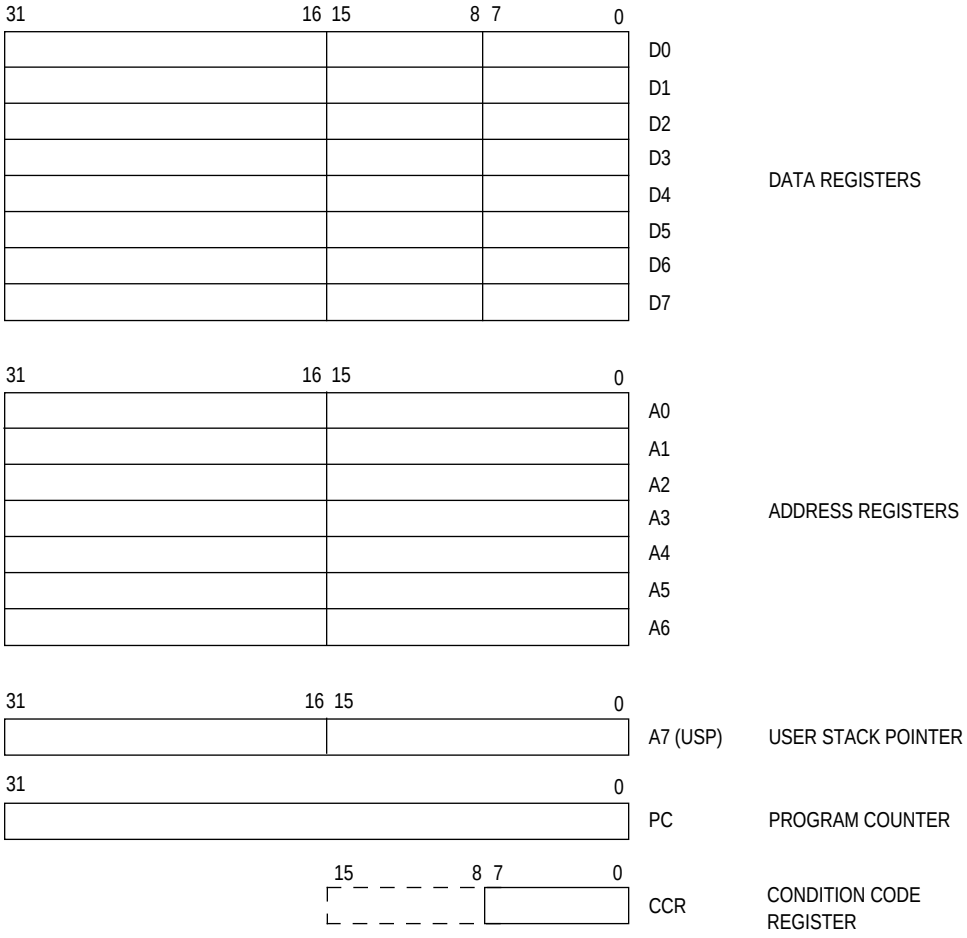


Figure 5-8. User Programming Model

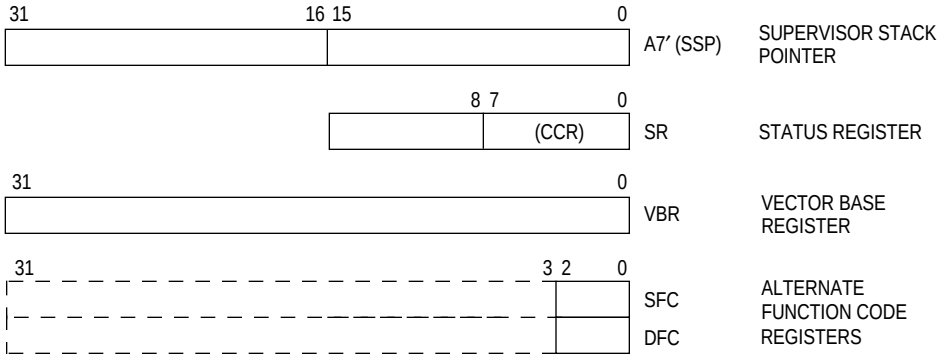


Figure 5-9. Supervisor Programming Model Supplement

5.3.2 Registers

Registers D7–D0 are used as data registers for bit, byte (8-bit), word (16-bit), long-word (32-bit), and quad-word (64-bit) operations. Registers A6 to A0 and the USP and SSP are address registers that may be used as software SPs or base address registers. Register A7 (shown as A7 and A7' in Figures 5-8 and 5-9) is a register designation that applies to the USP in the user privilege level and to the SSP in the supervisor privilege level. In addition, address registers may be used for word and long-word operations. All 16 general-purpose registers (D7–D0, A7–A0) may be used as index registers.

The PC contains the address of the next instruction to be executed by the CPU32+. During instruction execution and exception processing, the processor automatically increments the contents of the PC or places a new value in the PC, as appropriate.

The SR (see Figure 5-10) contains condition codes, an interrupt priority mask (three bits), and three control bits. Condition codes reflect the results of a previous operation. The codes are contained in the low byte (condition code register (CCR)) of the SR. The interrupt priority mask determines the level of priority an interrupt must have to be acknowledged. The control bits determine trace mode and privilege level. At user privilege level, only the CCR is available. At supervisor privilege level, software can access the full SR.

The VBR contains the base address of the exception vector table in memory. The displacement of an exception vector is added to the value in this register to access the vector table.

Alternate source and destination function code registers (SFC and DFC) contain 3-bit function codes. The CPU32+ generates a function code each time it accesses an address. Specific codes are assigned to each type of access. The codes can be used to select eight dedicated 4-Gbyte address spaces. The MOVEC instruction can use registers SFC and DFC to specify the function code of a memory address.

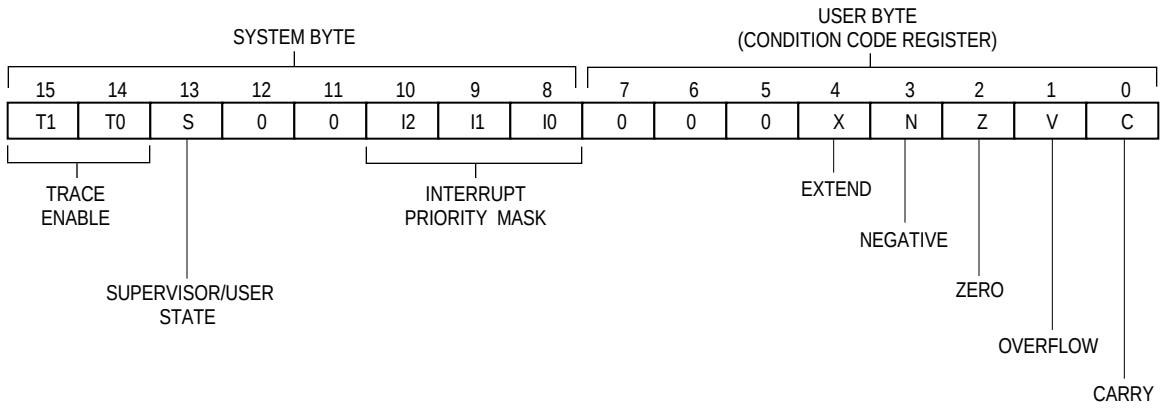


Figure 5-10. Status Register

5.4 INSTRUCTION SET

The following paragraphs describe the CPU32+ instruction set. A description of the instruction format, the operands used by the instructions, and a summary of the instructions by category are included. Complete programming information is provided in the M68000PM/AD, *M68000 Family Programmer's Reference Manual.*

The CPU32+ instructions include machine functions for all the following operations:

- Data Movement
- Arithmetic Operations
- Logical Operations
- Shifts and Rotates
- Bit Manipulation
- Conditionals and Branches
- System Control

The large instruction set encompasses a complete range of capabilities and, combined with the enhanced addressing modes, provides a flexible base for program development.

The instruction set of the CPU32+ is very similar to that of the MC68020. The following M68020 instructions are not implemented on the CPU32+:

BFxx	—	Bit Field Instructions (BFCHG, BFCLR, BFEXTS, BFEXTU, BFFFO, BFINS, BFSET, BFTST)
CALLM, RTM	—	Call Module, Return Module
CAS, CAS2	—	Compare and Set (Read-Modify-Write Instructions)
cpxxx	—	Coprocessor Instructions (cpBcc, cpDBcc, cpGEN, cpRESTORE, cpSAVE, cpScc, cpTRAPcc)
PACK, UNPK	—	Pack, Unpack BCD Instructions

The CPU32+ traps on unimplemented instructions or illegal effective addressing modes, allowing user-supplied code to emulate unimplemented capabilities or to define special-purpose functions. However, Motorola reserves the right to use all currently unimplemented instruction operation codes for future M68000 core enhancements.

Table 5-1 lists the instruction set of the MC68349.

Table 5-1. Instruction Set

Mnemonic	Description
ABCD	Add Decimal with Extend
ADD	Add
ADDA	Add Address
ADDI	Add Immediate
ADDQ	Add Quick
AND	Logical AND
ANDI	Logical AND Immediate
ASL	Arithmetic Shift Left
ASR	Arithmetic Shift Right
Bcc	Branch Conditionally (16 Tests)
BCHG	Bit Test and Change
BCLR	Bit Test and Clear
BGND	Enter Background Mode
BKPT	Breakpoint
BRA	Branch Always
BSET	Bit Test and Set
BSR	Branch to Subroutine
BTST	Bit Test
CHK	Check Register against Bounds
CHK2	Check Register against Upper and Lower Bounds
CLR	Clear Operand
CMP	Compare
CMPA	Compare Address
CMPI	Compare Immediate
CMPM	Compare Memory
CMP2	Compare Register against Upper and Lower Bounds
DBcc	Test Condition, Decrement and Branch (16 Tests)
DIVS, DIVSL	Signed Divide
DIVU, DIVUL	Unsigned Divide
EOR	Logical Exclusive OR
EORI	Logical Exclusive OR Immediate
EXG	Exchange Registers
EXT, EXTB	Sign Extend
ILLEGAL	Take Illegal Instruction Trap
JMP	Jump
JSR	Jump to Subroutine
LEA	Load Effective Address
LINK	Link and Allocate
LPSTOP	Low-Power Stop
LSL, LSR	Logical Shift Left and Right
MOVE	Move
MOVEA	Move Address
MOVE CCR	Move Condition Code Register
MOVE SR	Move to/from Status Register
MOVE USP	Move User Stack Pointer
MOVEC	Move Control Register
MOVEM	Move Multiple Registers
MOVEP	Move Peripheral Data
MOVEQ	Move Quick
MOVES	Move Alternate Address Space
MULS	Signed Multiply
MULU	Unsigned Multiply
NBCD	Negate Decimal with Extend
NEG	Negate
NEGX	Negate with Extend
NOP	No Operation
NOT	Ones Complement
OR	Logical Inclusive OR
ORI	Logical Inclusive OR Immediate
PEA	Push Effective Address
RESET	Reset External Devices
ROL, ROR	Rotate Left and Right
ROXL, ROXR	Rotate with Extend Left and Right
RTD	Return and Deallocate
RTE	Return from Exception
RTR	Return and Restore
RTS	Return from Subroutine
SBCD	Subtract Decimal with Extend
Scc	Set Conditionally
STOP	Stop
SUB	Subtract
SUBA	Subtract Address
SUBI	Subtract Immediate
SUBQ	Subtract Quick
SUBX	Subtract with Extend
SWAP	Swap Data Register Halves
TAS	Test and Set Operand
TBLS, TBLSN	Table Lookup and Interpolate, Signed
TBLU, TBLUN	Table Lookup and Interpolate, Unsigned
TRAPcc	Trap Conditionally (16 Tests)
TRAPV	Trap on Overflow
TST	Test
UNLK	Unlink

5.4.1 M68000 Family Compatibility

It is the philosophy of the M68000 Family that all user-mode programs should execute unchanged on a more advanced processor and that supervisor-mode programs and exception handlers should require only minimal alteration.

The CPU32+ can be thought of as an intermediate member of the M68000 family. Object code from an MC68000 or MC68010 may be executed on the CPU32+, and many of the instruction and addressing mode extensions of the MC68020 are also supported.

5.4.1.1 NEW INSTRUCTIONS. Two instructions have been added to the M68000 instruction set: LPSTOP and TBL.

5.4.1.1.1 Low-Power Stop (LPSTOP). In applications where power consumption is a consideration, the CPU32+ can force the device into a low-power standby mode when immediate processing is not required. The low-power mode is entered by executing the LPSTOP instruction. The processor remains in this mode until a user-specified or higher level interrupt or a reset occurs.

5.4.1.1.2 Table Lookup and Interpolate (TBL). To maximize throughput for real-time applications, reference data is often precalculated and stored in memory for quick access. The storage of sufficient data points can require an inordinate amount of memory. The TBL instruction uses linear interpolation to recover intermediate values from a sample of data points, thus conserving memory.

When the TBL instruction is executed, the CPU32+ looks up two table entries bounding the desired result and performs a linear interpolation between them. Byte, word, and long-word operand sizes are supported. The result can be rounded according to a round-to-nearest algorithm or returned unrounded along with the fractional portion of the calculated result (byte and word results only). This extra precision can be used to reduce cumulative error in complex calculations. See **5.4.2 Using the TBL Instructions** for examples.

5.4.1.2 UNIMPLEMENTED INSTRUCTIONS. The ability to trap on unimplemented instructions allows user-supplied code to emulate unimplemented capabilities or to define special-purpose functions. However, Motorola reserves the right to use all currently unimplemented instruction operation codes for future M68000 enhancements. See **5.6.2.8 Illegal or Unimplemented Instructions** for more details.

5.4.2 Using the TBL Instructions

There are four TBL instructions. TBLS returns a signed, rounded byte, word, or long-word result. TBLSN returns a signed, unrounded byte, word, or long-word result. TBLU returns an unsigned, rounded byte, word, or long-word result. TBLUN returns an unsigned, unrounded byte, word, or long-word result. All four instructions support two types of interpolation data: an n-element table stored in memory and a two-element range stored in a pair of data registers. The latter form provides a means of performing surface (3D) interpolation between two previously calculated linear interpolations.

The following examples show how to compress tables and use fewer interpolation levels between table entries. Example 1 (see Figure 5-7) demonstrates TBL for a 257-entry table, allowing up to 256 interpolation levels between entries. Example 2 (see Figure 5-8) reduces table length for the same data to four entries. Example 3 (see Figure 5-9) demonstrates use of an 8-bit independent variable with an instruction.

Two additional examples show how TBLSN can reduce cumulative error when multiple table lookup and interpolation operations are used in a calculation. Example 4 demonstrates addition of the results of three table interpolations. Example 5 illustrates use of TBLSN in surface interpolation.

5.4.2.1 TABLE EXAMPLE 1: STANDARD USAGE. The table consists of 257 word entries. As shown in Figure 5-11, the function is linear within the range $32768 \leq X \leq 49152$. Table entries within this range are as given in Table 5-2 .

Table 5-2. Standard Usage Entries

Entry Number	X-Value	Y-Value
128*	32768	1311
162	41472	1659
163	41728	1669
164	41984	1679
165	42240	1690
192*	49152	1966

*These values are the end points of the range.
All entries between these points fall on the line.

Figure 5-11. Table Example 1

The table instruction is executed with the following bit pattern in Dx:

31 … 16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
NOT USED	1	0	1	0	0	0	1	1	1	0	0	0	0	0	0	0

Table Entry Offset ⇒ Dx [8:15] = $A3 = 163

Interpolation Fraction ⇒ Dx [0:7] = $80 = 128

Using this information, the table instruction calculates dependent variable Y:

Y = 1669 + (128 (1679 – 1669)) / 256 = 1674

5.4.2.2 TABLE EXAMPLE 2: COMPRESSED TABLE. In Example 2 (see Figure 5-12), the data from Example 1 has been compressed by limiting the maximum value of the independent variable. Instead of the range 0 ≤ X = 65535, X is limited to 0 ≤ X ≤ 1023. The table has been compressed to only five entries, but up to 256 levels of interpolation are allowed between entries.

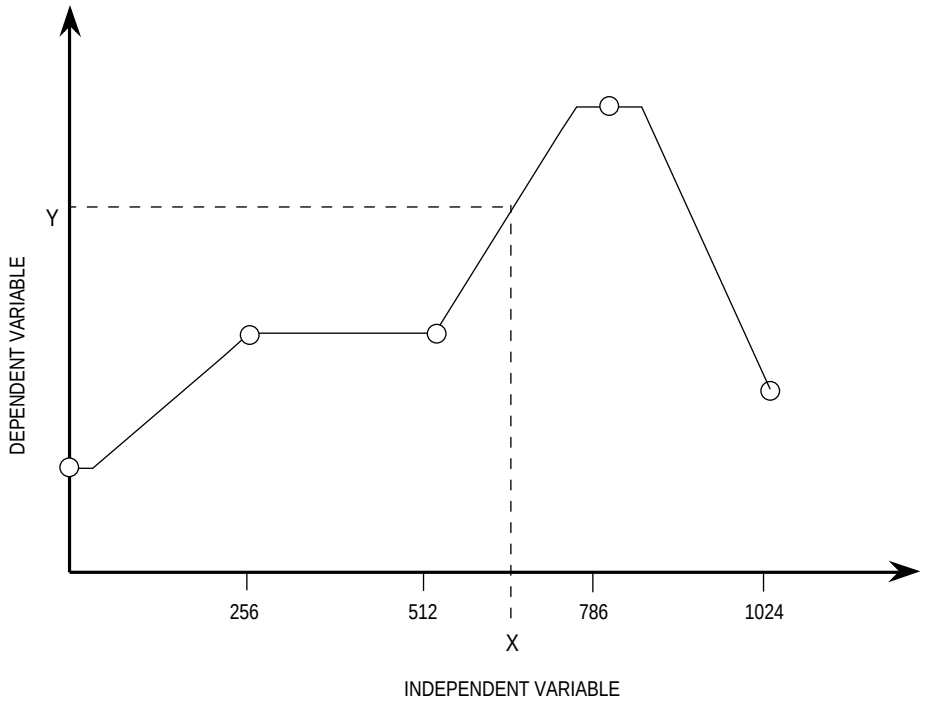


Figure 5-12. Table Example 2

NOTE

Extreme table compression with many levels of interpolation is possible only with highly linear functions. The table entries within the range of interest are listed in Table 5-3.

Table 5-3. Compressed Table Entries

Entry Number	X-Value	Y-Value
2	512	1311
3	786	1966

Since the table is reduced from 257 to 5 entries, independent variable X must be scaled appropriately. In this case the scaling factor is 64, and the scaling is done by a single instruction:

LSR.W #6,Dx

Thus, Dx now contains the following bit pattern:

31 … 16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
NOT USED	0	0	0	0	0	0	1	0	1	0	0	0	1	1	1	0

Table Entry Offset ⇒ Dx [8:15] = $02 = 2

Interpolation Fraction ⇒ Dx [0:7] = $8E = 142

Using this information, the table instruction calculates dependent variable Y:

Y = 1331 + (142 (1966 – 1311)) / 256 = 1674

The function chosen for Examples 1 and 2 is linear between data points. If another function had been used, interpolated values might not have been identical.

5.4.2.3 TABLE EXAMPLE 3: 8-BIT INDEPENDENT VARIABLE. This example shows how to use a table instruction within an interpolation subroutine. Independent variable X is calculated as an 8-bit value, allowing 16 levels of interpolation on a 17-entry table. X is passed to the subroutine, which returns an 8-bit result. The subroutine uses the data listed in Table 5-4, based on the function shown in Figure 5-13.

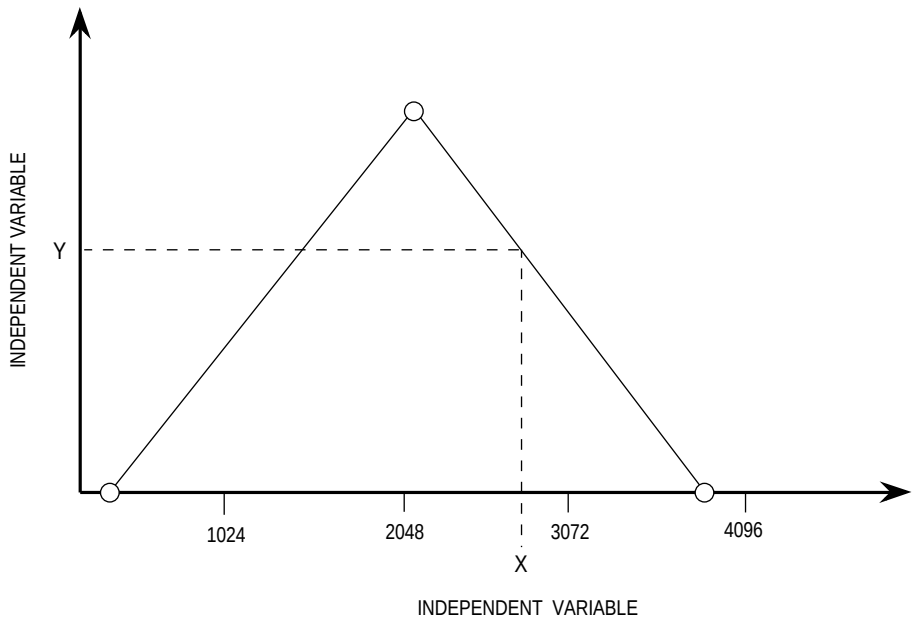


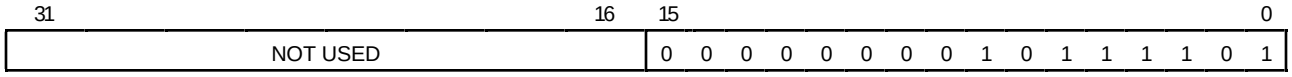
Figure 5-13. Table Example 3

Table 5-4. 8-Bit Independent Variable Entries

X (Subroutine)	X (Instruction)	Y
0	0	0
1	256	16
2	512	32
3	768	48
4	1024	64
5	1280	80
6	1536	96
7	1792	112
8	2048	128
9	2304	112
10	2560	96
11	2816	80
12	3072	64
13	3328	48
14	3584	32
15	3840	16
16	4096	0

The first column is the value passed to the subroutine, the second column is the value expected by the table instruction, and the third column is the result returned by the subroutine.

The following value has been calculated for independent variable X:



Since X is an 8-bit value, the upper four bits are used as a table offset, and the lower four bits are used as an interpolation fraction. The following results are obtained from the subroutine:

$$\text{Table Entry Offset} \Rightarrow Dx\ [4{:}7] = \$B = 11$$

$$\text{Interpolation Fraction} \Rightarrow Dx\ [0{:}3] = \$D = 13$$

Thus, Y is calculated as follows:

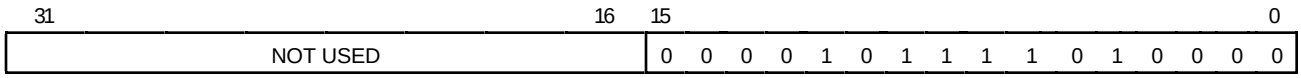
$$Y = 80 + (13\ (64 - 80)) / 16 = 67$$

If the 8-bit value for X were used directly by the table instruction, interpolation would be incorrectly performed between entries 0 and 1. Data must be shifted to the left four places before use:

LSL.W #4, Dx

The new range for X is $0 \leq X \leq 4096$; however, since a left shift fills the least significant digits of the word with zeros, the interpolation fraction can only have one of 16 values.

After the shift operation, Dx contains the following value:



Execution of the table instruction using the new value in Dx yields:

$$\text{Table Entry Offset} \Rightarrow Dx\ [8{:}15] = \$0B = 11$$

$$\text{Interpolation Fraction} \Rightarrow Dx\ [0{:}7] = \$D0 = 208$$

Thus, Y is calculated as follows:

$$Y = 80 + (208\ (64 - 80)) / 256 = 67$$

5.4.2.4 TABLE EXAMPLE 4: MAINTAINING PRECISION. In this example, three TBL operations are performed and the results are summed. The calculation is done once with the result of each TBL rounded before addition and once with only the final result rounded. Assume that the result of the three interpolations are as follows (a "." indicates the binary radix point).

TBL # 1	0010 0000 . 0111 0000
TBL# 2	0011 1111 . 0111 0000
TBL # 3	0000 0001 . 0111 0000

First, the results of each TBL are rounded with the TBLS round-to-nearest-even algorithm. The following values would be returned by TBLS:

```
TBL # 1        0010  0000 .
TBL # 2        0011  1111 .
TBL # 3        0000  0001 .
```

Summing, the following result is obtained:

```
0010  0000 .
0011  1111 .
0000  0001 .
0110  0000 .
```

Now, using the same TBL results, the sum is first calculated and then rounded according to the same algorithm:

```
0010  0000 . 0111  0000
0011  1111 . 0111  0000
0000  0001 . 0111  0000
0110  0001 . 0101  0000
```

Rounding yields:

```
0110  0001 .
```

The second result is preferred. The following code sequence illustrates how addition of a series of table interpolations can be performed without loss of precision in the intermediate results:

```
L0:
    TBLSN.B     ⟨ea⟩, Dx
    TBLSN.B     ⟨ea⟩, Dx
    TBLSN.B     ⟨ea⟩, Dl
    ADD.L       Dx, Dm      Long addition avoids problems with carry
    ADD.L       Dm, Dl
    ASR.L       #8, Dl      Move radix point
    BCC.B       L1          Fraction MSB in carry
    ADDQ.B      #1, Dl
L1: . . .
```

5.4.2.5 TABLE EXAMPLE 5: SURFACE INTERPOLATIONS. The various forms of table can be used to perform surface (3D) TBLs. However, since the calculation must be split into a series of 2D TBLs, it is possible to lose precision in the intermediate results. The following code sequence, incorporating both TBLS and TBLSN, eliminates this possibility.

```
L0:
    MOVE.W      Dx, Dl          Copy entry number and fraction number
    TBLSN.B     ⟨ea⟩, Dx
    TBLSN.B     ⟨ea⟩, Dl
    TBLS.W      Dx:Dl, Dm       Surface interpolation, with round
    ASR.L       #8, Dm          Read just the result
    BCC.B       L1              No round necessary
    ADDQ.B      #1, Dl          Half round up
L1: . . .
```

Before execution of this code sequence, Dx must contain fraction and entry numbers for the two TBL, and Dm must contain the fraction for surface interpolation. The ⟨ea⟩ fields in the TBLSN instructions point to consecutive columns in a 3D table. The TBLS size parameter must be word if the TBLSN size parameter is byte, and must be long word if TBLSN is word. Increased size is necessary because a larger number of significant digits is needed to accommodate the scaled fractional results of the 2D TBL.

5.4.3 Nested Subroutine Calls

The LINK instruction pushes an address onto the stack, saves the stack address at which the address is stored, and reserves an area of the stack for use. Using this instruction in a series of subroutine calls will generate a linked list of stack frames.

The UNLK instruction removes a stack frame from the end of the list by loading an address into the SP and pulling the value at that address from the stack. When the instruction operand is the address of the link address at the bottom of a stack frame, the effect is to remove the stack frame from both the stack and the linked list.

5.4.4 Pipeline Synchronization with the NOP Instruction

Although the no operation (NOP) instruction performs no visible operation, it does force synchronization of the instruction pipeline, since all previous instructions must complete execution before the NOP begins.

5.5 PROCESSING STATES

This section describes the processing states of the CPU32+. It includes a functional description of the bits in the supervisor portion of the SR and an overview of actions taken by the processor in response to exception conditions.

5.5.1 State Transitions

The processor is always in one of four processing states: normal, background, exception, or halted.

When the processor fetches instructions and operands or executes instructions, it is in the normal processing state. The stopped condition, which the processor enters when a STOP or LPSTOP instruction is executed, is a variation of the normal state in which no further bus cycles are generated.

Background state is an alternate operational mode used for system debugging. Refer to **5.7 Development Support** for more information.

Exception processing refers specifically to the transition from normal processing of a program to normal processing of system routines, interrupt routines, and other exception handlers. Exception processing includes the stack operations, the exception vector fetch, and the filling of the instruction pipeline caused by an exception. Exception processing ends when execution of an exception handler routine begins. Refer to **5.6 Exception Processing** for comprehensive information.

A catastrophic system failure occurs if the processor detects a bus error or generates an address error while in the exception processing state. This type of failure halts the processor. For example, if a bus error occurs during exception processing caused by another bus error, the CPU32+ assumes that the system is not operational and halts.

The halted condition should not be confused with the stopped condition. After the processor executes a STOP or LPSTOP instruction, execution of instructions can resume when a trace, interrupt, or reset exception occurs.

5.5.2 Privilege Levels

To protect system resources, the processor can operate with either of two levels of access—user or supervisor. Supervisor level is more privileged than user level. All instructions are available at the supervisor level, but execution of some instructions is not permitted at the user level. There are separate SPs for each level. The S-bit in the SR indicates privilege level and determines which SP is used for stack operations. The processor identifies each bus access (supervisor or user mode) via function codes to enforce supervisor and user access levels.

In a typical system, most programs execute at the user level. User programs can access only their own code and data areas and are restricted from accessing other information. The operating system executes at the supervisor privilege level, has access to all resources, performs the overhead tasks for the user level programs, and coordinates their activities.

5.5.2.1 SUPERVISOR PRIVILEGE LEVEL. If the S-bit in the SR is set, supervisor privilege level applies, and all instructions are executable. The bus cycles generated for instructions executed in supervisor level are normally classified as supervisor references, and the values of the function codes on FC2–FC0 refer to supervisor address spaces.

All exception processing is performed at the supervisor level. All bus cycles generated during exception processing are supervisor references, and all stack accesses use the SSP.

Instructions that have important system effects can only be executed at supervisor level. For instance, user programs are not permitted to execute STOP, LPSTOP, or RESET instructions. To prevent a user program from gaining privileged access, except in a controlled manner, instructions that can alter the S-bit in the SR are privileged. The TRAP #n instruction provides controlled user access to operating system services.

5.5.2.2 USER PRIVILEGE LEVEL. If the S-bit in the SR is cleared, the processor executes instructions at the user privilege level. The bus cycles for an instruction executed at the user privilege level are classified as user references, and the values of the function codes on FC2–FC0 specify user address spaces. While the processor is at the user level, implicit references to the system SP and explicit references to address register seven (A7) refer to the USP.

5.5.2.3 CHANGING PRIVILEGE LEVEL. To change from user privilege level to supervisor privilege level, a condition that causes exception processing must occur. When exception processing begins, the current values in the SR, including the S-bit, are saved on the supervisor stack, and then the S-bit is set to enable supervisor access. Execution continues at supervisor privilege level until exception processing is complete.

To return to user access level, a system routine must execute one of the following instructions: MOVE to SR, ANDI to SR, EORI to SR, ORI to SR, or RTE. These instructions execute only at supervisor privilege level and can modify the S-bit of the SR. After these instructions execute, the instruction pipeline is flushed, then refilled from the appropriate address space.

The RTE instruction causes a return to a program that was executing when an exception occurred. When RTE is executed, the exception stack frame saved on the supervisor stack can be restored in either of two ways.

If the frame was generated by an interrupt, breakpoint, trap, or instruction exception, the SR and PC are restored to the values saved on the supervisor stack, and execution resumes at the restored PC address, with access level determined by the S-bit of the restored SR.

If the frame was generated by a bus error or an address error exception, the entire processor state is restored from the stack.

5.6 EXCEPTION PROCESSING

An exception is a special condition that pre-empts normal processing. Exception processing is the transition from normal mode program execution to execution of a routine that deals with an exception. The following paragraphs discuss system resources related to exception handling, exception processing sequence, and specific features of individual exception processing routines.

5.6.1 Exception Vectors

An exception vector is the address of a routine that handles an exception. The VBR contains the base address of a 1024-byte exception vector table, which consists of 256 exception vectors. Sixty-four vectors are defined by the processor, and 192 vectors are reserved for user definition as interrupt vectors. Except for the reset vector, which is two long words, each vector in the table is one long word. Refer to Table 5-5 for information on vector assignment.

CAUTION

Because there is no protection on the 64 processor-defined vectors, external devices can access vectors reserved for internal purposes. This practice is strongly discouraged.

All exception vectors, except the reset vector, are located in supervisor data space. The reset vector is located in supervisor program space. Only the initial reset vector is fixed in the processor memory map. When initialization is complete, there are no fixed assignments. Since the VBR stores the vector table base address, the table can be located anywhere in memory. It can also be dynamically relocated for each task executed by an operating system.

Each vector is assigned an 8-bit number. Vector numbers for some exceptions are obtained from an external device; others are supplied by the processor. The processor multiplies the vector number by 4 to calculate vector offset, then adds the offset to the contents of the VBR. The sum is the memory address of the vector.

Table 5-5. Exception Vector Assignments

Vector Number	Vector Offset			Assignment
	Dec	Hex	Space	
0	0	000	SP	Reset: Initial Stack Pointer
1	4	004	SP	Reset: Initial Program Counter
2	8	008	SD	Bus Error
3	12	00C	SD	Address Error
4	16	010	SD	Illegal Instruction
5	20	014	SD	Zero Division
6	24	018	SD	CHK, CHK2 Instructions
7	28	01C	SD	TRAPcc, TRAPV Instructions
8	32	020	SD	Privilege Violation
9	36	024	SD	Trace
10	40	028	SD	Line 1010 Emulator
11	44	02C	SD	Line 1111 Emulator
12	48	030	SD	Hardware Breakpoint
13	52	034	SD	(Reserved for Coprocessor Protocol Violation)
14	56	038	SD	Format Error
15	60	03C	SD	Uninitialized Interrupt
16–23	64 92	040 05C	SD	(Unassigned, Reserved) —
24	96	060	SD	Spurious Interrupt
25	100	064	SD	Level 1 Interrupt Autovector
26	104	068	SD	Level 2 Interrupt Autovector
27	108	06C	SD	Level 3 Interrupt Autovector
28	112	070	SD	Level 4 Interrupt Autovector
29	116	074	SD	Level 5 Interrupt Autovector
30	120	078	SD	Level 6 Interrupt Autovector
31	124	07C	SD	Level 7 Interrupt Autovector
32–47	128 188	080 0BC	SD	Trap Instruction Vectors (0–15) —
48–58	192 232	0C0 0E8	SD	(Reserved for Coprocessor) —
59–63	236 252	0EC 0FC	SD	(Unassigned, Reserved) —
64–255	256 1020	100 3FC	SD	User-Defined Vectors (192)

5.6.1.1 TYPES OF EXCEPTIONS. An exception can be caused by internal or external events.

An internal exception can be generated by an instruction or by an error. The TRAP, TRAPcc, TRAPV, BKPT, CHK, CHK2, RTE, and DIV instructions can cause exceptions during normal execution. Illegal instructions, instruction fetches from odd addresses, word or long-word operand accesses from odd addresses, and privilege violations also cause internal exceptions.

Sources of external exception include interrupts, breakpoints, bus errors, and reset requests. Interrupts are peripheral device requests for processor action. Breakpoints are used to support development equipment. Bus error and reset are used for access control and processor restart.

5.6.1.2 EXCEPTION PROCESSING SEQUENCE. For all exceptions other than a reset exception, exception processing occurs in the following sequence. Refer to **5.6.2.1 Reset** for details of reset processing.

As exception processing begins, the processor makes an internal copy of the SR. After the copy is made, the processor state bits in the SR are changed—the S-bit is set, establishing supervisor access level, and bits T1 and T0 are cleared, disabling tracing. For reset and interrupt exceptions, the interrupt priority mask is also updated.

Next, the exception number is obtained. For interrupts, the number is fetched from CPU space $F (the bus cycle is an interrupt acknowledge). For all other exceptions, internal logic provides a vector number.

Next, current processor status is saved. An exception stack frame is created and placed on the supervisor stack. All stack frames contain copies of the SR and the PC for use by RTE. The type of exception and the context in which the exception occurs determine what other information is stored in the stack frame.

Finally, the processor prepares to resume normal execution of instructions. The exception vector offset is determined by multiplying the vector number by 4, and the offset is added to the contents of the VBR to determine displacement into the exception vector table. The exception vector is loaded into the PC. If no other exception is pending, the processor will resume normal execution at the new address in the PC.

5.6.1.3 EXCEPTION STACK FRAME. During exception processing, the most volatile portion of the current context is saved on the top of the supervisor stack. This context is organized in a format called the exception stack frame.

The exception stack frame always includes the contents of SR and PC at the time the exception occurred. To support generic handlers, the processor also places the vector offset in the exception stack frame and marks the frame with a format code. The format field allows an RTE instruction to identify stack information so that it can be properly restored.

The general form of the exception stack frame is illustrated in Figure 5-14. Although some formats are peculiar to a particular M68000 family processor, format 0000 is always legal and always indicates that only the first four words of a frame are present. See **5.6.4 CPU32+ Stack Frames** for a complete discussion of exception stack frames.

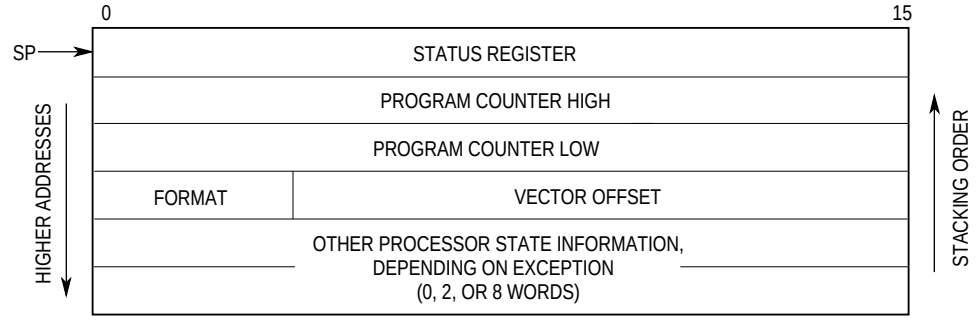


Figure 5-14. Exception Stack Frame

5.6.1.4 MULTIPLE EXCEPTIONS. Each exception has been assigned a priority based on its relative importance to system operation. Priority assignments are shown in Table 5-6. Group 0 exceptions have the highest priorities; group 4 exceptions have the lowest priorities. Exception processing for exceptions that occur simultaneously is done by priority, from highest to lowest.

It is important to be aware of the difference between exception processing mode and execution of an exception handler. Each exception has an assigned vector that points to an associated handler routine. Exception processing includes steps described in **5.6.1.2 Exception Processing Sequence**, but does not include execution of handler routines, which is done in normal mode.

When the CPU32+ completes exception processing, it is ready to begin either exception processing for a pending exception or execution of a handler routine. Priority assignment governs the order in which exception processing occurs, not the order in which exception handlers are executed.

As a general rule, when simultaneous exceptions occur, the handler routines for lower priority exceptions are executed before the handler routines for higher priority exceptions. For example, consider the arrival of an interrupt during execution of a TRAP instruction while tracing is enabled. Trap exception processing (2) is done first, followed immediately by exception processing for the trace (4.1), and then by exception processing for the interrupt (4.3). Each exception places a new context on the stack. When the processor resumes normal instruction execution, it is vectored to the interrupt handler, which returns to the trace handler that returns to the trap handler.

Table 5-6. Exception Priority Groups

Group.Priority	Exception and Relative Priority	Characteristics
0	Reset	Aborts all processing (instruction or exception); does not save old context.
1.1 1.2	Address Error Bus Error	Suspends processing (instruction or exception); saves internal context.
2	BKPT#n, CHK, CHK2, Division by Zero, RTE, TRAP#n, TRAPcc, TRAPV	Exception processing is a part of instruction execution.
3	Illegal Instruction, Line A, Unimplemented Line F, Privilege Violation	Exception processing begins before instruction execution.
4.1 4.2 4.3	Trace Hardware Breakpoint Interrupt	Exception processing begins when current instruction or previous exception processing is complete.

There are special cases to which the general rule does not apply. The reset exception will always be the first exception handled since reset clears all other exceptions. It is also possible for high-priority exception processing to begin before low-priority exception processing is complete. For example, if a bus error occurs during trace exception processing, the bus error will be processed and handled before trace exception processing has completed.

5.6.2 Processing of Specific Exceptions

The following paragraphs provide details concerning sources of specific exceptions, how each arises, and how each is processed.

5.6.2.1 RESET. Assertion of $\overline{\text{RESET}}$ by external hardware or assertion of the internal $\overline{\text{RESET}}$ signal by an internal module causes a reset exception. The reset exception has the highest priority of any exception. Reset is used for system initialization and for recovery from catastrophic failure. When the reset exception is recognized, it aborts any processing in progress, and that processing cannot be recovered. Reset performs the following operations:

1. Clears T0 and T1 in the SR to disable tracing
2. Sets the S-bit in the SR to establish supervisor privilege
3. Sets the interrupt priority mask to the highest priority level ($7)
4. Initializes the VBR to zero ($00000000)
5. Generates a vector number to reference the reset exception vector
6. Loads the first long word of the vector into the interrupt SP
7. Loads the second long word of the vector into the PC
8. Fetches and initiates decode of the first instruction to be executed

Figure 5-15 is a flowchart of the reset exception

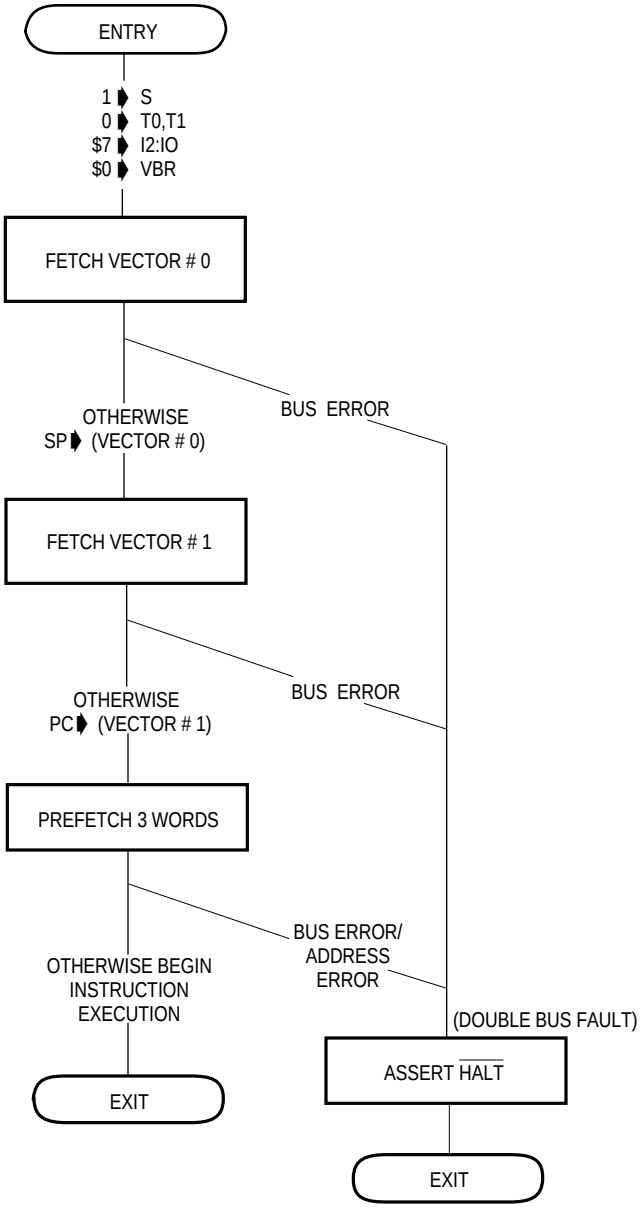


Figure 5-15. Reset Operation Flowchart

After initial instruction prefetches, normal program execution begins at the address in the PC. The reset exception does not save the value of either the PC or the SR.

If a bus error or address error occurs during reset exception processing, a double bus fault occurs, the processor halts, and the $\overline{\text{HALT}}$ signal is asserted to indicate the halted condition.

Execution of the RESET instruction does not cause a reset exception nor does it affect any internal CPU register. The SIM49 registers and the module control register in each internal peripheral module (DMA, quad data memory module (QDMM), and serial modules) are not affected. All other internal peripheral module registers are reset the same as for a hardware reset. The external devices connected to the $\overline{\text{RESET}}$ signal are reset at the completion of the RESET instruction.

5.6.2.2 BUS ERROR. A bus error exception occurs when an assertion of the $\overline{\text{BERR}}$ signal is acknowledged. The $\overline{\text{BERR}}$ signal can be asserted by one of three sources:

1. External logic by assertion of the $\overline{\text{BERR}}$ input pin
2. Direct assertion of the internal $\overline{\text{BERR}}$ signal by an internal module
3. Direct assertion of the internal $\overline{\text{BERR}}$ signal by the on-chip hardware watchdog after detecting a no-response condition

Bus error exception processing begins when the processor attempts to use information from an aborted bus cycle.

When the aborted bus cycle is an instruction prefetch, the processor will not initiate exception processing unless the prefetched information is used. For example, if a branch instruction flushes an aborted prefetch, that word is not accessed, and no exception occurs.

When the aborted bus cycle is a data access, the processor initiates exception processing immediately, except in the case of released operand writes. Released write bus errors are delayed until the next instruction boundary or until another operand access is attempted.

Exception processing for bus error exceptions follows the regular sequence, but context preservation is more involved than for other exceptions because a bus exception can be initiated while an instruction is executing. Several bus error stack format organizations are utilized to provide additional information regarding the nature of the fault.

First, any register altered by a faulted-instruction EA calculation is restored to its initial value. Then an SSW is placed on the stack. The SSW contains specific information about the aborted access—size, type of access (read or write), bus cycle type, and function code. Finally, fault address, bus error exception vector number, PC value, and a copy of the SR are saved.

If a bus error occurs during exception processing for a bus error, an address error, a reset, or while the processor is loading stack information during RTE execution, the processor halts. This simplifies isolation of catastrophic system failure by preventing processor interaction with stacks and memory. Only assertion of $\overline{\text{RESET}}$ can restart a halted processor.

5.6.2.3 ADDRESS ERROR. Address error exceptions occur when the processor attempts to access an instruction at an odd address. The effect is much the same as an internally generated bus error. The exception processing sequence is the same as that for bus error, except that the vector number refers to the address error exception vector.

Address error exception processing begins when the processor attempts to use information from the aborted bus cycle.

An address exception on a branch to an odd address is delayed until the PC is changed. No exception occurs if the branch is not taken. In this case, the fault address and return PC value placed in the exception stack frame are the odd address, and the current instruction PC points to the instruction that caused the exception.

If an address error occurs during exception processing for a bus error, another address error, or a reset, the processor halts.

5.6.2.4 INSTRUCTION TRAPS. Traps are exceptions caused by instructions. They arise from either processor recognition of abnormal conditions during instruction execution or from use of specific trapping instructions. Traps are generally used to handle abnormal conditions that arise in control routines.

The TRAP instruction, which always forces an exception, is useful for implementing system calls for user programs. The TRAPcc, TRAPV, CHK, and CHK2 instructions force exceptions when a program detects a run-time error. The DIVS and DIVU instructions force an exception if a division operation is attempted with a divisor of zero.

Exception processing for traps follows the regular sequence. If tracing is enabled when an instruction that causes a trap begins execution, a trace exception will be generated by the instruction, but the trap handler routine will not be traced. (The trap exception will be processed first, then the trace exception.)

The vector number for the TRAP instruction is internally generated—part of the number comes from the instruction itself. The trap vector number, PC value, and a copy of the SR are saved on the supervisor stack. The saved PC value is the address of the instruction that follows the instruction that generated the trap. For all instruction traps other than TRAP, a pointer to the instruction causing the trap is also saved in the fifth and sixth words of the exception stack frame.

5.6.2.5 SOFTWARE BREAKPOINTS. To support hardware emulation, the CPU32+ must provide a means of inserting breakpoints into target code and of announcing when a breakpoint is reached.

The MC68000 and MC68008 can detect an illegal instruction inserted at a breakpoint when the processor fetches from the illegal instruction exception vector location. Since the VBR on the CPU32+ allows relocation of exception vectors, the exception vector address is not a reliable indication of a breakpoint. CPU32+ breakpoint support is provided by extending the function of a set of illegal instructions ($4848–$484F).

When a breakpoint instruction is executed, the CPU32+ performs a read from CPU space $0, at a location corresponding to the breakpoint number. If this bus cycle is terminated by $\overline{\text{BERR}}$, the processor performs illegal instruction exception processing. If the bus cycle is terminated by $\overline{\text{DSACKx}}$, the processor uses the data returned to replace the breakpoint in the instruction pipeline and begins execution of that instruction. See **Section 3 Bus Operation** for a description of CPU space operations.

5.6.2.6 HARDWARE BREAKPOINTS. The CPU32+ recognizes hardware breakpoint requests. Hardware breakpoint requests do not force immediate exception processing, but are left pending. An instruction breakpoint is not made pending until the instruction corresponding to the request is executed.

A pending breakpoint can be acknowledged between instructions or at the end of exception processing. To acknowledge a breakpoint, the CPU performs a read from CPU space $0 at location $1E (see **Section 3 Bus Operation)**.

If the bus cycle terminates normally, instruction execution continues with the next instruction as if no breakpoint request occurred. If the bus cycle is terminated by $\overline{\text{BERR}}$, the CPU begins exception processing. Data returned during this bus cycle is ignored.

Exception processing follows the regular sequence. Vector number 12 (offset $30) is internally generated. The PC of the executing instruction, the PC of the next instruction to be executed, and a copy of the SR are saved on the supervisor stack.

5.6.2.7 FORMAT ERROR. The processor checks certain data values for control operations. The validity of the stack format code and, in the case of a bus cycle fault format, the version number of the processor that generated the frame are checked during execution of the RTE instruction. This check ensures that the program does not make erroneous assumptions about information in the stack frame.

If the format of the control data is improper, the processor generates a format error exception. This exception saves a four-word format exception frame and then vectors through vector table entry number 14. The stacked PC is the address of the RTE instruction that discovered the format error.

5.6.2.8 ILLEGAL OR UNIMPLEMENTED INSTRUCTIONS. An instruction is illegal if it contains a word bit pattern that does not correspond to the bit pattern of the first word of a legal CPU32+ instruction, if it is a MOVEC instruction that contains an undefined register specification field in the first extension word, or if it contains an indexed addressing mode extension word with bits 5–4 = 00 or bits 3–0 ≠ 0000.

If an illegal instruction is fetched during instruction execution, an illegal instruction exception occurs. This facility allows the operating system to detect program errors or to emulate instructions in software.

Word patterns with bits 15–12 = 1010 (referred to as A-line opcodes) are unimplemented instructions. A separate exception vector (vector 10, offset $28) is given to unimplemented instructions to permit efficient emulation.

Word patterns with bits 15–12 = 1111 (referred to as F-line opcodes) are used for M68000 family instruction set extensions. They can generate an unimplemented instruction exception caused by the first extension word of the instruction or by the addressing mode extension word. A separate F-line emulation vector (vector 11, offset $2C) is used for the exception vector.

All unimplemented instructions are reserved for use by Motorola for enhancements and extensions to the basic M68000 architecture. Opcode pattern $4AFC is defined to be illegal on all M68000 family members. Those customers requiring the use of an unimplemented opcode for synthesis of "custom instructions," operating system calls, etc., should use this opcode.

Exception processing for illegal and unimplemented instructions is similar to that for traps. The instruction is fetched and decoding is attempted. When the processor determines that execution of an illegal instruction is being attempted, exception processing begins. No registers are altered.

Exception processing follows the regular sequence. The vector number is generated to refer to the illegal instruction vector or in the case of an unimplemented instruction, to the corresponding emulation vector. The illegal instruction vector number, current PC, and a copy of the SR are saved on the supervisor stack, with the saved value of the PC being the address of the illegal or unimplemented instruction.

5.6.2.9 PRIVILEGE VIOLATIONS. To provide system security, certain instructions can be executed only at the supervisor access level. An attempt to execute one of these instructions at the user level will cause an exception. The privileged instructions are as follows:

- AND Immediate to SR
- EOR Immediate to SR
- LPSTOP
- MOVE from SR
- MOVE to SR
- MOVE USP
- MOVEC
- MOVES
- OR Immediate to SR
- RESET
- RTE
- STOP

Exception processing for privilege violations is nearly identical to that for illegal instructions. The instruction is fetched and decoded. If the processor determines that a privilege violation has occurred, exception processing begins before instruction execution.

Exception processing follows the regular sequence. The vector number (8) is generated to reference the privilege violation vector. Privilege violation vector offset, current PC, and SR are saved on the supervisor stack. The saved PC value is the address of the first word of the instruction causing the privilege violation.

5.6.2.10 TRACING. To aid in program development, M68000 processors include a facility to allow tracing of instruction execution. CPU32+ tracing also has the ability to trap on changes in program flow. In trace mode, a trace exception is generated after each instruction executes, allowing a debugging program to monitor the execution of a program under test. The T1 and T0 bits in the supervisor portion of the SR are used to control tracing (see Table 5-7).

Table 5-7. Tracing Control

T1	T0	Tracing Function
0	0	No tracing
0	1	Trace on change of flow
1	0	Trace on instruction execution
1	1	Undefined; reserved

When T1, T0 = 00, tracing is disabled, and instruction execution proceeds normally.

When T1, T0 = 01 at the beginning of instruction execution, a trace exception will be generated if the PC changes sequence during execution. All branches, jumps, subroutine calls, returns, and SR manipulations can be traced in this way. No exception occurs if a branch is not taken.

When T1, T0 = 10 at the beginning of instruction execution, a trace exception will be generated when execution is complete. If the instruction is not executed, either because an interrupt is taken or because the instruction is illegal, unimplemented, or privileged, an exception is not generated.

At the present time, T1, T0 = 11 is an undefined condition. It is reserved by Motorola for future use.

Exception processing for trace starts at the end of normal processing for the traced instruction and before the start of the next instruction. Exception processing follows the regular sequence; tracing is disabled so that the trace exception itself is not traced. A vector number is generated to reference the trace exception vector. The address of the instruction that caused the trace exception, the trace exception vector offset, the current PC, and a copy of the SR are saved on the supervisor stack. The saved value of the PC is the address of the next instruction to be executed.

A trace exception can be viewed as an extension to the function of any instruction. If a trace exception is generated by an instruction, the execution of that instruction is not complete until the trace exception processing associated with it is also complete.

If an instruction is aborted by a bus error or address error exception, trace exception processing is deferred until the suspended instruction is restarted and completed normally. An RTE from a bus error or address error will not be traced because of the possibility of continuing the instruction from the fault.

If an instruction is executed and an interrupt is pending on completion, the trace exception is processed before the interrupt exception.

If an instruction forces an exception, the forced exception is processed before the trace exception.

If an instruction is executed and a breakpoint is pending upon completion of the instruction, the trace exception is processed before the breakpoint.

If an attempt is made to execute an illegal, unimplemented, or privileged instruction while tracing is enabled, no trace exception will occur because the instruction is not executed. This is particularly important to an emulation routine that performs an instruction function, adjusts the stacked PC to beyond the unimplemented instruction, and then returns. The SR on the stack must be checked to determine if tracing is on before the return is executed. If tracing is on, trace exception processing must be emulated so that the trace exception handler can account for the emulated instruction.

Tracing also affects normal operation of the STOP and LPSTOP instructions. If either instruction begins execution with T1 set, a trace exception will be taken after the instruction loads the SR. Upon return from the trace handler routine, execution will continue with the instruction following STOP (LPSTOP), and the processor will not enter the stopped condition.

5.6.2.11 INTERRUPTS. There are seven levels of interrupt priority and 192 assignable interrupt vectors within each exception vector table. Careful use of multiple vector tables and hardware chaining will permit a virtually unlimited number of peripherals to interrupt the processor.

Interrupt recognition and subsequent processing are based on internal interrupt request signals ($\overline{IRQ7}$–$\overline{IRQ1}$) and the current priority set in SR priority mask I2–I0. Interrupt request level 0 ($\overline{IRQ7}$–$\overline{IRQ1}$ negated) indicates that no service is requested. When an interrupt of level 1 through 6 is requested via $\overline{IRQ6}$–$\overline{IRQ1}$, the processor compares the request level with the interrupt mask to determine whether the interrupt should be processed. Interrupt requests are inhibited for all priority levels less than or equal to the current priority. Level 7 interrupts are nonmaskable.

$\overline{IRQ7}$–$\overline{IRQ1}$ are synchronized and debounced by input circuitry on consecutive rising edges of the processor clock. To be valid, an interrupt request must be held constant for at least two consecutive clock periods.

Interrupt requests do not force immediate exception processing, but are left pending. A pending interrupt is detected between instructions or at the end of exception processing—all interrupt requests must be held asserted until they are acknowledged by the CPU. If the priority of the interrupt is greater than the current priority level, exception processing begins.

Exception processing occurs as follows. First, the processor makes an internal copy of the SR. After the copy is made, the processor state bits in the SR are changed—the S-bit is set, establishing supervisor access level, and bits T1 and T0 are cleared, disabling tracing. Priority level is then set to the level of the interrupt, and the processor fetches a vector number from the interrupting device (CPU space $F). The fetch bus cycle is classified as an interrupt acknowledge, and the encoded level number of the interrupt is placed on the address bus.

If an interrupting device requests automatic vectoring, the processor generates a vector number (25 to 31) determined by the interrupt level number.

If the response to the interrupt acknowledge bus cycle is a bus error, the interrupt is taken to be spurious, and the spurious interrupt vector number (24) is generated.

The exception vector number, PC, and SR are saved on the supervisor stack. The saved value of the PC is the address of the instruction that would have executed if the interrupt had not occurred.

Priority level 7 interrupt is a special case. Level 7 interrupts are nonmaskable interrupts (NMI). Level 7 requests are transition sensitive to eliminate redundant servicing and resultant stack overflow. Transition sensitive means that the level 7 input must change state before the CPU will detect an interrupt.

An NMI is generated each time the interrupt request level changes to level 7 (regardless of priority mask value), and each time the priority mask changes from 7 to a lower number while the request level remains at 7.

Many M68000 peripherals provide for programmable interrupt vector numbers to be used in the system interrupt request/acknowledge mechanism. If the vector number is not initialized after reset and if the peripheral must acknowledge an interrupt request, the peripheral should return the uninitialized interrupt vector number (15).

See **Section 3 Bus Operation** for detailed information on interrupt acknowledge cycles.

5.6.2.12 RETURN FROM EXCEPTION. When exception stacking operations for all pending exceptions are complete, the processor begins execution of the handler for the last exception processed. After the exception handler has executed, the processor must restore the system context in existence prior to the exception. The RTE instruction is designed to accomplish this task.

When RTE is executed, the processor examines the stack frame on top of the supervisor stack to determine if it is valid and determines what type of context restoration must be performed. See **5.6.4 CPU32+ Stack Frames** for a description of stack frames.

For a normal four-word frame, the processor updates the SR and PC with data pulled from the stack, increments the SSP by 8, and resumes normal instruction execution. For a six-word frame, the SR and PC are updated from the stack, the active SSP is incremented by 12, and normal instruction execution resumes.

For a bus fault frame, the format value on the stack is first checked for validity. In addition, the version number on the stack must match the version number of the processor that is attempting to read the stack frame. The version number is located in the most significant byte (bits 15–8) of the internal register word at location SP + $14 in the stack frame. The validity check ensures that stack frame data will be properly interpreted in multiprocessor systems.

If a frame is invalid, a format error exception is taken. If it is inaccessible, a bus error exception is taken. Otherwise, the processor reads the entire frame into the proper internal registers, de-allocates the stack (12 words), and resumes normal processing. Bus error frames for faults during exception processing require the RTE instruction to rewrite the faulted stack frame. If an error occurs during any of the bus cycles required by rewrite, the processor halts.

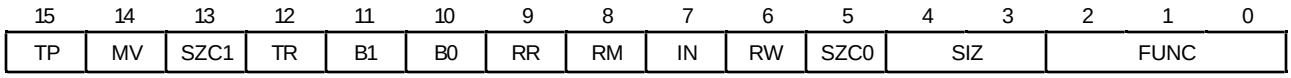
If a format error occurs during RTE execution, the processor creates a normal four-word fault stack frame below the frame that it was attempting to use. If a bus error occurs, a bus-error stack frame will be created. The fault stack frame remains intact, so that it may

be examined and repaired by an exception handler or used by a different type of processor (e.g., MC68010, MC68020, or future M68000 processor) in a multiprocessor system.

5.6.3 Fault Recovery

There are four phases of recovery from a fault: recognizing the fault, saving the processor state, repairing the fault (if possible), and restoring the processor state. Saving and restoring the processor state are described in the following paragraphs.

The stack contents are identified by the special status word (SSW). In addition to identifying the fault type represented by the stack frame, the SSW contains the internal processor state corresponding to the fault.



TP—BERR Frame Type

The TP field defines the class of the faulted bus operation. Two bus error exception frame types are defined. One is for faults on prefetch and operand accesses, and the other is for faults during exception frame stacking.

0 = Operand or prefetch bus fault
1 = Exception processing bus fault

MV—MOVEM in Progress

MV is set when the operand transfer portion of the MOVEM instruction is in progress at the time of a bus fault. If a prefetch bus fault occurs while prefetching the MOVEM opcode and extension word, both the MV and IN bits will be set.

0 = MOVEM was not in progress when fault occurred
1 = MOVEM was in progress when fault occurred

SZC1,SZC0—Original Operand Size

The SZC1,SZC0 field specifies the size of the original bus cycle (i.e., the size bits of the first cycle, when a transaction is divided into two or three cycles due to bus size or operand address).

00 = Original operand size was long word
01 = Original operand size was byte
10 = Original operand size was word
11 = Unused, reserved

TR—Trace Pending

TR indicates that a trace exception was pending when a bus error exception was processed. The instruction that generated the trace will not be restarted upon return from the exception handler. This includes MOVEM and released write bus errors indicated by the assertion of either MV or RR in the SSW.

0 = Trace not pending
1 = Trace pending

B1—Breakpoint Channel 1 Pending

B1 indicates that a breakpoint exception was pending on channel 1 (external breakpoint source) when a bus error exception was processed. Pending breakpoint status is stacked, regardless of the type of bus error exception.

0 = Breakpoint not pending
1 = Breakpoint pending

B0—Breakpoint Channel 0 Pending

B0 indicates that a breakpoint exception was pending on channel 0 (internal breakpoint source) when the bus error exception was processed. Pending breakpoint status is stacked, regardless of the type of bus error exception.

0 = Breakpoint not pending
1 = Breakpoint pending

RR—Rerun Write Cycle after RTE

RR will be set if the faulted bus cycle was a released write. A released write is one that is overlapped. If the write is completed (rerun) in the exception handler, the RR bit should be cleared before executing RTE. The bus cycle will be rerun if the RR bit is set upon return from the exception handler.

0 = Faulted cycle was read, RMW, or unreleased write
1 = Faulted cycle was a released write

RM—Faulted Cycle Was Read-Modify-Write

Faulted RMW bus cycles set the RM bit. RM is ignored during unstacking.

0 = Faulted cycle was non-RMW cycle
1 = Faulted cycle was either the read or write of an RMW cycle

IN—Instruction/Other

Instruction prefetch faults are distinguished from operand (both read and write) faults by the IN bit. If IN is cleared, the error was on an operand cycle; if IN is set, the error was on an instruction prefetch. IN is ignored during unstacking.

0 = Operand
1 = Prefetch

RW—Read/Write of Faulted Bus Cycle

Read and write bus cycles are distinguished by the RW bit. Read bus cycles will set this bit, and write bus cycles will clear it. RW is reloaded into the bus controller if the RR bit is set during unstacking.

0 = Faulted cycle was an operand write
1 = Faulted cycle was a prefetch or operand read

SIZ—Remaining Size of Faulted Bus Cycle

The SIZ field shows operand size remaining when a fault was detected. This field does not indicate the initial size of the operand, nor does it necessarily indicate the proper status of a dynamically sized bus cycle. Dynamic sizing occurs on the external bus and is transparent to the CPU. Byte size is shown only when the original operand was a byte. The field is reloaded into the bus controller if the RR bit is set during unstacking. The SIZ field is encoded as follows:

00 = Long word
01 = Byte
10 = Word
11 = Three byte

FUNC—Function Code of Faulted Bus Cycle

The function code for the faulted cycle is stacked in the FUNC field of the SSW, which is a copy of FC2–FC0 for the faulted bus cycle. This field is reloaded into the bus controller if the RR bit is set during unstacking. All unused bits are stacked as zeros and are ignored during unstacking. Further discussion of the SSW is included in **5.6.3.1 Types of Faults**.

5.6.3.1 TYPES OF FAULTS. An efficient implementation of instruction restart dictates that faults on some bus cycles be treated differently than faults on other bus cycles. The CPU32+ defines four fault types: released write faults, faults during exception processing, faults during MOVEM operand transfer, and faults on any other bus cycle.

5.6.3.1.1 Type I—Released Write Faults. CPU32+ instruction pipelining can cause a final instruction write to overlap the execution of a following instruction. A write that is overlapped is called a released write. A released write fault occurs when a bus error or some other fault occurs on the released write.

Released write faults are taken at the next instruction boundary. The stacked PC is that of the next unexecuted instruction. If a subsequent instruction attempts an operand access while a released write fault is pending, the instruction is aborted and the write fault is acknowledged. This action prevents the instruction from using stale data.

The SSW for a released write fault contains the following bit pattern:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	0
0	0	SZC1	TR	B1	B0	1	0	0	0	SZC0	SIZ		FUNC	

TR , B1, and B0 are set if the corresponding exception is pending when the bus error exception is taken. Status regarding the faulted bus cycle is reflected in the SZCx, SIZ, and FUNC fields.

The remainder of the stack contains the PC of the next unexecuted instruction, the current SR, the address of the faulted memory location, and the contents of the data buffer that was to be written to memory. This data is written on the stack in the format depicted in Figure 5-15. When a released write fault exception handler executes, the machine will complete the faulted write and then continue executing instructions wherever the PC indicates.

5.6.3.1.2 Type II—Prefetch, Operand, RMW, and MOVEP Faults. The majority of bus error exceptions are included in this category—all instruction prefetches, all operand reads, all RMW cycles, and all operand accesses resulting from execution of MOVEP (except the last write of a MOVEP Rn,⟨ea⟩ or the last write of MOVEM, which are type I faults). The TAS, MOVEP, and MOVEM instructions account for all operand writes not considered released write faults.

All type II faults cause an immediate exception that aborts the current instruction. Any registers that were altered as the result of an EA calculation (i.e., postincrement or predecrement) are restored prior to processing the bus cycle fault.

The SSW for faults in this category contains the following bit pattern:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	0
0	0	SZC1	0	B1	B0	0	RM	IN	RW	SZC0	SIZ		FUNC	

The trace pending bit is always cleared since the instruction will be restarted upon return from the handler. Saving a pending exception on the stack causes a trace exception to be taken prior to restarting the instruction. If the exception handler does not alter the stacked SR trace bits, the trace is requeued when the instruction is started.

The breakpoint pending bits are stacked in the SSW, even though the instruction is restarted upon return from the handler. This avoids problems with bus state analyzer equipment that has been programmed to breakpoint only the first access to a specific location or to count accesses to that location. If this response is not desired, the exception handler can clear the bits before return. The RM, IN, RW, SZCx, FUNC, and SIZ fields reflect the type of bus cycle that caused the fault. If the bus cycle was an RMW, the RM bit will be set, and the RW bit will show whether the fault was on a read or write.

5.6.3.1.3 Type III—Faults During MOVEM Operand Transfer. Bus faults that occur as a result of MOVEM operand transfer are classified as type III faults. MOVEM instruction prefetch faults are type II faults.

Type III faults cause an immediate exception that aborts the current instruction. Registers altered during execution of the faulted instruction are not restored prior to execution of the fault handler. This includes any register predecremented as a result of the effective address calculation or any register overwritten during instruction execution. Since postincremented registers are not updated until the end of an instruction, the register retains its pre-instruction value unless overwritten by operand movement.

The SSW for faults in this category contains the following bit pattern:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	0
0	1	SZC1	TR	B1	B0	RR	0	IN	RW	SZC0	SIZ		FUNC	

MV is set, indicating that MOVEM should be continued from the point where the fault occurred upon return from the exception handler. TR, B1, and B0 are set if a corresponding exception is pending when the bus error exception is taken. IN is set if a bus fault occurs while prefetching an opcode or an extension word during instruction restart. RW, SZCx, SIZ, and FUNC all reflect the type of bus cycle that caused the fault. All write faults have the RR bit set to indicate that the write should be rerun upon return from the exception handler.

The remainder of the stack frame contains sufficient information to continue MOVEM with operand transfer following a faulted transfer. The address of the next operand to be transferred, incremented or decremented by operand size, is stored in the faulted address location ($08). The stacked transfer counter is set to 16 minus the number of transfers attempted (including the faulted cycle). Refer to Figure 5-12 for the stacking format.

5.6.3.1.4 Type IV—Faults During Exception Processing. The fourth type of fault occurs during exception processing. If this exception is a second address or bus error, the machine halts in the double bus fault condition. However, if the exception is one that causes a four- or six-word stack frame to be written, a bus cycle fault frame is written below the faulted exception stack frame.

The SSW for a fault within an exception contains the following bit pattern:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	0
1	0	SZC1	TR	B1	B0	0	0	0	1	SZC0	SIZ		FUNC	

TR, B1, and B0 are set if a corresponding exception is pending when the bus error exception is taken.

The contents of the faulted exception stack frame are included in the bus fault stack frame. The pre-exception SR and the format/vector word of the faulted frame are stacked. The type of exception can be determined from the format/vector word. If the faulted

exception stack frame contains six words, the PC of the instruction that caused the initial exception is also stacked. This data is placed on the stack in the format shown in Figure 5-13. The return address from the initial exception is stacked for RTE .

5.6.3.2 CORRECTING A FAULT. There are two ways to complete a faulted released write bus cycle. The first is to use a software handler. The second is to rerun the bus cycle via RTE.

Type II fault handlers must terminate with RTE, but specific requirements must also be met before an instruction is restarted.

There are three varieties of type III operand fault recovery. The first is completion of an instruction in software. The second is conversion to type II with restart via RTE. The third is continuation from the fault via RTE.

5.6.3.2.1 Type I—Completing Released Writes via Software. To complete a bus cycle in software, a handler must first read the SSW function code field to determine the appropriate address space, access the fault address pointer on the stack, and then transfer data from the stacked image of the output buffer to the fault address.

If the CPU32+ is configured to 16-bit operation, rather than 32-bit operation, on the internal data bus, long operands require two bus accesses. A fault during the second access of a long operand causes the SZCx bits in the SSW to be set to long word. The SIZ field indicates remaining operand size. If operand coherency is important, the complete operand must be rewritten. After a long operand is rewritten, the RR bit must be cleared. Failure to clear the RR bit can cause the RTE instruction to rerun the bus cycle. Following rewrite, it is not necessary to adjust the PC (or other stack contents) before executing RTE.

5.6.3.2.2 Type I—Completing Released Writes via RTE. An exception handler can use the RTE instruction to complete a faulted bus cycle. When RTE executes, the fault address, data output buffer, PC, and SR are restored from the stack. Any pending breakpoint or trace exceptions, as indicated by TR, B1, and B0 in the stacked SSW, are requeued during SSW restoration. The RR bit in the SSW is checked during the unstacking operation; if it is set, the RW, FUNC, and SIZ fields are restored and the released write cycle is rerun.

To maintain long-word operand coherence, stack contents must be adjusted prior to the RTE execution. The fault address must be decremented by 2 if the SZCx bits are set to long word and SIZ indicates a remaining byte or word. SIZ must be set to long. All other fields should be left unchanged. The bus controller uses the modified fault address and SIZ field to rerun the complete released write cycle.

Manipulating the stacked SSW can cause unpredictable results because RTE checks only the RR bit to determine if a bus cycle must be rerun. Inadvertent alteration of the control bits could cause the bus cycle to be a read instead of a write or could cause access to a different address space than the original bus cycle. If the rerun bus cycle is a read, returned data will be ignored.

5.6.3.2.3 Type II—Correcting Faults via RTE. Instructions aborted because of a type II fault are restarted upon return from the exception handler. A fault handler must establish safe restart conditions. If a fault is caused by a nonresident page in a demand-paged virtual memory configuration, the fault address must be read from the stack, and the appropriate page retrieved. An RTE instruction terminates the exception handler. After unstacking the machine state, the instruction is refetched and restarted.

5.6.3.2.4 Type III—Correcting Faults via Software. Sufficient information is contained in the stack frame to complete MOVEM in software. After the cause of the fault is corrected, the faulted bus cycle must be rerun. Perform the following procedures to complete an instruction through software:

A. Set Up for Rerun

1. Read the MOVEM opcode and extension from locations pointed to by stack frame PC and PC + 2. The EA need not be recalculated since the next operand address is saved in the stack frame. However, the opcode EA field must be examined to determine how to update the address register and PC when the instruction is complete.
2. Adjust the mask to account for operands already transferred. Subtract the stacked operand transfer count from 16 to obtain the number of operands transferred. Scan the mask using this count value. Each time a set bit is found, clear it and decrement the counter. When the count is zero, the mask is ready for use.
3. Adjust the operand address. If the predecrement addressing mode is in effect, subtract the operand size from the stacked value; otherwise, add the operand size to the stacked value.

B. Rerun Instruction

1. Scan the mask for set bits. Read/write the selected register from/to the operand address as each bit is found.
2. As each operand is transferred, clear the mask bit and increment (decrement) the operand address. When all bits in the mask are cleared, all operands have been transferred.
3. If the addressing mode is predecrement or postincrement, update the register to complete the execution of the instruction.
4. If TR is set in the stacked SSW, create a six-word stack frame and execute the trace handler. If either B1 or B0 is set in the SSW, create another six-word stack frame and execute the hardware breakpoint handler.
5. De-allocate the stack and return control to the faulted program.

5.6.3.2.5 Type III—Correcting Faults by Conversion and Restart. In some situations it may be necessary to rerun all the operand transfers for a faulted instruction rather than continue from a faulted operand. Clearing the MV bit in the stacked SSW converts a type III fault into a type II fault. Consequently, MOVEM, like all other type II exceptions, will be restarted upon return from the exception handler. When a fault occurs after an operand has transferred, that transfer is not "undone". However, these memory locations are

accessed a second time when the instruction is restarted. If a register used in an EA calculation is overwritten before a fault occurs, an incorrect EA is calculated upon instruction restart.

5.6.3.2.6 Type III—Correcting Faults via RTE. The preferred method of MOVEM bus fault recovery is to correct the cause of the fault and then execute an RTE instruction without altering the stack contents.

The RTE recognizes that MOVEM was in progress when a fault occurred, restores the appropriate machine state, refetches the instruction, repeats the faulted transfer, and continues the instruction.

MOVEM is the only instruction continued upon return from an exception handler. Although the instruction is refetched, the EA is not recalculated, and the mask is rescanned the same number of times as before the fault. Modifying the code prior to RTE can cause unexpected results.

5.6.3.2.7 Type IV—Correcting Faults via Software. Bus error exceptions can occur during exception processing while the processor is fetching an exception vector or while it is stacking. The same stack frame and SSW are used in both cases, but each has a distinct fault address. The stacked faulted exception format/vector word identifies the type of faulted exception and the contents of the remainder of the frame. A fault address corresponding to the vector specified in the stacked format/vector word indicates that the processor could not obtain the address of the exception handler.

A bus error exception handler should execute RTE after correcting a fault. RTE restores the internal machine state, fetches the address of the original exception handler, recreates the original exception stack frame, and resumes execution at the exception handler address.

If the fault is intractable, the exception handler should rewrite the faulted exception stack frame at SP + $14 + $06 and then jump directly to the original exception handler. The stack frame can be generated from the information in the bus error frame: the pre-exception SR (SP + $0C), the format/vector word (SP + $0E), and, if the frame being written is a six-word frame, the PC of the instruction causing the exception (SP + $10). The return PC value is available at SP + $02.

A stacked fault address equal to the current SP may indicate that, although the first exception received a bus error while stacking, the bus error exception stacking successfully completed. This occurrence is extremely improbable, but the CPU32+ supports recovery from it. Once the exception handler determines that the fault has been corrected, recovery can proceed as described previously. If the fault cannot be corrected, move the supervisor stack to another area of memory, copy all valid stack frames to the new stack, create a faulted exception frame on top of the stack, and resume execution at the exception handler address.

5.6.4 CPU32+ Stack Frames

The CPU32+ generates three different stack frames: four-word frames, six-word frames, and twelve-word bus error frames.

5.6.4.1 FOUR-WORD STACK FRAME. This stack frame is created by interrupt, format error, TRAP #n, illegal instruction, A-line and F-line emulator trap, and privilege violation exceptions. Depending on the exception type, the PC value is either the address of the next instruction to be executed or the address of the instruction that caused the exception (see Figure 5-16).

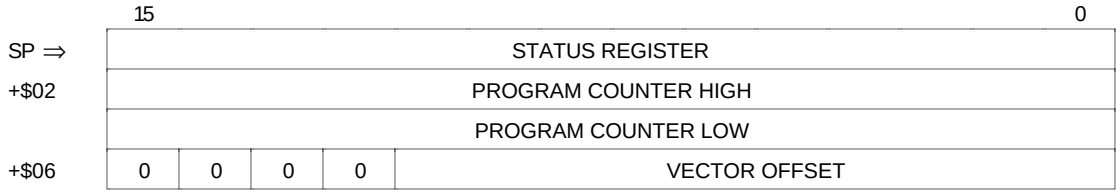


Figure 5-16. Format $0—Four-Word Stack Frame

5.6.4.2 SIX-WORD STACK FRAME. This stack frame (see Figure 5-17) is created by instruction-related traps, which include CHK, CHK2, TRAPcc, TRAPV, and divide-by-zero, and by trace exceptions. The faulted instruction PC value is the address of the instruction that caused the exception. The next PC value (the address to which RTE returns) is the address of the next instruction to be executed.

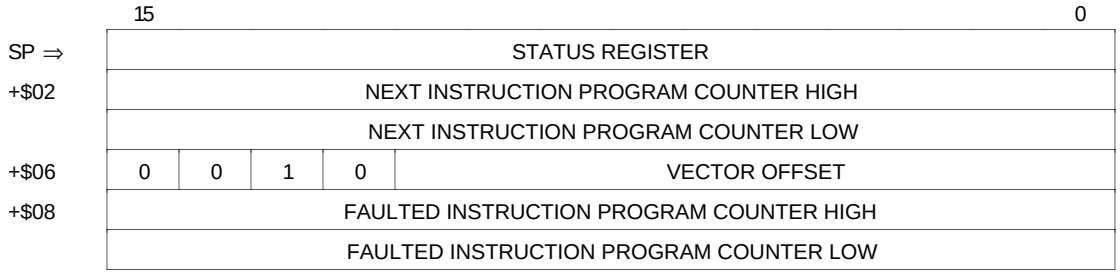


Figure 5-17. Format $2—Six-Word Stack Frame

Hardware breakpoints also utilize this format. The faulted instruction PC value is the address of the instruction executing when the breakpoint was sensed. Usually this is the address of the instruction that caused the breakpoint, but, because released writes can overlap following instructions, the faulted instruction PC may point to an instruction following the instruction that caused the breakpoint. The address to which RTE returns is the address of the next instruction to be executed.

5.6.4.3 BUS ERROR STACK FRAME. This stack frame is created when a bus cycle fault is detected. The CPU32+ bus error stack frame differs significantly from the equivalent stack frames of other M68000 family members. The only internal machine state required in the CPU32+ stack frame is the bus controller state at the time of the error and a single register.

Bus operation in progress at the time of a fault is conveyed by the SSW.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
TP	MV	SZC1	TR	B1	B0	RR	RM	IN	RW	SZC0	SIZ		FUNC		

The bus error stack frame is 12 words in length. There are three variations of the frame, each distinguished by different values in the SSW TP and MV fields.

An internal transfer count register appears at location SP + $14 in all bus error stack frames. The register contains an 8-bit microcode revision number and, for type III faults, an 8-bit transfer count. Register format is shown in Figure 5-18.

15	8	7	0
MICROCODE REVISION NUMBER		TRANSFER COUNT	

Figure 5-18. Internal Transfer Count Register

The microcode revision number is checked before a bus error stack frame is restored via RTE. In a multiprocessor system, this check ensures that a processor using stacked information is at the same revision level as the processor that created it.

The transfer count is ignored unless the MV bit in the stacked SSW is set. If the MV bit is set, the least significant byte of the internal register is reloaded into the MOVEM transfer counter during RTE execution.

For faults occurring during normal instruction execution (both prefetches and non-MOVEM operand accesses), SSW TP,MV = 00. The stack frame format is shown in Figure 5-19.

Faults that occur during the operand portion of the MOVEM instruction are identified by SSW TP,MV = 01. The stack frame format is shown in Figure 5-20.

When a bus error occurs during exception processing, SSW TP,MV = 10. The frame shown in Figure 5-21 is written below the faulting frame. Stacking begins at the address pointed to by SP – 6 (SP value is the value before initial stacking on the faulted frame).

The frame can have either four or six words, depending on the type of error. Four-word stack frames do not include the faulted instruction PC. (The internal transfer count register is located at SP + $10 and the SSW is located at SP + $12.)

The fault address of a dynamically sized bus cycle is the address of the upper byte, regardless of the byte that caused the error.

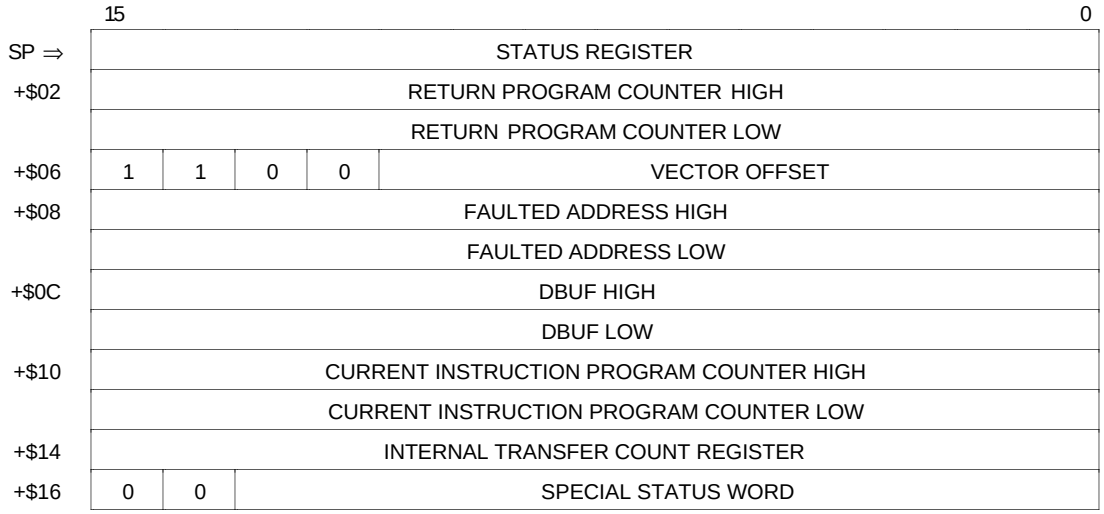


Figure 5-19. Format $C—Bus Error Stack for Prefetches and Operands

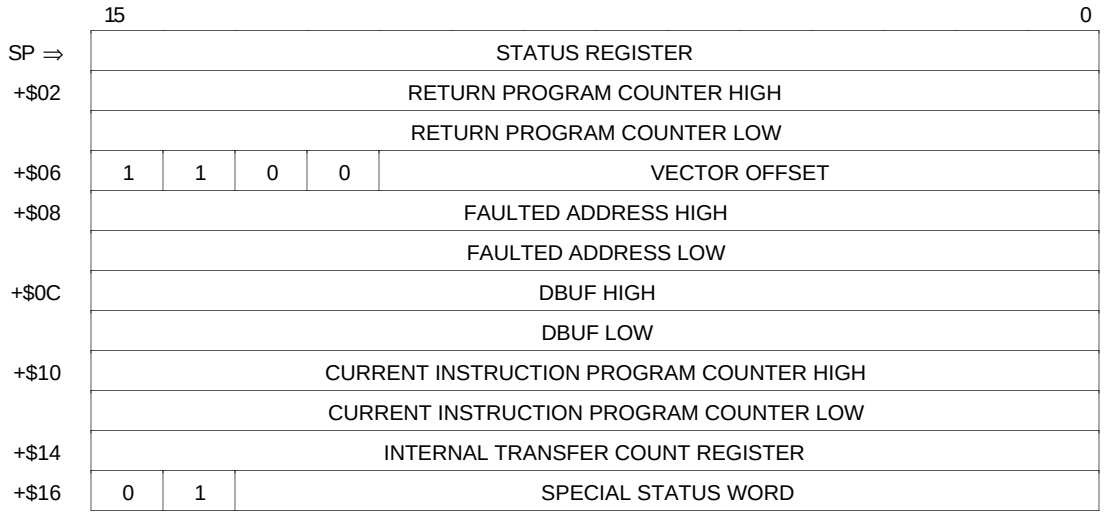


Figure 5-20. Format $C—Bus Error Stack on MOVEM Operand

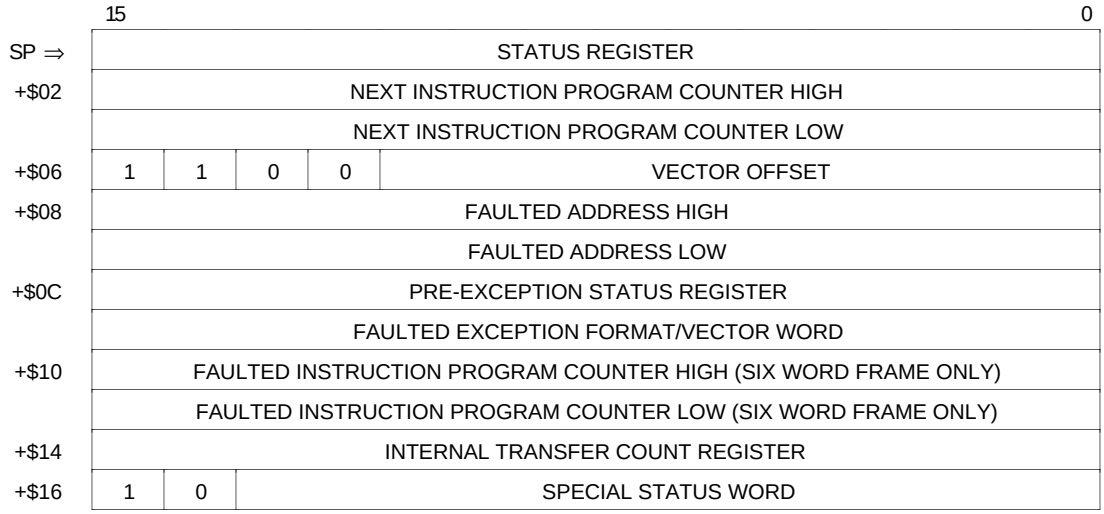


Figure 5-21. Format $C—Four- and Six-Word Bus Error Stack

5.7 DEVELOPMENT SUPPORT

All M68000 family members have the following special features that facilitate applications development.

Trace on Instruction Execution—All M68000 processors include an instruction-by-instruction tracing facility to aid in program development. The MC68020, MC68030, and CPU32+ can also trace those instructions that change program flow. In trace mode, an exception is generated after each instruction is executed, allowing a debugger program to monitor execution of a program under test. See **5.6.2.10 Tracing** for more information.

Breakpoint Instruction—An emulator can insert software breakpoints into target code to indicate when a breakpoint occurs. On the MC68010, MC68020, MC68030, and CPU32+, this function is provided via illegal instructions ($4848–$484F) that serve as breakpoint instructions. See **5.6.2.5 Software Breakpoints** for more information.

Unimplemented Instruction Emulation—When an attempt is made to execute an illegal instruction, an illegal instruction exception occurs. Unimplemented instructions (F-line, A-line) utilize separate exception vectors to permit efficient emulation of unimplemented instructions in software. See **5.6.2.8 Illegal or Unimplemented Instructions** for more information.

5.7.1 CPU32+ Integrated Development Support

In addition to standard MC68000 family capabilities, the CPU32+ has features to support advanced integrated system development. These features include background debug mode, deterministic opcode tracking, hardware breakpoints, and internal visibility in a single-chip environment.

5.7.1.1 BACKGROUND DEBUG MODE (BDM) OVERVIEW. Microprocessor systems generally provide a debugger, implemented in software, for system analysis at the lowest level. The BDM on the CPU32+ is unique because the debugger is implemented in CPU microcode.

BDM incorporates a full set of debug options—registers can be viewed and/or altered, memory can be read or written, and test features can be invoked.

A resident debugger simplifies implementation of an in-circuit emulator. In a common setup (see Figure 5-22), emulator hardware replaces the target system processor. A complex, expensive pod-and-cable interface provides a communication path between target system and emulator.

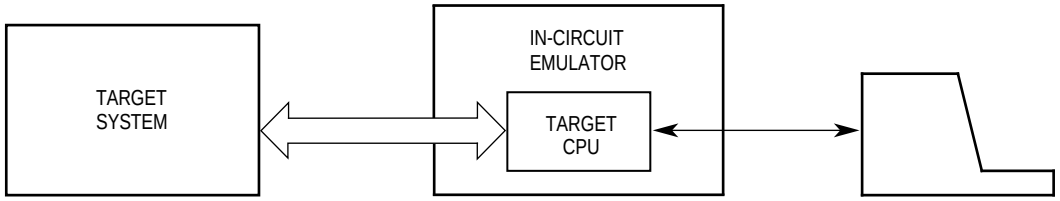


Figure 5-22. In-Circuit Emulator Configuration

By contrast, an integrated debugger supports use of a bus state analyzer (BSA) for in-circuit emulation. The processor remains in the target system (see Figure 5-23), and the interface is simplified. The BSA monitors target processor operation and the on-chip debugger controls the operating environment. Emulation is much closer to target hardware; thus, many interfacing problems (i.e., limitations on high-frequency operation, AC and DC parametric mismatches, and restrictions on cable length) are minimized.

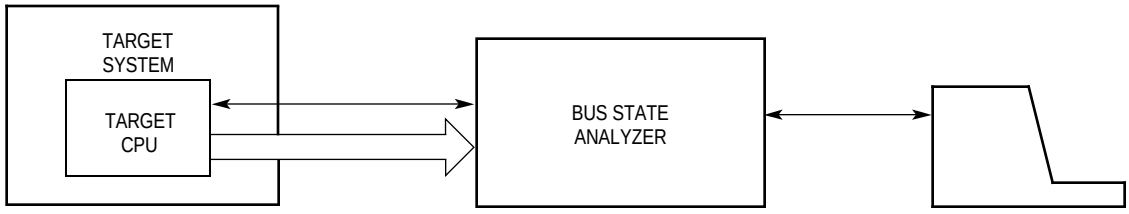


Figure 5-23. Bus State Analyzer Configuration

5.7.1.2 DETERMINISTIC OPCODE TRACKING OVERVIEW. CPU32+ function code outputs are augmented by three supplementary signals that monitor the instruction pipeline. The $\overline{\text{IFETCH}}$ output signal identifies bus cycles in which data is loaded into the pipeline and signals pipeline flushes. The $\overline{\text{IPIPE1}}$, $\overline{\text{IPIPE0}}$ output signals indicate when each mid-instruction pipeline advance occurs and when instruction execution begins. These signals allow a BSA to synchronize with instruction stream activity. Refer to **5.7.3 Deterministic Opcode Tracking** for complete information.

5.7.1.3 ON-CHIP HARDWARE BREAKPOINT OVERVIEW. An external breakpoint input and an on-chip hardware breakpoint capability permit breakpoint trap on any memory access. Off-chip address comparators will not detect breakpoints on internal accesses unless show cycles are enabled. Breakpoints on prefetched instructions, which are flushed from the pipeline before execution, are not acknowledged, but operand breakpoints are always acknowledged. Acknowledged breakpoints can initiate either exception processing or BDM. See **5.6.2.6 Hardware Breakpoints** for more information.

5.7.2 Background Debug Mode

BDM is an alternate CPU32+ operating mode. During BDM, normal instruction execution is suspended, and special microcode performs debugging functions under external control. Figure 5-24 is a BDM block diagram.

BDM can be initiated in several ways—by externally generated breakpoints, by internal peripheral breakpoints, by the background instruction (BGND), or by catastrophic exception conditions. While in BDM, the CPU32+ ceases to fetch instructions via the parallel bus and communicates with the development system via a dedicated, high-speed, SPI-type serial command interface.

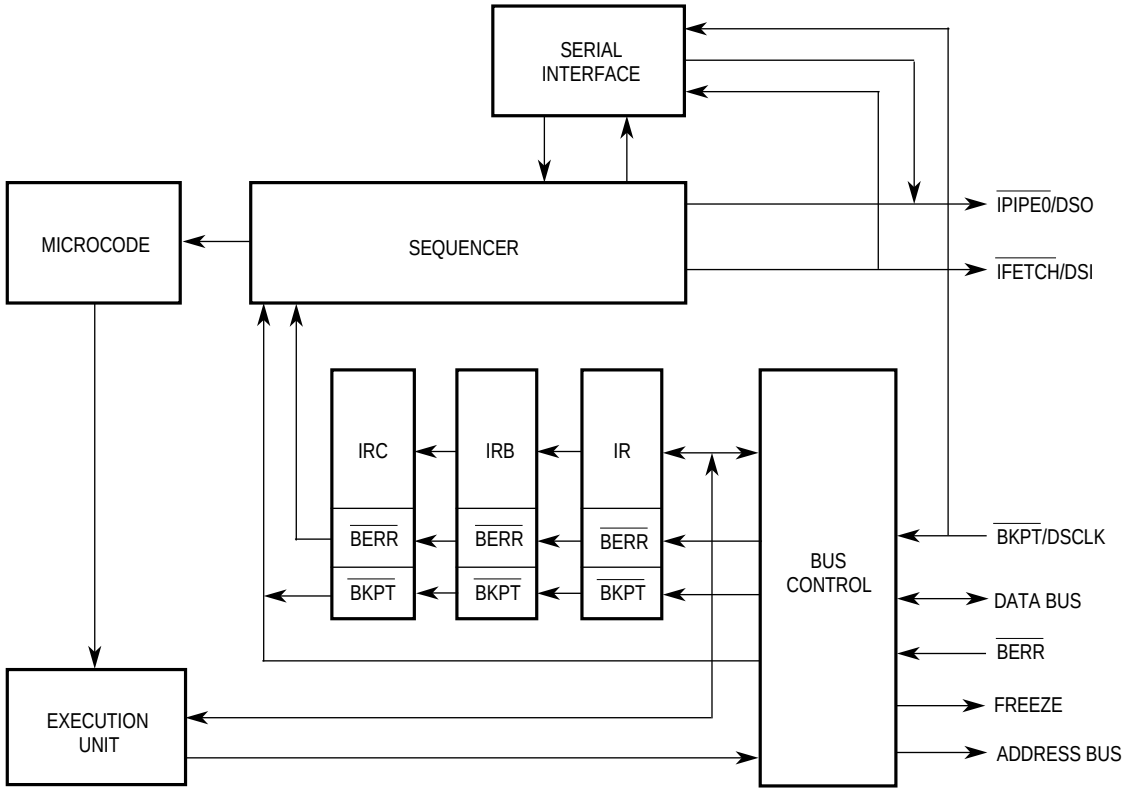


Figure 5-24. BDM Block Diagram

5.7.2.1 ENABLING BDM. Accidentally entering BDM in a nondevelopment environment could lock up the CPU32+ since the serial command interface would probably not be available. For this reason, BDM is enabled during reset via the $\overline{\text{BKPT}}$ signal.

BDM operation is enabled when $\overline{\text{BKPT}}$ is asserted (low) at the rising edge of $\overline{\text{RESET}}$. BDM remains enabled until the next system reset. A high $\overline{\text{BKPT}}$ on the trailing edge of $\overline{\text{RESET}}$ disables BDM. $\overline{\text{BKPT}}$ is relatched on each rising transition of $\overline{\text{RESET}}$. $\overline{\text{BKPT}}$ is synchronized internally and must be held low for at least two clock cycles prior to negation of $\overline{\text{RESET}}$.

BDM enable logic must be designed with special care. If hold time on $\overline{\text{BKPT}}$ (after the trailing edge of $\overline{\text{RESET}}$) extends into the first bus cycle following reset, this bus cycle could be tagged with a breakpoint. Refer to **Section 3 Bus Operation** for timing information.

5.7.2.2 BDM SOURCES. When BDM is enabled, any of several sources can cause the transition from normal mode to BDM. These sources include external $\overline{\text{BKPT}}$ hardware, the BGND instruction, a double bus fault, and internal peripheral breakpoints. If BDM is not enabled when an exception condition occurs, the exception is processed normally. Table 5-8 summarizes the processing of each source for both enabled and disabled cases. Note that the BKPT instruction never causes a transition into BDM.

Table 5-8. BDM Source Summary

Source	BDM Enabled	BDM Disabled
$\overline{\text{BKPT}}$	Background	Breakpoint Exception
Double Bus Fault	Background	Halted
BGND Instruction	Background	Illegal Instruction
BKPT Instruction	Opcode Substitution/ Illegal Instruction	Opcode Substitution/ Illegal Instruction

5.7.2.2.1 External $\overline{\text{BKPT}}$ Signal. Once enabled, BDM is initiated whenever assertion of $\overline{\text{BKPT}}$ is acknowledged. If BDM is disabled, a breakpoint exception (vector $0C) is acknowledged. The $\overline{\text{BKPT}}$ input has the same timing relationship to the data strobe trailing edge as read cycle data. There is no breakpoint acknowledge bus cycle when BDM is entered.

5.7.2.2.2 BGND Instruction. An illegal instruction, $4AFA, is reserved for use by development tools. The CPU32+ defines $4AFA (BGND) to be a BDM entry point when BDM is enabled. If BDM is disabled, an illegal instruction trap is acknowledged. Illegal instruction traps are discussed in **5.6.2.8 Illegal or Unimplemented Instructions**.

5.7.2.2.3 Double Bus Fault. The CPU32+ normally treats a double bus fault (two bus faults in succession) as a catastrophic system error and halts. When this condition occurs during initial system debug (a fault in the reset logic), further debugging is impossible until the problem is corrected. In BDM, the fault can be temporarily bypassed so that its origin can be isolated and eliminated.

5.7.2.3 ENTERING BDM. When the processor detects a $\overline{\text{BKPT}}$ or a double bus fault or decodes a BGND instruction, it suspends instruction execution and asserts the FREEZE output. FREEZE assertion is the first indication that the processor has entered BDM. Once FREEZE has been asserted, the CPU enables the serial communication hardware and awaits a command.

The CPU writes a unique value indicating the source of BDM transition into temporary register A (ATEMP) as part of the process of entering BDM. A user can poll ATEMP and determine the source (see Table 5-9) by issuing a read system register command (RSREG). ATEMP is used in most debugger commands for temporary storage—it is imperative that the RSREG command be the first command issued after transition into BDM.

Table 5-9. Polling the BDM Entry Source

Source	ATEMP 31–16	ATEMP 15–0
Double Bus Fault	SSW*	$FFFF
BGND Instruction	$0000	$0001
Hardware Breakpoint	$0000	$0000

*SSW is described in detail in **5.5.3 Fault Recovery.**

A double bus fault during initial SP/PC fetch sequence is distinguished by a value of $FFFFFFFF in the current instruction PC. At no other time will the processor write an odd value into this register.

5.7.2.4 COMMAND EXECUTION. Figure 5-25 summarizes BDM command execution. Commands consist of one 16-bit operation word and can include one or more 16-bit extension words. Each incoming word is read as it is assembled by the serial interface. The microcode routine corresponding to a command is executed as soon as the command is complete. Result operands are loaded into the output shift register to be shifted out as the next command is read. This process is repeated for each command until the CPU returns to normal operating mode.

5.7.2.5 BDM REGISTERS. BDM processing uses three special-purpose registers to track program context during development. A description of each register follows.

5.7.2.5.1 Fault Address Register (FAR). The FAR contains the address of the faulting bus cycle immediately following a bus or address error. This address remains available until overwritten by a subsequent bus cycle. Following a double bus fault, the FAR contains the address of the last bus cycle. The address of the first fault (if one occurred) is not visible to the user.

5.7.2.5.2 Return Program Counter (RPC). The RPC points to the location where fetching will commence after transition from BDM to normal mode. This register should be accessed to change the flow of a program under development. Changing the RPC to an odd value will cause an address error when normal mode prefetching begins.

5.7.2.5.3 Current Instruction Program Counter (PCC). The PCC holds a pointer to the first word of the last instruction executed prior to transition into BDM. Due to instruction pipelining, the instruction pointed to may not be the instruction that caused the transition. An example is a breakpoint on a released write. The bus cycle may overlap as many as two subsequent instructions before stalling the instruction sequencer. A $\overline{\text{BKPT}}$ asserted during this cycle will not be acknowledged until the end of the instruction executing at completion of the bus cycle. PCC will contain $00000001 if BDM is entered via a double bus fault immediately out of reset.

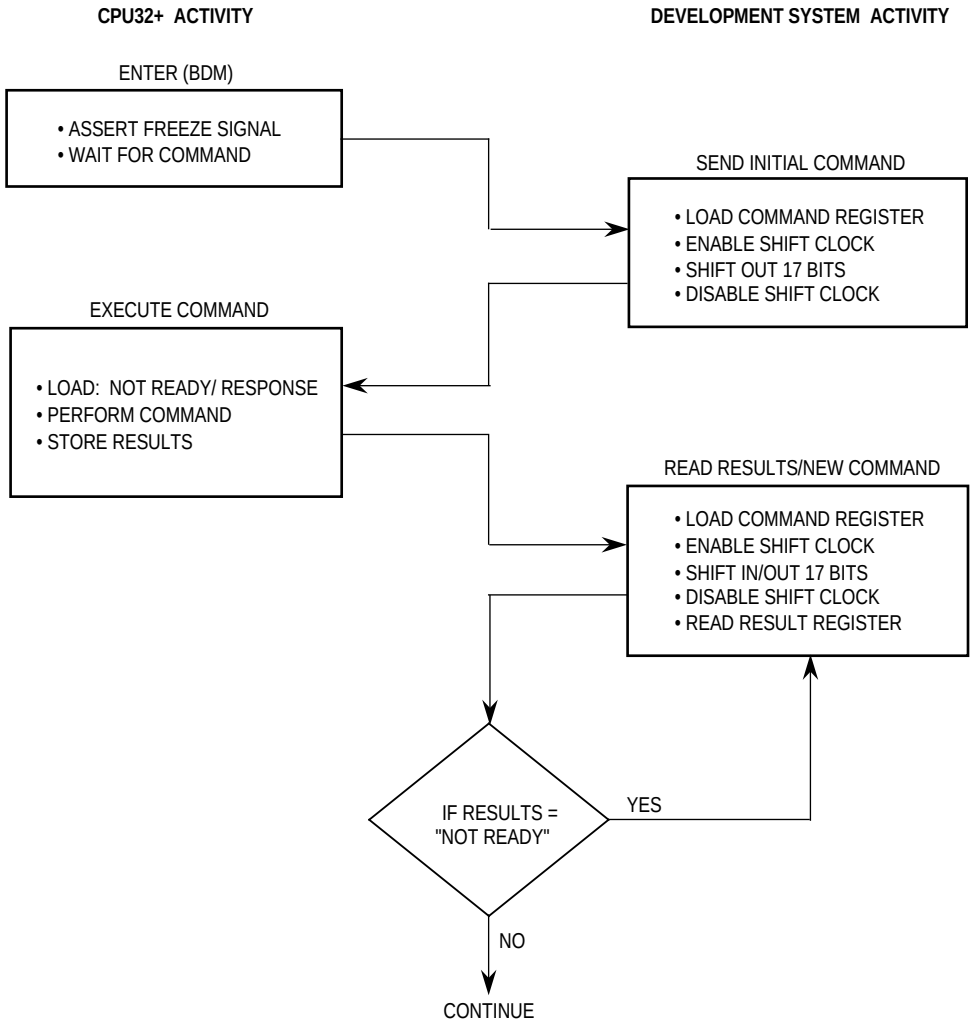


Figure 5-25. BDM Command Execution Flowchart

5.7.2.6 RETURNING FROM BDM. BDM is terminated when a resume execution (GO) or call user code (CALL) command is received. Both GO and CALL flush the instruction pipeline and prefetch instructions from the location pointed to by the RPC.

The return PC and the memory space referred to by the SR SUPV bit reflect any changes made during BDM. FREEZE is negated prior to initiating the first prefetch. Upon negation of FREEZE, the serial subsystem is disabled, and the signals revert to $\overline{\text{IPIPE}}$ and $\overline{\text{IFETCH}}$ functionality.

5.7.2.7 SERIAL INTERFACE. Communication with the CPU32+ during BDM occurs via a dedicated serial interface, which shares pins with other development features. The $\overline{\text{BKPT}}$ signal becomes the DSCLK; DSI is received on $\overline{\text{IFETCH}}$, and DSO is transmitted on $\overline{\text{IPIPE0}}$.

The serial interface uses a full-duplex synchronous protocol similar to the serial peripheral interface (SPI) protocol. The development system serves as the master of the serial link since it is responsible for the generation of DSCLK. If DSCLK is derived from the CPU32+ system clock, development system serial logic is unhindered by the operating frequency of the target processor. Operable frequency range of the serial clock is from DC to one-half the processor system clock frequency.

The serial interface operates in full-duplex mode—i.e., data is transmitted and received simultaneously by both master and slave devices. In general, data transitions occur on the falling edge of DSCLK and are stable by the following rising edge of DSCLK. Data is transmitted most significant bit (MSB) first and is latched on the rising edge of DSCLK.

The serial data word is 17 bits wide—16 data bits and a status/control (S/C) bit.

Bit 16 indicates the status of CPU-generated messages as listed in Table 5-10.

Table 5-10. CPU Generated Message Encoding

Encoding	Data	Message Type
0	xxxx	Valid Data Transfer
0	FFFF	Command Complete; Status OK
1	0000	Not Ready with Response; Come Again
1	0001	$\overline{\text{BERR}}$ Terminated Bus Cycle; Data Invalid
1	FFFF	Illegal Command

Command and data transfers initiated by the development system should clear bit 16. The current implementation ignores this bit; however, Motorola reserves the right to use this bit for future enhancements.

5.7.2.7.1 CPU Serial Logic. CPU32+ serial logic, shown in the left-hand portion of Figure 5-26, consists of transmit and receive shift registers and of control logic that includes synchronization, serial clock generation circuitry, and a received bit counter.

Both DSCLK and DSI are synchronized to on-chip clocks, thereby minimizing the chance of propagating metastable states into the serial state machine. Data is sampled during the high phase of CLKOUT. At the falling edge of CLKOUT, the sampled value is made available to internal logic. If there is no synchronization between CPU32+ and development system hardware, the minimum hold time on DSI with respect to DSCLK is one full period of CLKOUT.

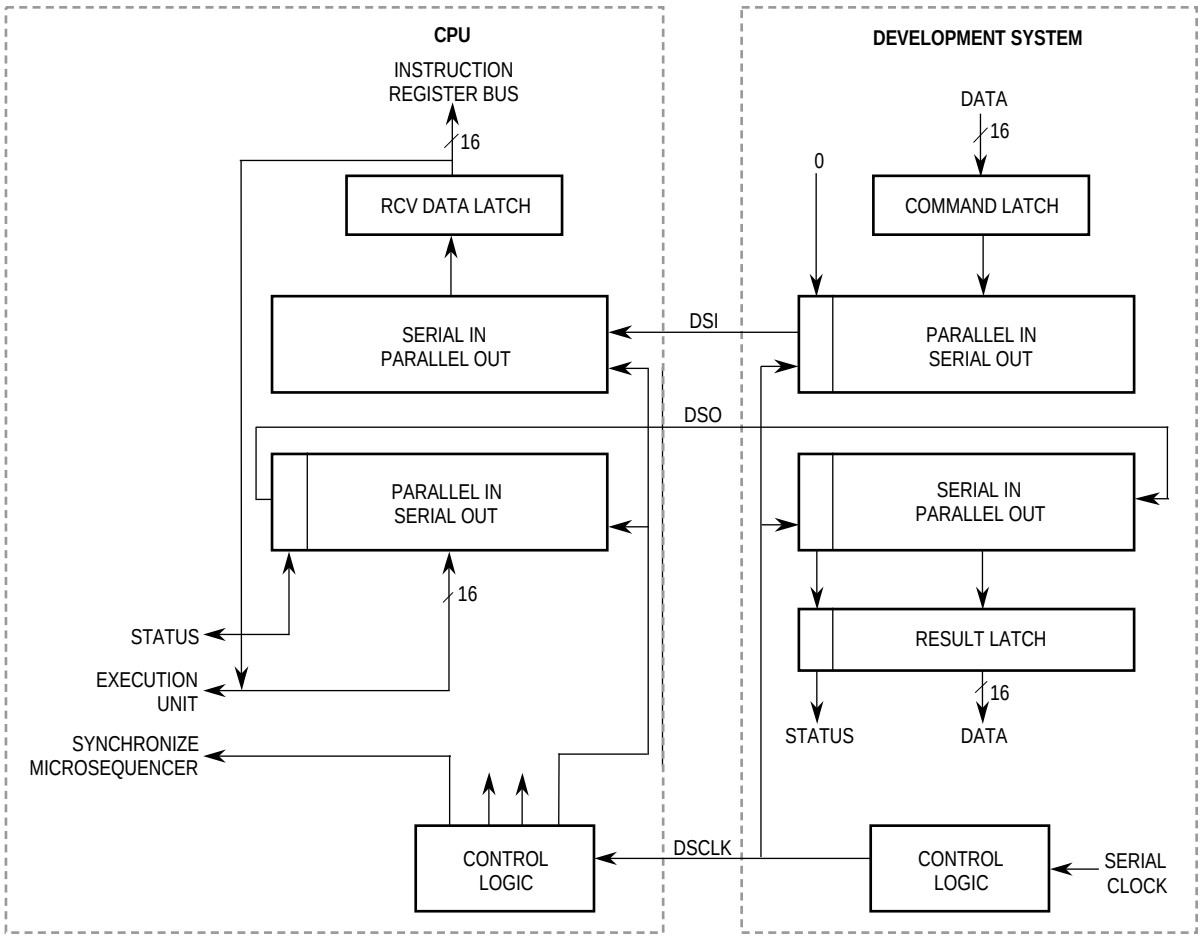


Figure 5-26. Debug Serial I/O Block Diagram

The serial state machine begins a sequence of events based on the rising edge of the synchronized DSCLK (see Figure 5-27). Synchronized serial data is transferred to the input shift register, and the received bit counter is decremented. One-half clock period later, the output shift register is updated, bringing the next output bit to the DSO signal. DSO changes relative to the rising edge of DSCLK and does not necessarily remain stable until the falling edge of DSCLK.

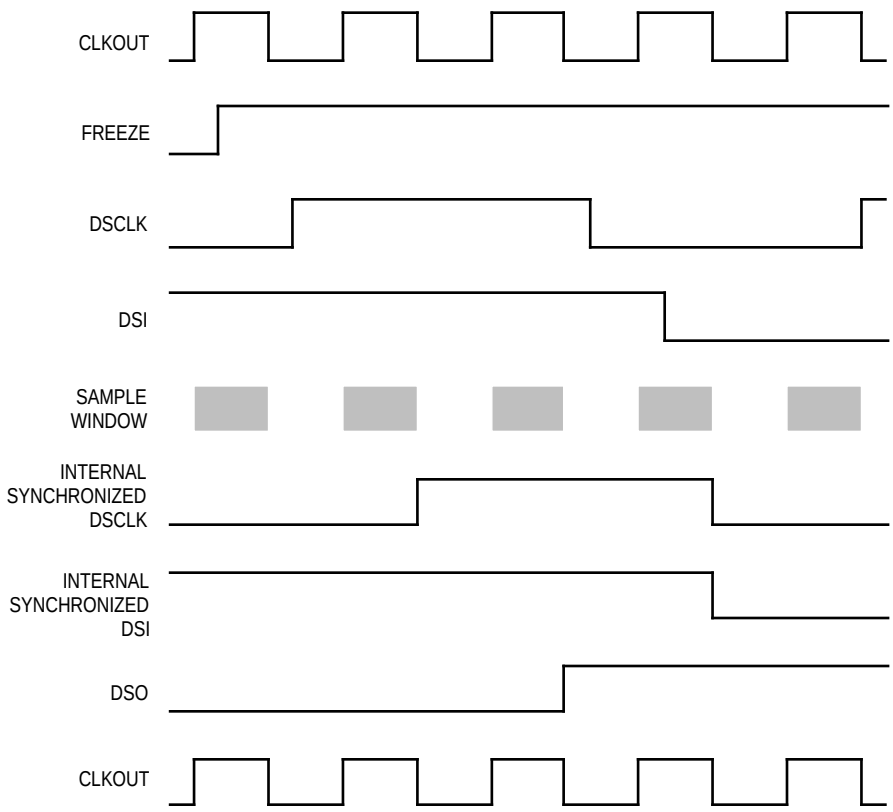


Figure 5-27. Serial Interface Timing Diagram

One clock period after the synchronized DSCLK has been seen internally, the updated counter value is checked. If the counter has reached zero, the receive data latch is updated from the input shift register. At this same time, the output shift register is reloaded with the “not ready/come again” response. Once the receive data latch has been loaded, the CPU is released to act on the new data. Response data overwrites the “not ready” response when the CPU has completed the current operation.

Data written into the output shift register appears immediately on the DSO signal. In general, this action changes the state of the signal from a high (“not ready” response status bit) to a low (valid data status bit) logic level. However, this level change only occurs if the command completes successfully. Error conditions overwrite the “not ready” response with the appropriate response that also has the status bit set.

A user can use the state change on DSO to signal hardware that the next serial transfer may begin. A timeout of sufficient length to trap error conditions that do not change the state of DSO should also be incorporated into the design. Hardware interlocks in the CPU prevent result data from corrupting serial transfers in progress.

5.7.2.7.2 Development System Serial Logic. The development system, as the master of the serial data link, must supply the serial clock. However, normal and BDM operations could interact if the clock generator is not properly designed.

Breakpoint requests are made by asserting $\overline{\text{BKPT}}$ to the low state in either of two ways. The primary method is to assert $\overline{\text{BKPT}}$ during a single bus cycle for which an exception is desired. Another method is to assert $\overline{\text{BKPT}}$, then continue to assert it until the CPU32+ responds by asserting FREEZE. This method is useful for forcing a transition into BDM when the bus is not being monitored. Each method requires a slightly different serial logic design to avoid spurious serial clocks.

Figure 5-28 represents the timing required for asserting $\overline{\text{BKPT}}$ during a single bus cycle.

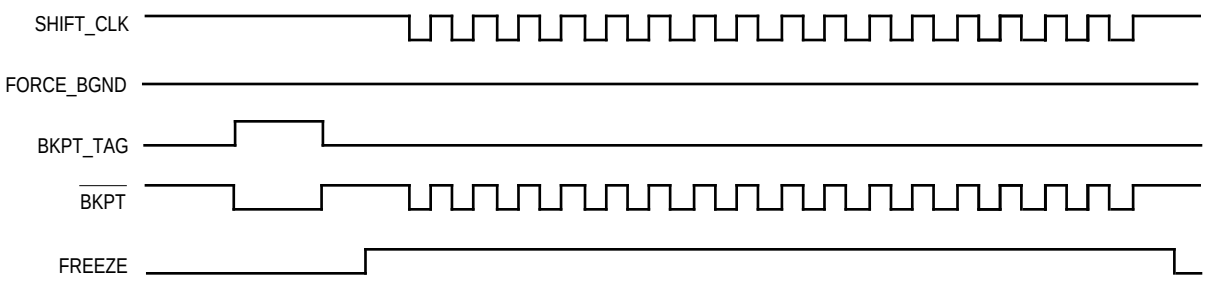


Figure 5-28. $\overline{\text{BKPT}}$ Timing for Single Bus Cycle

Figure 5-29 depicts the timing of the $\overline{\text{BKPT}}$/FREEZE method. In both cases, the serial clock is left high after the final shift of each transfer. This technique eliminates the possibility of accidentally tagging the prefetch initiated at the conclusion of a BDM session. As mentioned previously, all timing within the CPU is derived from the rising edge of the clock; the falling edge is effectively ignored.

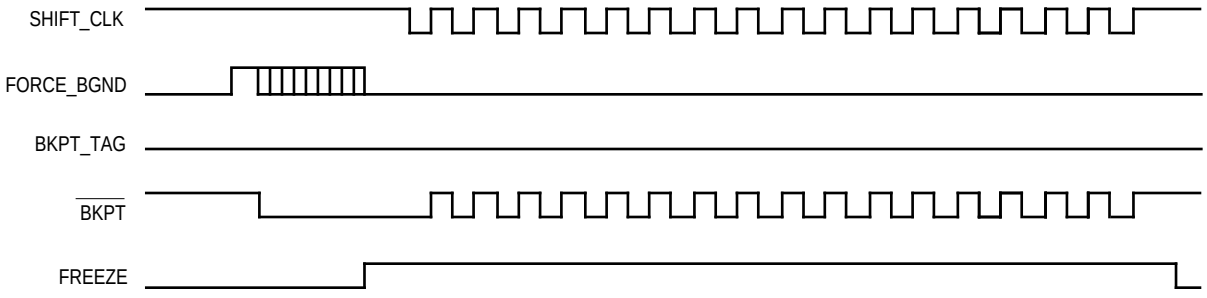


Figure 5-29. $\overline{\text{BKPT}}$ Timing for Forcing BDM

Figure 5-30 represents a sample circuit providing for both $\overline{\text{BKPT}}$ assertion methods. As the name implies, FORCE_BGND is used to force a transition into BDM by the assertion of $\overline{\text{BKPT}}$. FORCE_BGND can be a short pulse or can remain asserted until FREEZE is asserted. Once asserted, the set-reset latch holds $\overline{\text{BKPT}}$ low until the first SHIFT_CLK is applied.

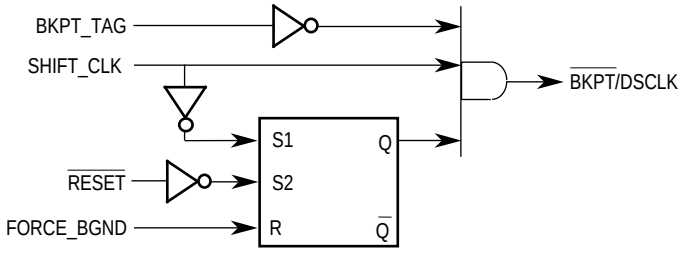


Figure 5-30. BKPT/DSCLK Logic Diagram

BKPT_TAG should be timed to the bus cycles since it is not latched. If extended past the assertion of FREEZE, the negation of BKPT_TAG appears to the CPU32+ as the first DSCLK.

DSCLK, the gated serial clock, is normally high, but it pulses low for each bit to be transferred. At the end of the seventeenth clock period, it remains high until the start of the next transmission. Clock frequency is implementation dependent and may range from DC to the maximum specified frequency. Although performance considerations might dictate a hardware implementation, software solutions can be used provided serial bus timing is maintained.

5.7.2.8 COMMAND SET. The following paragraphs describe the command set available in BDM.

5.7.2.8.1 Command Format. The following standard bit format is utilized by all BDM commands.

15 — 10	9	8	7 — 6	5	4	3	2 — 0
OPERATION	0	R/W	OP SIZE	0	0	A/D	REGISTER
EXTENSION WORD(S)							

Bits 15–10—Operation Field

The operation field specifies the commands. This 6-bit field provides for a maximum of 64 unique commands.

R/W Field

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

Operand Size

For sized operations, this field specifies the operand data size. All addresses are expressed as 32-bit absolute values. The size field is encoded as listed in Table 5-11.

Table 5-11. Size Field Encoding

Encoding	Operand Size
00	Byte
01	Word
10	Long
11	Reserved

Address/Data (A/D) Field

The A/D field is used by commands that operate on address and data registers. It determines whether the register field specifies a data or address register. One indicates an address register; zero indicates a data register. For other commands, this field may be interpreted differently.

Register Field:

In most commands, this field specifies the register number for operations performed on an address or data register.

Extension Word(s) (as required):

At this time, no command requires an extension word to specify fully the operation to be performed, but some commands require extension words for addresses or immediate data. Addresses require two extension words because only absolute long addressing is permitted. Immediate data can be either one or two words in length—byte and word data each require a single extension word; long-word data requires two words. Both operands and addresses are transferred most significant word first.

5.7.2.8.2 Command Sequence Diagram. A command sequence diagram (see Figure 5-31) illustrates the serial bus traffic for each command. Each bubble in the diagram represents a single 17-bit transfer across the bus. The top half in each diagram corresponds to the data transmitted by the development system to the CPU; the bottom half corresponds to the data returned by the CPU in response to the development system commands. Command and result transactions are overlapped to minimize latency.

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

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

NOTE

The “not ready” response can be ignored unless a memory bus cycle is in progress. Otherwise, the CPU can accept a new serial transfer with eight system clock periods.

In the third cycle, the development system supplies the low-order 16 bits of a memory address. The CPU always returns the “not ready” response in this cycle. At the completion of the third cycle, the CPU initiates a memory read operation. Any serial transfers that begin while the memory access is in progress return the “not ready” response.

Results are returned in the two serial transfer cycles following the completion of memory access. The data transmitted to the CPU during the final transfer is the opcode for the following command. Should a memory access generate either a bus or address error, an error status is returned in place of the result data.

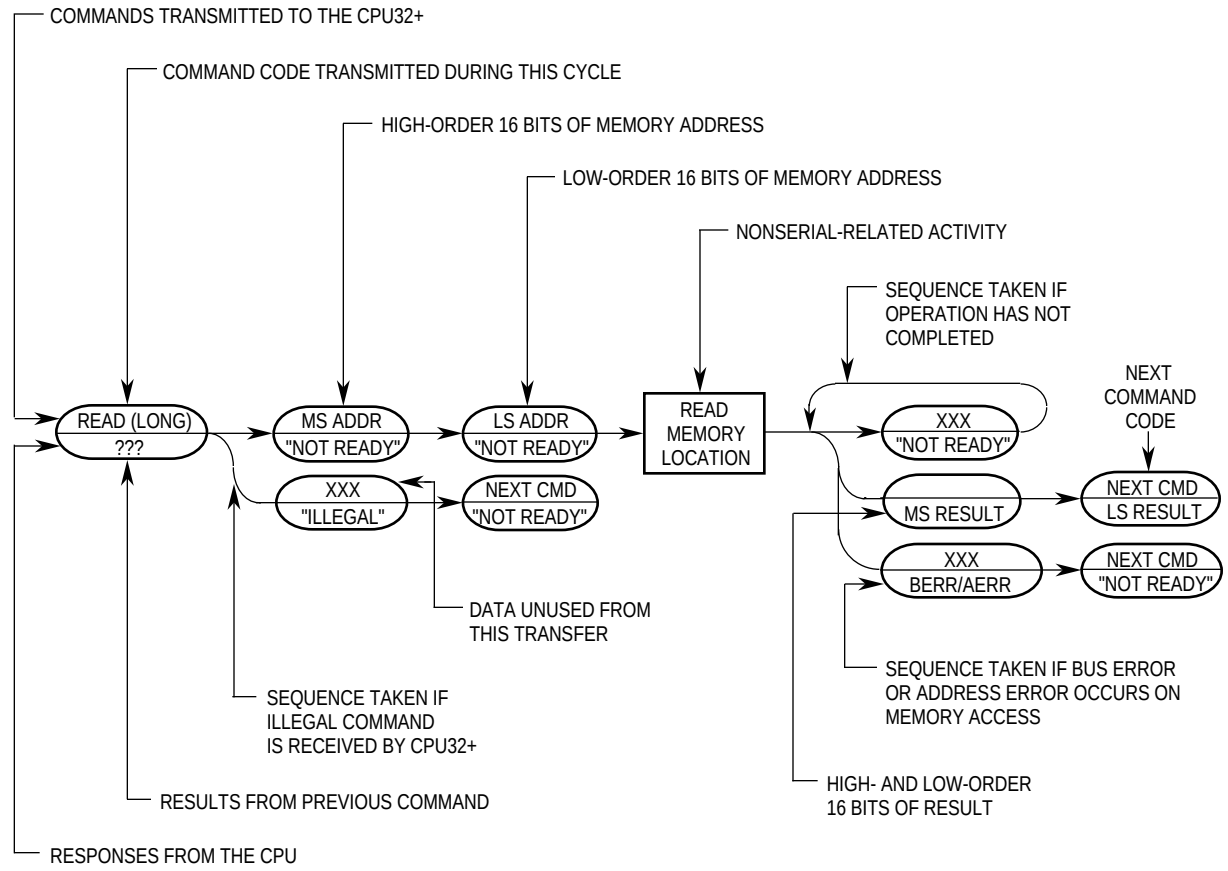


Figure 5-31. Command Sequence Diagram

5.7.2.8.3 Command Set Summary. The BDM command set is summarized in Table 5-12. Subsequent paragraphs contain detailed descriptions of each command.

Table 5-12. BDM Command Summary

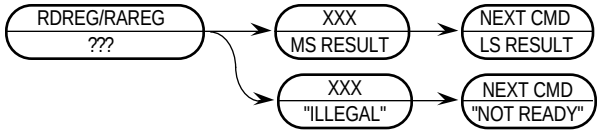
Command	Mnemonic	Description
Read A/D Register	RAREG/RDREG	Read the selected address or data register and return the results via the serial interface.
Write A/D Register	WAREG/WDREG	The data operand is written to the specified address or data register.
Read System Register	RSREG	The specified system control register is read. All registers that can be read in supervisor mode can be read in BDM.
Write System Register	WSREG	The operand data is written into the specified system control register.
Read Memory Location	READ	Read the sized data at the memory location specified by the long-word address. The SFC register determines the address space accessed.
Write Memory Location	WRITE	Write the operand data to the memory location specified by the long-word address. The DFC register determines the address space accessed.
Dump Memory Block	DUMP	Used in conjunction with the READ command to dump large blocks of memory. An initial READ is executed to set up the starting address of the block and to retrieve the first result. Subsequent operands are retrieved with the DUMP command.
Fill Memory Block	FILL	Used in conjunction with the WRITE command to fill large blocks of memory. An initial WRITE is executed to set up the starting address of the block and to supply the first operand. Subsequent operands are written with the FILL command.
Resume Execution	GO	The pipeline is flushed and refilled before resuming instruction execution at the return PC.
Call User Code	CALL	Current PC is stacked at the location of the current SP. Instruction execution begins at user patch code.
Reset Peripherals	RST	Asserts RESET for 512 clock cycles. The CPU is not reset by this command. Synonymous with the CPU RESET instruction.
No Operation	NOP	NOP performs no operation and may be used as a null command.

5.7.2.8.4 Read A/D Register (RAREG/RDREG). Read the selected address or data register and return the results via the serial interface.

Command Format:

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

Command Sequence:



Operand Data:

None

Result Data:

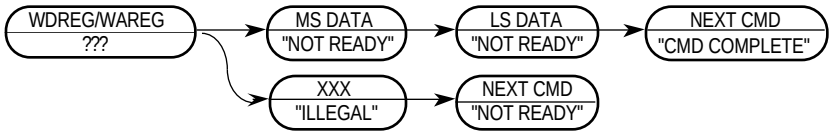
The contents of the selected register are returned as a long-word value. The data is returned most significant word first.

5.7.2.8.5 Write A/D Register (WAREG/WDREG). The operand (long-word) data is written to the specified address or data register. All 32 bits of the register are altered by the write.

Command Format:

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

Command Sequence:



Operand Data:

Long-word data is written into the specified address or data register. The data is supplied most significant word first.

Result Data:

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

5.7.2.8.6 Read System Register (RSREG). The specified system control register is read. All registers that can be read in supervisor mode can be read in BDM. Several internal temporary registers are also accessible.

Command Format:

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

Command Sequence:

Operand Data:

None

Result Data:

Always returns 32 bits of data, regardless of the size of the register being read. If the register is less than 32 bits, the result is returned zero extended.

Register Field:

The system control register is specified by the register field (see Table 5-13).

Table 5-13. Register Field for RSREG and WSREG

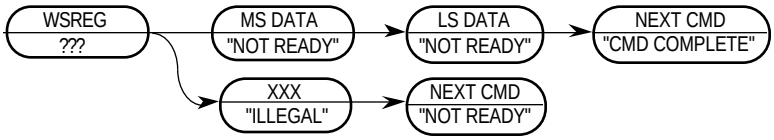
System Register	Select Code
Return Program Counter (RPC)	0000
Current Instruction Program Counter (PCC)	0001
Status Register (SR)	1011
User Stack Pointer (USP)	1100
Supervisor Stack Pointer (SSP)	1101
Source Function Code Register (SFC)	1110
Destination Function Code Register (DFC)	1111
Temporary Register A (ATEMP)	1000
Fault Address Register (FAR)	1001
Vector Base Register (VBR)	1010

5.7.2.8.7 Write System Register (WSREG). Operand data is written into the specified system control register. All registers that can be written in supervisor mode can be written in BDM. Several internal temporary registers are also accessible.

Command Format:

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

Command Sequence:



Operand Data:

The data to be written into the register is always supplied as a 32-bit long word. If the register is less than 32 bits, the least significant word is used.

Result Data:

“Command complete” status is returned when register write is complete.

Register Field:

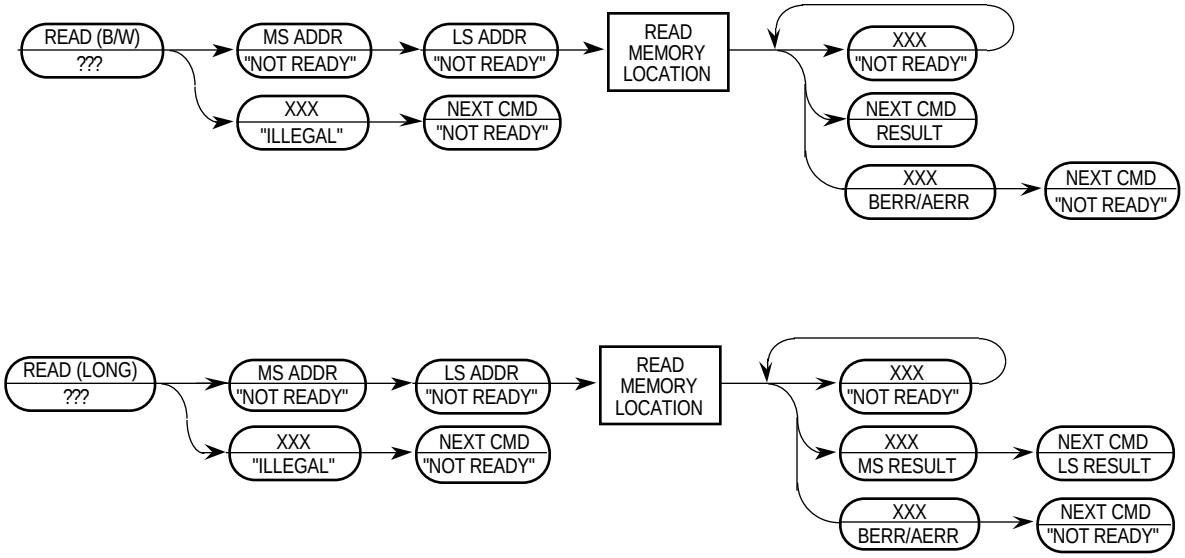
The system control register is specified by the register field (see Table 5-24). The FAR is a read-only register—any write to it is ignored.

5.7.2.8.8 Read Memory Location (READ). Read the sized data at the memory location specified by the long-word address. Only absolute addressing is supported. The SFC register determines the address space accessed. Valid data sizes include byte, word, or long word.

Command Format:

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

Command Sequence:



Operand Data:

The single operand is the long-word address of the requested memory location.

Result Data:

The requested data is returned as either a word or long word. Byte data is returned in the least significant byte of a word result, with the upper byte cleared. Word results return 16 bits of significant data; long-word results return 32 bits.

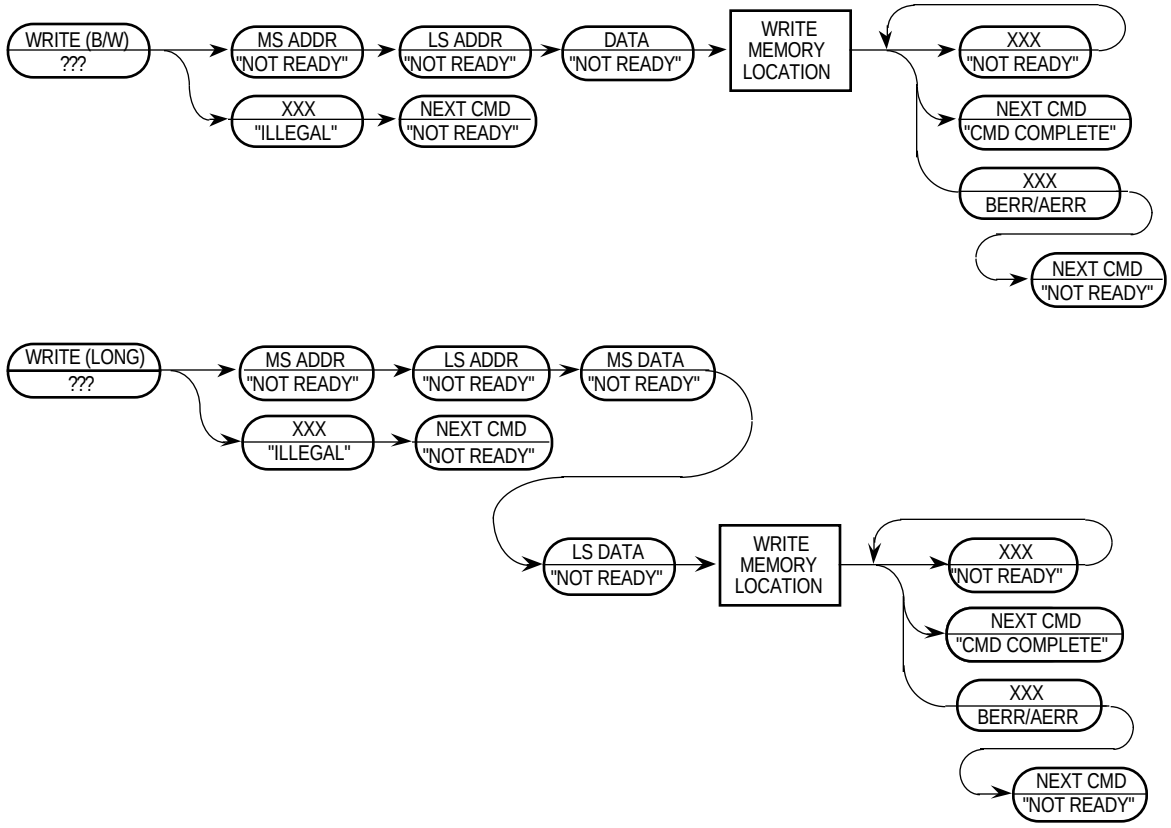
A successful read operation returns data bit 16 cleared. If a bus or address error is encountered, the returned data is $10001.

5.7.2.8.9 Write Memory Location (WRITE). Write the operand data to the memory location specified by the long-word address. The DFC register determines the address space accessed. Only absolute addressing is supported. Valid data sizes include byte, word, and long word.

Command Format:

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

Command Sequence:



Operand Data:

Two operands are required for this instruction. The first operand is a long-word absolute address that specifies a location to which the operand data is to be written. The second operand is the data. Byte data is transmitted as a 16-bit word, justified in the least significant byte; 16- and 32-bit operands are transmitted as 16 and 32 bits, respectively.

Result Data:

Successful write operations return a status of $0FFFF. Bus or address errors on the write cycle are indicated by the assertion of bit 16 in the status message and by a data pattern of $0001.

5.7.2.8.10 Dump Memory Block (DUMP). DUMP is used in conjunction with the READ command to dump large blocks of memory. An initial READ is executed to set up the starting address of the block and to retrieve the first result. Subsequent operands are retrieved with the DUMP command. The initial address is incremented by the operand size (1, 2, or 4) and saved in a temporary register. Subsequent DUMP commands use this address, increment it by the current operand size, and store the updated address back in the temporary register.

NOTE

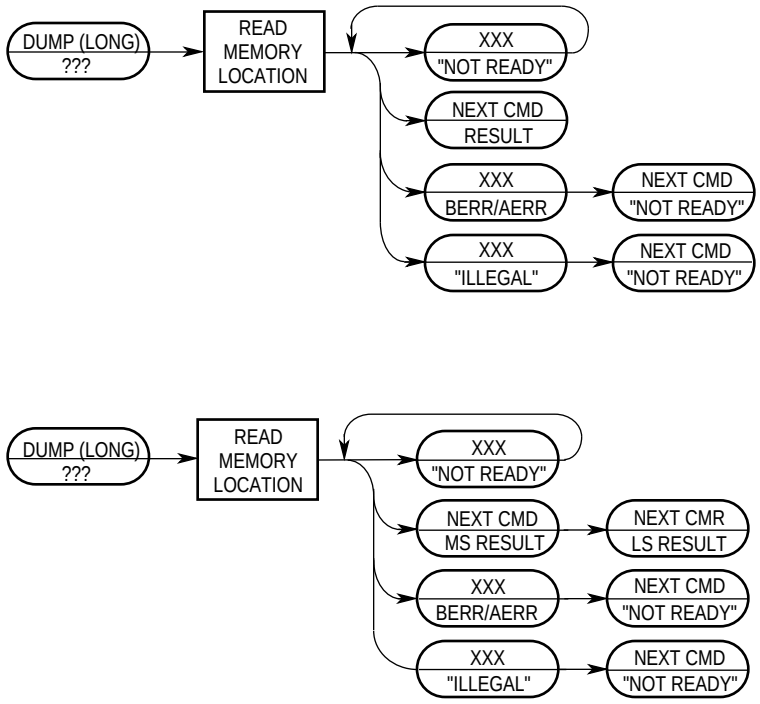
The DUMP command does not check for a valid address in the temporary register—DUMP is a valid command only when preceded by another DUMP or by a READ command. Otherwise, the results are undefined. The NOP command can be used for intercommand padding without corrupting the address pointer.

The size field is examined each time a DUMP command is given, allowing the operand size to be altered dynamically.

Command Format:

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

Command Sequence:



Operand Data:

None

Result Data:

Requested data is returned as either a word or long word. Byte data is returned in the least significant byte of a word result. Word results return 16 bits of significant data; long-word results return 32 bits. Status of the read operation is returned as in the READ command: $0xxxx for success, $10001 for bus or address errors.

5.7.2.8.11 Fill Memory Block (FILL). FILL is used in conjunction with the WRITE command to fill large blocks of memory. An initial WRITE is executed to set up the starting address of the block and to supply the first operand. Subsequent operands are written with the FILL command. The initial address is incremented by the operand size (1, 2, or 4) and is saved in a temporary register. Subsequent FILL commands use this address, increment it by the current operand size, and store the updated address back in the temporary register.

NOTE

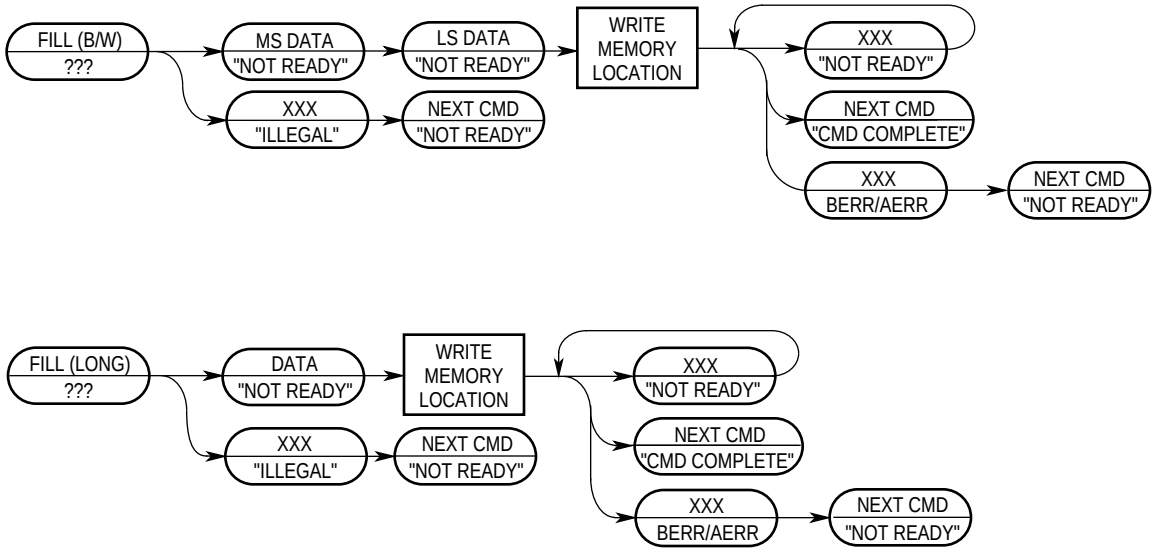
The FILL command does not check for a valid address in the temporary register—FILL is a valid command only when preceded by another FILL or by a WRITE command. Otherwise, the results are undefined. The NOP command can be used for intercommand padding without corrupting the address pointer.

The size field is examined each time a FILL command is given, allowing the operand size to be altered dynamically.

Command Format:

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

Command Sequence:



Operand Data:

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

Result Data:

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

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

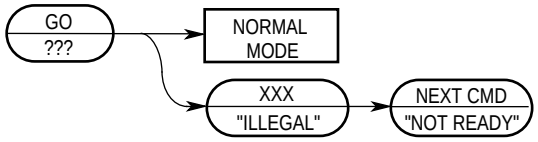
NOTE

The processor exits BDM when a bus error or address error occurs on the first instruction prefetch from the new PC—the error is trapped as a normal mode exception. The stacked value of the current PC may not be valid in this case, depending on the state of the machine prior to entering BDM. For address error, the PC does not reflect the true return PC. Instead, the stacked fault address is the (odd) return PC.

Command Format:

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

Command Sequence:



Operand Data:

None

Result Data:

None

5.7.2.8.13 Call User Code (CALL). This instruction provides a convenient way to patch user code. The return PC is stacked at the location pointed to by the current SP. The stacked PC serves as a return address to be restored by the RTS command that terminates the patch routine. After stacking is complete, the 32-bit operand data is loaded into the PC. The pipeline is flushed and refilled from the location pointed to by the new PC, BDM is exited, and normal mode instruction execution begins.

NOTE

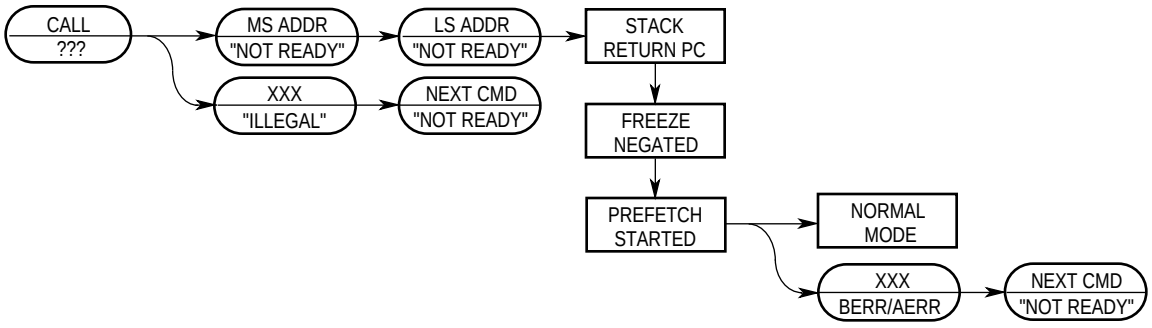
If a bus error or address error occurs during return address stacking, the CPU returns an error status via the serial interface and remains in BDM.

If a bus error or address error occurs on the first instruction prefetch from the new PC, the processor exits BDM and the error is trapped as a normal mode exception. The stacked value of the current PC may not be valid in this case, depending on the state of the machine prior to entering BDM. For address error, the PC does not reflect the true return PC. Instead, the stacked fault address is the (odd) return PC.

Command Format:

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

Command Sequence:



Operand Data:

The 32-bit operand data is the starting location of the patch routine, which is the initial PC upon exiting BDM.

Result Data:

None

As an example, consider the following code segment. It outputs a character from the MC68340 serial module channel A.

```
CHKSTAT:    MOVE.B    SRA,D0          Move serial status to D0
            BNE.B     CHKSTAT         Loop until condition true
            MOVE.B    TBA,OUTPUT      Transmit character


MISSING:    ANDI.B    #3,D0           Check for TxEMP flag
            RTS
```

BDM and the CALL command can be used to patch the code as follows:

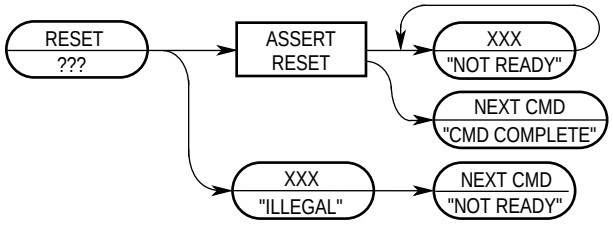
1. Breakpoint user program at CHKSTAT
2. Enter BDM
3. Execute CALL command to MISSING
4. Exit BDM
5. Execute MISSING code
6. Return to user program

5.7.2.8.14 Reset Peripherals (RST). RST asserts $\overline{\text{RESET}}$ for 512 clock cycles. The CPU is not reset by this command. This command is synonymous with the CPU RESET instruction.

Command Format:

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

Command Sequence:



Operand Data:

None

Result Data:

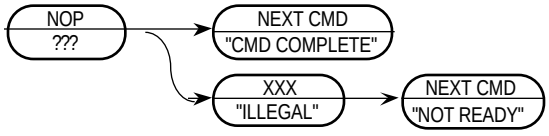
The “command complete” response ($0FFFF) is loaded into the serial shifter after negation of RESET.

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

Command Format:

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

Command Sequence:



Operand Data:

None

Result Data:

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

5.7.2.8.16 Future Commands. Unassigned command opcodes are reserved by Motorola for future expansion. All unused formats within any revision level will perform a NOP and return the ILLEGAL command response.

5.7.3 Deterministic Opcode Tracking

The CPU32+ utilizes deterministic opcode tracking to trace program execution. Three signals, IPIPE1, IPIPE0, and IFETCH, provide all information required to analyze instruction pipeline operation.

5.7.3.1 INSTRUCTION FETCH (IFETCH). IFETCH indicates which bus cycles are accessing data to fill the instruction pipeline. IFETCH is pulse-width modulated to multiplex two indications on a single pin. Asserted for a single clock cycle, IFETCH indicates that the data from the current bus cycle is to be routed to the instruction pipeline. IFETCH held low for two clock cycles indicates that the instruction pipeline has been flushed. The data from the bus cycle is used to begin filling the empty pipeline. Both user and supervisor mode fetches are signaled by IFETCH.

Proper tracking of bus cycles via IFETCH on a fast bus requires a simple state machine. On a two-clock bus, IFETCH may signal a pipeline flush with associated prefetch followed immediately by a second prefetch. That is, IFETCH remains asserted for three clocks, two clocks indicating the flush/fetch and a third clock signaling the second fetch. These two operations are easily discerned if the tracking logic samples IFETCH on the two rising edges of CLKOUT, which follow the AS (DS during show cycles) falling edge. Three-clock and slower bus cycles allow time for negation of the signal between consecutive indications and do not experience this operation.

5.7.3.2 INSTRUCTION PIPE (IPIPE1, IPIPE0). The internal instruction pipeline can be modeled as a three-stage FIFO (see Figure 5-32). Stage A is an input buffer—data can be used out of stages B and C. The IPIPE1, IPIPE0 signals indicate the advance of instructions in the pipeline.

The 16-bit instruction register A (IRA) and 16-bit instruction register L (IRL) hold incoming words as they are prefetched. No decoding occurs in IRA or IRL. Instruction register B (IRB) provides initial decoding of the opcode and decoding of extension words; it is a source of immediate data. Instruction register C (IRC) supplies residual opcode decoding during instruction execution.

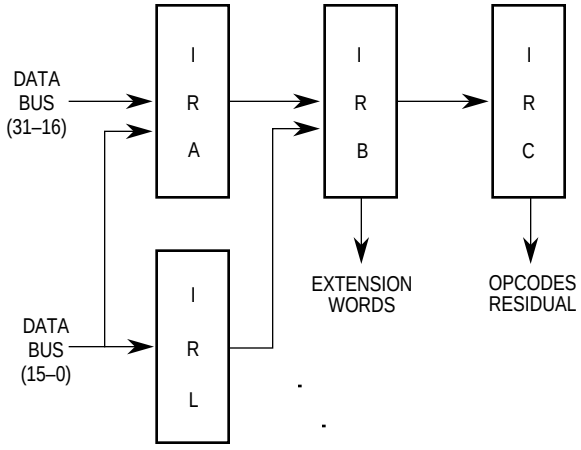


Figure 5-32. Functional Model of Instruction Pipeline

When the CPU32+ is performing 32-bit instruction fetches, IRA and IRL are loaded in parallel from the IMB or the CIC—the most significant word is loaded into IRA. As the microcode sequencing generates requests from pipeline stage A, instruction words are loaded into IRB first from IRA, followed by IRL. Once IRL is transferred, IRA and IRL are refilled during the next instruction fetch bus cycle. For 16-bit instruction fetches, IRL is not used, and IRB is always loaded from IRA.

When $\overline{\text{IPIPE0}}$ is low during a clock cycle, it indicates the start of a new instruction and subsequent replacement of data in IRC. This action causes a full advance of the pipeline (IRB $\Rightarrow$ IRC and IRA/IRL $\Rightarrow$ IRB).

When $\overline{\text{IPIPE1}}$ is low during a clock cycle, it indicates the use of an extension word from IRB on that clock cycle. Regardless of the presence of valid data in IRA or IRL, the contents of IRB are invalidated when $\overline{\text{IPIPE1}}$ is asserted. If IRA or IRL contain valid data, the data is copied into IRB (IRA/IRL $\Rightarrow$ IRB), and the IRB stage is revalidated.

Data loaded into IRA and IRL propagates automatically through subsequent empty pipeline stages. Signals that show the progress of instructions through IRB and IRC are necessary to accurately monitor pipeline operation. These signals are provided by IRA, IRL and IRB validity bits. When a pipeline advance occurs, the validity bit of the stage being loaded is set, and the validity bit of the stage supplying the data is negated.

Because instruction execution is not timed to bus activity, $\overline{\text{IPIPE1}}$ and $\overline{\text{IPIPE0}}$ are synchronized with the system clock and not the bus. Figure 5-33 illustrates the timing in relation to the system clock.

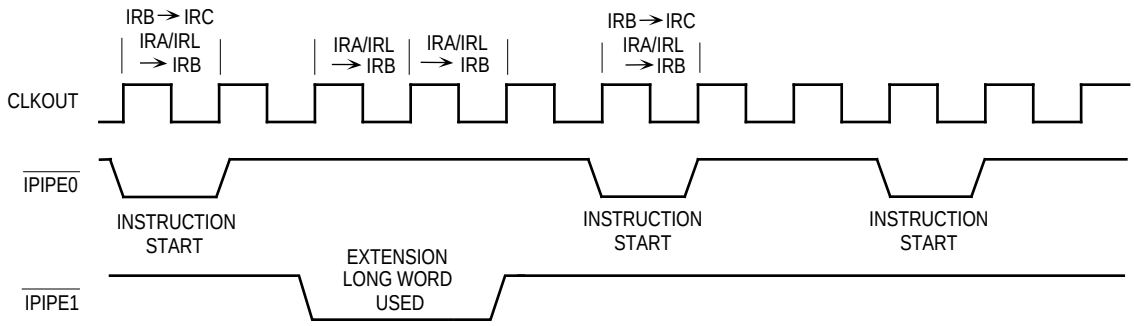


Figure 5-33. Instruction Pipeline Timing Diagram

$\overline{\text{IPIPE1}}$ and $\overline{\text{IPIPE0}}$ should be sampled on the falling edge of the clock. In BDM mode, $\overline{\text{IPIPE1}}$ is undefined and $\overline{\text{IPIPE0}}$ functions as DSO.

5.7.3.3 OPCODE TRACKING DURING LOOP MODE. $\overline{\text{IPIPEx}}$ and $\overline{\text{IFETCH}}$ continue to work normally during loop mode. $\overline{\text{IFETCH}}$ indicates all instruction fetches up through the point that data begins recirculating within the instruction pipeline. $\overline{\text{IPIPEx}}$ continues to signal the start of instructions and the use of extension words even though data is being recirculated internally. $\overline{\text{IFETCH}}$ returns to normal operation with the first fetch after exiting loop mode.

5.8 INSTRUCTION EXECUTION TIMING

This section describes the instruction execution timing of the CPU32+. External clock cycles are used to provide accurate execution and operation timing guidelines, but not exact timing for every possible circumstance. This approach is used because exact execution time for an instruction or operation depends on concurrence of independently scheduled resources, on memory speeds, and on other variables.

An assembly language programmer or compiler writer can use the information in this section to predict the performance of the CPU32+. Additionally, timing for exception processing is included so that designers of multitasking or real-time systems can predict task-switch overhead, maximum interrupt latency, and similar timing parameters. Instruction timing is given in clock cycles to eliminate clock frequency dependency.

Most instruction timing information in the following subsections is taken from the CPU32 documentation. It applies to the CPU32+ when it is executing in 16-bit mode.

5.8.1 Resource Scheduling

The CPU32+ contains several independently scheduled resources. The organization of these resources within the CPU32+ is shown in Figure 5-34. Some variation in instruction execution timing results from concurrent resource utilization. Because resource scheduling is not directly related to instruction boundaries, it is impossible to make an accurate prediction of the time required to complete an instruction without knowing the entire context within which the instruction is executing.

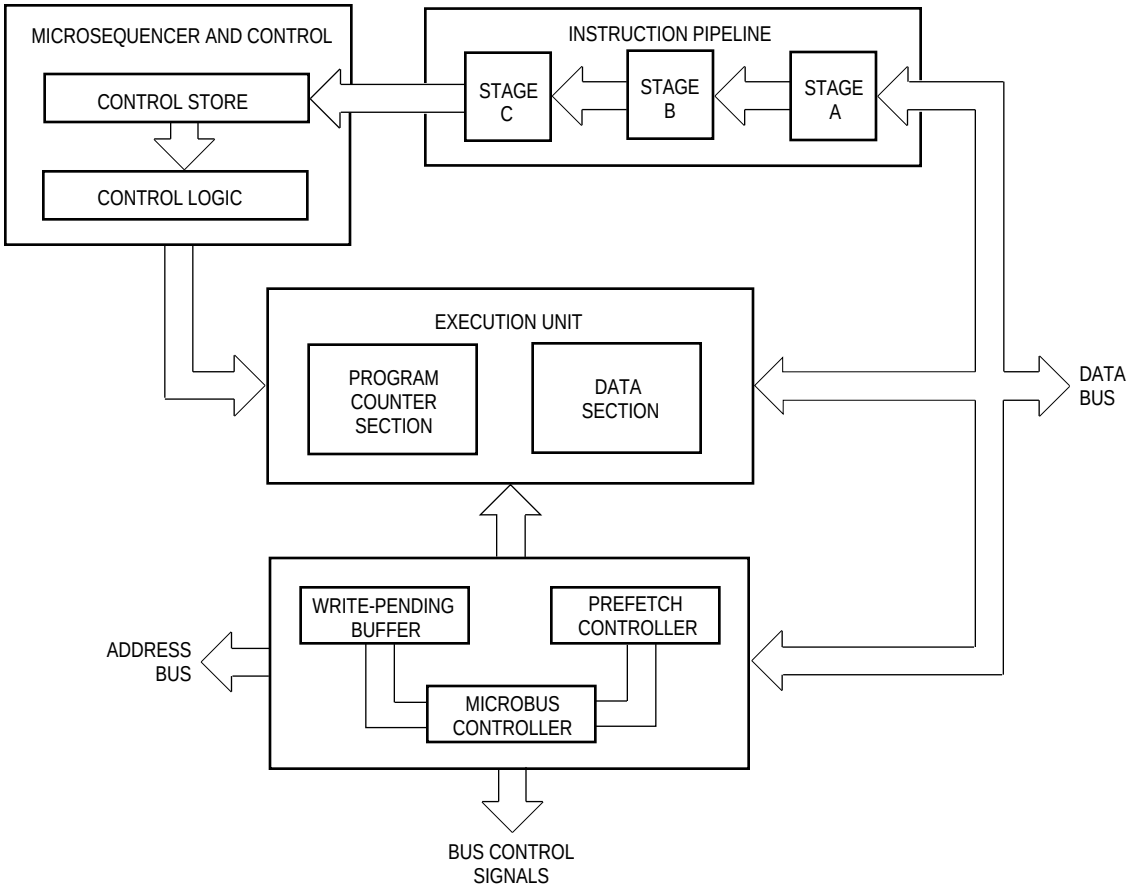


Figure 5-34. Block Diagram of Independent Resources

5.8.1.1 MICROSEQUENCER. The microsequencer either executes microinstructions or awaits completion of accesses necessary to continue microcode execution. The microsequencer supervises the bus controller, instruction execution, and internal processor operations such as calculation of EA and setting of condition codes. It also initiates instruction word prefetches after a change of flow and controls validation of instruction words in the instruction pipeline.

5.8.1.2 INSTRUCTION PIPELINE. The CPU32+ contains a two-word instruction pipeline where instruction opcodes are decoded. Each stage of the pipeline is initially filled under microsequencer control and subsequently refilled by the prefetch controller as it empties.

Stage A of the instruction pipeline is a buffer. Prefetches completed on the bus before stage B empties are temporarily stored in this buffer. Instruction words (instruction operation words and all extension words) are decoded at stage B. Residual decoding and execution occur in stage C.

Each pipeline stage has an associated status bit that shows whether the word in that stage was loaded with data from a bus cycle that terminated abnormally.

5.8.1.3 BUS CONTROLLER RESOURCES. The bus controller consists of the instruction prefetch controller, the write pending buffer, and the microbus controller. These three resources transact all reads, writes, and instruction prefetches required for instruction execution.

The bus controller and microsequencer operate concurrently. The bus controller can perform a read or write or schedule a prefetch while the microsequencer controls EA calculation or sets condition codes.

The microsequencer can also request a bus cycle that the bus controller cannot perform immediately. When this happens, the bus cycle is queued, and the bus controller runs the cycle when the current cycle has completed.

5.8.1.3.1 Prefetch Controller. The instruction prefetch controller receives an initial request from the microsequencer to initiate prefetching at a given address. Subsequent prefetches are initiated by the prefetch controller whenever a pipeline stage is invalidated, either through instruction completion or through use of extension words. Prefetch occurs as soon as the bus is free of operand accesses previously requested by the microsequencer. Additional state information permits the controller to inhibit prefetch requests when a change in instruction flow (e.g., a jump or branch instruction) is anticipated.

In a typical program, 10 to 25 percent of the instructions cause a change of flow. Each time a change occurs, the instruction pipeline must be flushed and refilled from the new instruction stream. If instruction prefetches, rather than operand accesses, were given priority, many instruction words would be flushed unused, and necessary operand cycles would be delayed. To maximize available bus bandwidth, the CPU32+ will schedule a prefetch only when the next instruction is not a change-of-flow instruction and when there is room in the pipeline for the prefetch.

5.8.1.3.2 Write-Pending Buffer. The CPU32+ incorporates a single-operand write-pending buffer. The buffer permits the microsequencer to continue execution after a request for a write cycle is queued in the bus controller. The time needed for a write at the end of an instruction can overlap the head cycle time for the following instruction, thus reducing overall execution time. Interlocks prevent the microsequencer from overwriting the buffer.

5.8.1.3.3 Microbus Controller. The microbus controller performs bus cycles issued by the microsequencer. Operand accesses always have priority over instruction prefetches. Word and byte operands are accessed in a single CPU-initiated bus cycle, although the external bus interface may be required to initiate a second cycle when a word operand is sent to a byte-sized external port. If long operands are accessed from a 16-bit port, they are accessed in two bus cycles, most significant word first.

The instruction pipeline is capable of recognizing instructions that cause a change of flow. It informs the bus controller when a change of flow is imminent, and the bus controller refrains from starting prefetches that would be discarded due to the change of flow.

5.8.1.4 INSTRUCTION EXECUTION OVERLAP. Overlap is the time, measured in clock cycles, that an instruction executes concurrently with the previous instruction. As shown in Figure 5-35, portions of instructions A and B execute simultaneously, reducing total execution time. Because portions of instructions B and C also overlap, overall execution time for all three instructions is also reduced.

Each instruction contributes to the total overlap time. The portion of execution time at the end of instruction A that can overlap the beginning of instruction B is called the tail of instruction A. The portion of execution time at the beginning of instruction B that can overlap the end of instruction A is called the head of instruction B. The total overlap time between instructions A and B is the smaller tail of A and the head of B.

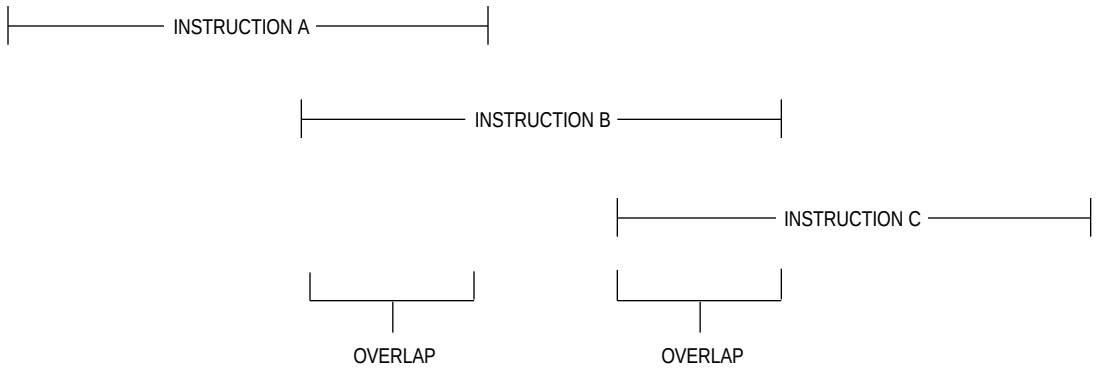


Figure 5-35. Simultaneous Instruction Execution

The execution time attributed to instructions A, B, and C after considering the overlap is illustrated in Figure 5-36. The overlap time is attributed to the execution time of the completing instruction. The following equation shows the method for calculating the overlap time:

$$\text{Overlap} = \min (\text{Tail}_N, \text{Head}_{N+1})$$

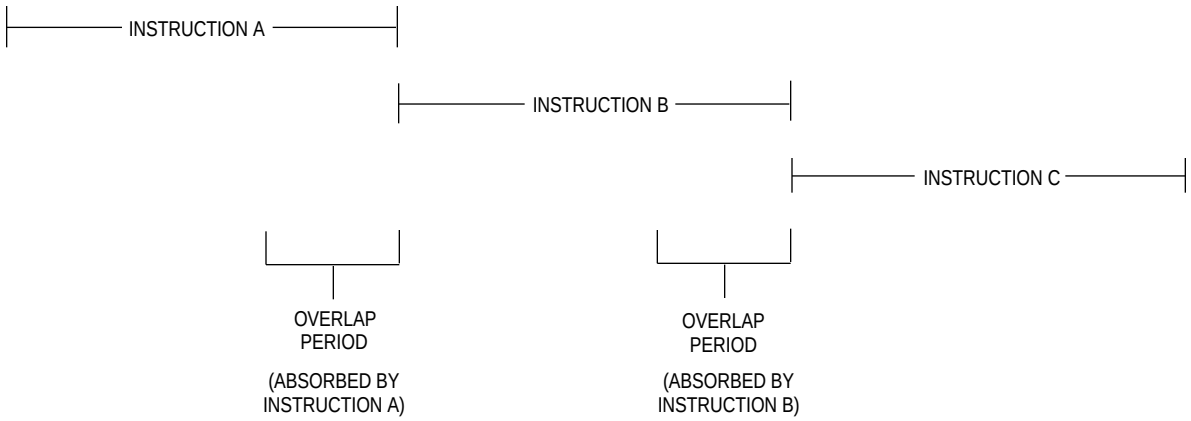


Figure 5-36. Attributed Instruction Times

5.8.1.5 EFFECTS OF WAIT STATES. The CPU32+ access time for on-chip peripherals is two clocks. While two-clock external accesses are possible when the bus is operated in a synchronous mode, a typical external memory speed is three or more clocks.

All instruction times listed in this section are for word access only (unless an explicit exception is given), and are based on the assumption that both instruction fetches and operand cycles are to a two-clock memory. Wait states due to slow external memory must be added to the access time for each bus cycle.

A typical application has a mixture of bus speeds—program execution from an off-chip ROM, accesses to on-chip peripherals, storage of variables in slow off-chip RAM, and accesses to external peripherals with speeds ranging from moderate to very slow. To arrive at an accurate instruction time calculation, each bus access must be individually considered. Many instructions have a head cycle count, which can overlap the cycles of an operand fetch to slower memory started by a previous instruction. In these cases, an increase in access time has no effect on the total execution time of the pair of instructions.

To trace instruction execution time by monitoring the external bus, note that the order of operand accesses for a particular instruction sequence is always the same provided bus speed is unchanged and the interleaving of instruction prefetches with operands within each sequence is identical.

5.8.1.6 INSTRUCTION EXECUTION TIME CALCULATION. The overall execution time for an instruction depends on the amount of overlap with previous and subsequent instructions. To calculate an instruction time estimate, the entire code sequence must be analyzed. To derive the actual instruction execution times for an instruction sequence, the instruction times listed in the tables must be adjusted to account for overlap.

The formula for this calculation is as follows:

$$C_1 - \min(T_1, H_2) + C_2 - \min(T_2, H_3) + C_3 - \min(T_3, H_4) + \ldots$$

where:

C_N is the number of cycles listed for instruction N

T_N is the tail time for instruction N

H_N is the head time for instruction N

$\min(T_N, H_M)$ is the minimum of parameters T_N and H_M

The number of cycles for the instruction (C_N) can include one or two EA calculations in addition to the raw number in the cycles column. In these cases, calculate overall instruction time as if it were for multiple instructions, using the following equation:

$$\langle CEA \rangle - \min (T_{EA}, H_{OP}) + C_{OP}$$

where:

$\langle CEA \rangle$ is the instruction's EA time

C_{OP} is the instruction's operation time

T_{EA} is the EA's tail time

H_{OP} is the instruction operation's head time

$\min (T_N, H_M)$ is the minimum of parameters T_N and H_M

The overall head for the instruction is the head for the EA, and the overall tail for the instruction is the tail for the operation. Therefore, the actual equation for execution time becomes:

$$C_{OP1} - \min (T_{OP1}, H_{EA2}) + \langle CEA \rangle_2 - \min (T_{EA2}, H_{OP2}) + C_{OP2} - \min (T_{OP2}, H_{EA3}) + \ldots$$

Every instruction must prefetch to replace itself in the instruction pipe. Usually, these prefetches occur during or after an instruction. A prefetch is permitted to begin in the first clock of any indexed EA mode operation.

Additionally, a prefetch for an instruction is permitted to begin two clocks before the end of an instruction provided the bus is not being used. If the bus is being used, then the prefetch occurs at the next available time when the bus would otherwise be idle.

5.8.1.7 EFFECTS OF NEGATIVE TAILS. When the CPU32+ changes instruction flow, the instruction decode pipeline must begin refilling before instruction execution can resume. Refilling forces a two-clock idle period at the end of the change-of-flow instruction. This idle period can be used to prefetch an additional word on the new instruction path. Because of the stipulation that each instruction must prefetch to replace itself, the concept of negative tails has been introduced to account for these free clocks on the bus.

On a two-clock bus, it is not necessary to adjust instruction timing to account for the potential extra prefetch. The cycle times of the microsequencer and bus are matched, and no additional benefit or penalty is obtained. In the instruction execution time equations, a zero should be used instead of a negative number.

Negative tails are used to adjust for slower fetches on slower buses. Normally, increasing the length of prefetch bus cycles directly affects the cycle count and tail values found in the tables.

In the following equations, negative tail values are used to negate the effects of a slower bus. The equations are generalized, however, so that they may be used on any speed bus with any tail value.

```
NEW_TAIL = OLD_TAIL + (NEW_CLOCK – 2)

IF ((NEW_CLOCK – 4) > 0) THEN

   NEW_CYCLE = OLD_CYCLE + (NEW_CLOCK – 2) + (NEW_CLOCK – 4)

ELSE

   NEW_CYCLE = OLD_CYCLE + (NEW _CLOCK – 2)
```

where:

NEW_TAIL/NEW_CYCLE is the adjusted tail/cycle at the slower speed

OLD_TAIL/OLD_CYCLE is the value listed in the instruction timing tables

NEW_CLOCK is the number of clocks per cycle at the slower speed

Note that many instructions listed as having negative tails are change-of-flow instructions and that the bus speed used in the calculation is that of the new instruction stream.

5.8.2 Instruction Stream Timing Examples

The following programming examples provide a detailed examination of timing effects. In all examples, the memory access is from external synchronous memory, the bus is idle, and the instruction pipeline is full at the start.

5.8.2.1 TIMING EXAMPLE 1—EXECUTION OVERLAP. Figure 5-37 illustrates execution overlap caused by the bus controller's completion of bus cycles while the sequencer is calculating the next EA. One clock is saved between instructions since that is the minimum time of the individual head and tail numbers.

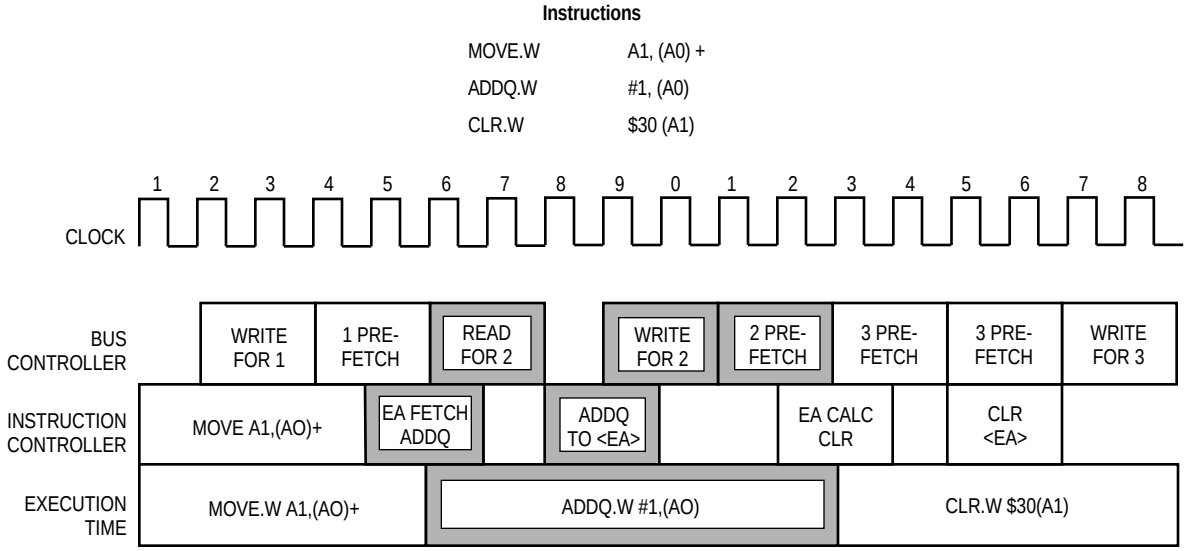


Figure 5-37. Example 1—Instruction Stream

5.8.2.2 TIMING EXAMPLE 2—BRANCH INSTRUCTIONS. Example 2 shows what happens when a branch instruction is executed for both the taken and not-taken cases. (see Figures 5-38 and 5-39). The instruction stream is for a simple limit check with the variable already in a data register.

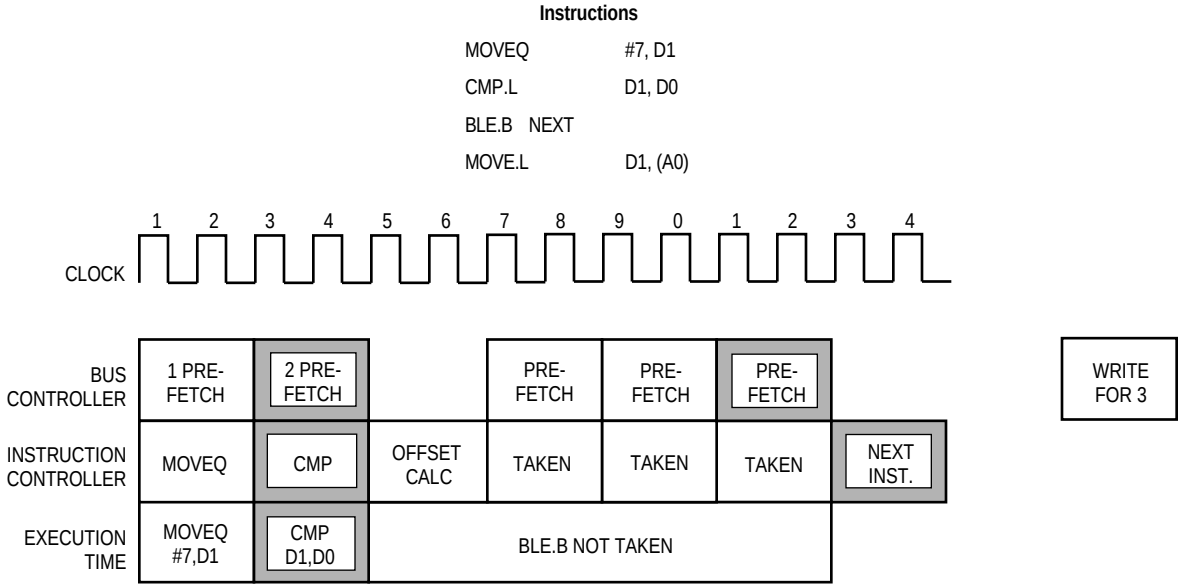


Figure 5-38. Example 2—Branch Taken

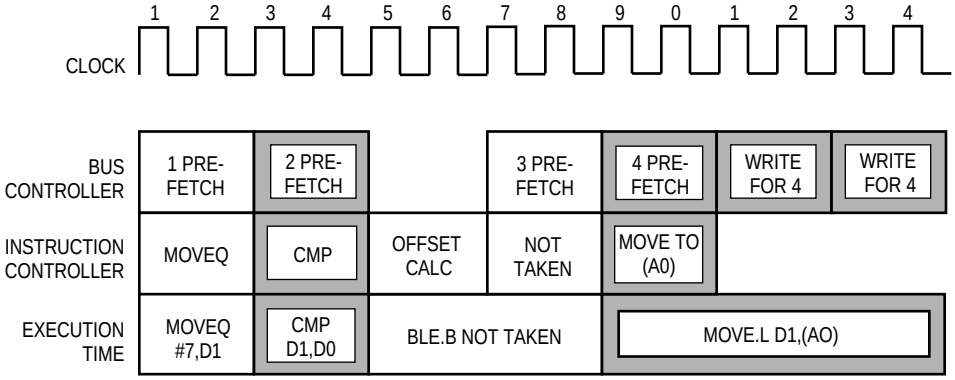


Figure 5-39. Example 2—Branch Not Taken

5.8.2.3 TIMING EXAMPLE 3—NEGATIVE TAILS. This example (see Figure 5-40) shows how to use negative tail figures for branches and other change-of-flow instructions. In this example, bus speed is assumed to be four clocks per access. Instruction three is at the branch destination.

Although the CPU32+ has a two-word instruction pipeline, internal delay causes minimum branch instruction time to be three bus cycles. The negative tail is a reminder that an extra two clocks are available for prefetching a third word on a fast bus; on a slower bus, there is no extra time for the third word.

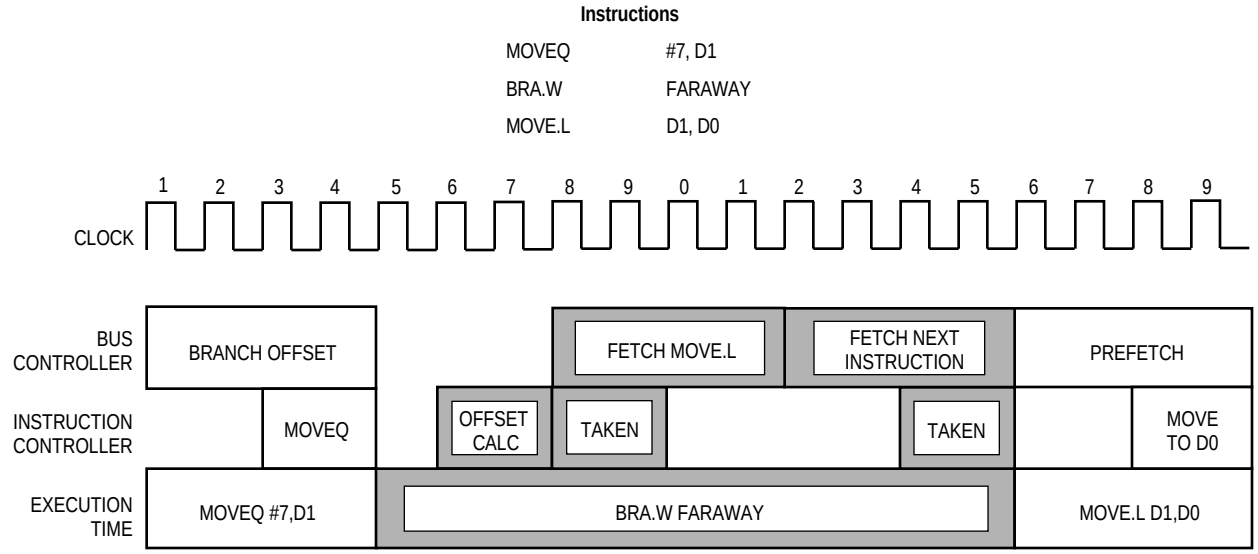


Figure 5-40. Example 3—Branch Negative Tail

Example 3 illustrates three different aspects of instruction time calculation:

1. The branch instruction does not attempt to prefetch beyond the minimum number of words needed for itself.
2. The negative tail allows execution to begin sooner than a three-word pipeline would allow.
3. There is a one-clock delay due to late arrival of the displacement at the CPU.

Only changes of flow require negative tail calculation, but the concept can be generalized to any instruction—only two words are required to be in the pipeline, but up to three words may be present. When there is an opportunity for an extra prefetch, it is made. A prefetch to replace an instruction can begin ahead of the instruction, resulting in a faster processor.

5.8.3 Instruction Timing Tables

The following assumptions apply to the times shown in the subsequent tables:

1. A 16-bit data bus is used for all memory accesses (CPU32+ in 16-bit mode).
2. Memory access times are based on two-clock bus cycles with no wait states.
3. The instruction pipeline is full at the beginning of the instruction and is refilled by the end of the instruction.

Three values are listed for each instruction and addressing mode:

Head: The number of cycles available at the beginning of an instruction to complete a previous instruction write or to perform a prefetch.

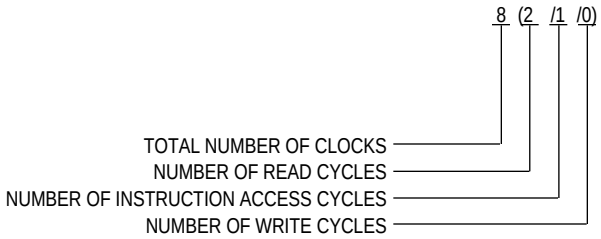
Tail: The number of cycles an instruction uses to complete a write.

Cycles: Four numbers per entry, three contained in parentheses. The outer number is the minimum number of cycles required for the instruction to complete. Numbers within the parentheses represent the number of bus accesses performed by the instruction. The first number is the number of operand read accesses performed by the instruction. The second number is the number of instruction fetches performed by the instruction, including all prefetches that keep the instruction and the instruction pipeline filled. The third number is the number of write accesses performed by the instruction.

As an example, consider an ADD.L (12, A3, D7.W ∗ 4), D2 instruction.

5.8.3.5 Arithmetic/Logic Instructions shows that the instruction has a head = 0, a tail = 0, and cycles = 2 (0/1/0). However, in indexed address register indirect addressing mode, additional time is required to fetch the EA. **5.7.3.1 Fetch Effective Address** gives addressing mode data. For (d_8, An, Xn.Sz ∗ Scale), head = 4, tail = 2, cycles = 8 (2/1/0). Because this example is for a long access and the fetch EA table lists data for word accesses, add two clocks to the tail and to the number of cycles (“X” in table notation) to obtain head = 4, tail = 4, cycles = 10 (2/1/0).

Assuming that no trailing write exists from the previous instruction, EA calculation requires six clocks. Replacement fetch for the EA occurs during these six clocks, leaving a head of four. If there is no time in the head to perform a prefetch due to a previous trailing write, then additional time to perform the prefetches must be allotted in the middle of the instruction or after the tail.



The total number of clocks for bus activity is as follows:

$$(2 \text{ Reads} \times 2 \text{ Clocks/Read}) + (1 \text{ Instruction Access} \times 2 \text{ Clocks/Access}) + (0 \text{ Writes} \times 2 \text{ Clocks/Write}) = 6 \text{ Clocks of Bus Activity}$$

The number of internal clocks (not overlapped by bus activity) is as follows:

$$10 \text{ Clocks Total} - 6 \text{ Clocks Bus Activity} = 4 \text{ Internal Clocks}$$

Memory read requires two bus cycles at two clocks each. This read time, implied in the tail figure for the EA, cannot be overlapped with the instruction because the instruction has a head of zero. An additional two clocks are required for the ADD instruction itself. The total is $6 + 4 + 2 = 12$ clocks. If bus cycles take more time (i.e., the memory is off-chip), add an appropriate number of clocks to each memory access.

The instruction sequence MOVE.L D0, (A0) followed by LSL.L #7, D2 provides an example of overlapped execution. The MOVE instruction has a head of zero and a tail of four because it is a long write. The LSL instruction has a head of four. The trailing write from the MOVE overlaps the LSL head completely. Thus, the two-instruction sequence has a head of zero, a tail of zero, and a total execution of 8 rather than 12 clocks.

General observations regarding calculation of execution time are as follows:

- Any time the number of bus cycles is listed as "X", substitute a value of one for byte and word cycles and a value of two for long cycles. For long bus cycles, usually add a value of two to the tail.
- The time calculated for an instruction on a three-clock (or longer) bus is usually longer than the actual execution time. All times shown are for two-clock bus cycles.
- If the previous instruction has a negative tail, then a prefetch for the current instruction can begin during the execution of that previous instruction.
- Certain instructions requiring an immediate extension word (immediate word EA, absolute word EA, address register indirect with displacement EA, conditional branches with word offsets, bit operations, LPSTOP, TBL, MOVEM, MOVEC, MOVES, MOVEP, MUL.L, DIV.L, CHK2, CMP2, and DBcc) are not permitted to begin until the extension word has been in the instruction pipeline for at least one cycle. This does not apply to long offsets or displacements.

5.8.3.1 FETCH EFFECTIVE ADDRESS. The fetch EA table indicates the number of clock periods needed for the processor to calculate and fetch the specified EA. The total number of clock cycles is outside the parentheses. The numbers inside parentheses (r/p/w) are included in the total clock cycle number. All timing data assumes two-clock reads and writes.

Instruction	Head	Tail	Cycles	Notes
Dn	–	–	0(0/0/0)	–
An	–	–	0(0/0/0)	–
(An)	1	1	3(X/0/0)	1
(An)+	1	1	3(X/0/0)	1
–(An)	2	2	4(X/0/0)	1
(d_{16},An) or (d_{16},PC)	1	3	5(X/1/0)	1,3
(xxx).W	1	3	5(X/1/0)	1
(xxx).L	1	5	7(X/2/0)	1
#⟨data⟩.B	1	1	3(0/1/0)	1
#⟨data⟩.W	1	1	3(0/1/0)	1
#⟨data⟩.L	1	3	5(0/2/0)	1
(d_8,An,Xn.Sz × Sc) or (d_8,PC,Xn.Sz × Sc)	4	2	8(X/1/0)	1,2,3,4
(0) (All Suppressed)	2	2	6(X/1/0)	1,4
(d_{16})	1	3	7(X/2/0)	1,4
(d_{32})	1	5	9(X/3/0)	1,4
(An)	1	1	5(X/1/0)	1,2,4
(Xm.Sz × Sc)	4	2	8(X/1/0)	1,2,4
(An,Xm.Sz × Sc)	4	2	8(X/1/0)	1,2,3,4
(d_{16},An) or (d_{16},PC)	1	3	7(X/2/0)	1,3,4
(d_{32},An) or (d_{32},PC)	1	5	9(X/3/0)	1,3,4
(d_{16},An,Xm) or (d_{16},PC,Xm)	2	2	8(X/2/0)	1,3,4
(d_{32},An,Xm) or (d_{32},PC,Xm)	1	3	9(X/3/0)	1,3,4
(d_{16},An,Xm.Sz × Sc) or (d_{16},PC,Xm.Sz × Sc)	2	2	8(X/2/0)	1,2,3,4
(d_{32},An,Xm.Sz × Sc) or (d_{32},PC,Xm.Sz × Sc)	1	3	9(X/3/0)	1,2,3,4

X = There is one bus cycle for byte and word operands and two bus cycles for long-word operands. For long-word bus cycles, add two clocks to the tail and to the number of cycles.

NOTES:

1. The read of the EA and replacement fetches overlap the head of the operation by the amount specified in the tail.
2. Size and scale of the index register do not affect execution time.
3. The PC may be substituted for the base address register An.
4. When adjusting the prefetch time for slower buses, extra clocks may be subtracted from the head until the head reaches zero, at which time additional clocks must be added to both the tail and cycle counts.
5. Timing is calculated with the CPU32+ in 16-bit mode.

5.8.3.2 CALCULATE EFFECTIVE ADDRESS. The calculate EA table indicates the number of clock periods needed for the processor to calculate a specified EA. The timing is equivalent to fetch EA except there is no read cycle. The tail and cycle time are reduced by the amount of time the read would occupy. The total number of clock cycles is outside the parentheses. The numbers inside parentheses (r/p/w) are included in the total clock cycle number. All timing data assumes two-clock reads and writes.

Instruction	Head	Tail	Cycles	Notes
Dn	–	–	0(0/0/0)	–
An	–	–	0(0/0/0)	–
(An)	1	0	2(0/0/0)	–
(An)+	1	0	2(0/0/0)	–
–(An)	2	0	2(0/0/0)	–
(d_{16},An) or (d_{16},PC)	1	1	3(0/1/0)	1,3
(xxx).W	1	1	3(0/1/0)	1
(xxx).L	1	3	5(0/2/0)	1
(d_8,An,Xn.Sz × Sc) or (d_8,PC,Xn.Sz × Sc)	4	0	6(0/1/0)	2,3,4
(0) (All Suppressed)	2	0	4(0/1/0)	4
(d_{16})	1	1	5(0/2/0)	1,4
(d_{32})	1	3	7(0/3/0)	1,4
(An)	1	0	4(0/1/0)	4
(Xm.Sz × Sc)	4	0	6(0/1/0)	2,4
(An,Xm.Sz × Sc)	4	0	6(0/1/0)	2,4
(d_{16},An) or (d_{16},PC)	1	1	5(0/2/0)	1,3,4
(d_{32},An) or (d_{32},PC)	1	3	7(0/3/0)	1,3,4
(d_{16},An,Xm) or (d_{16},PC,Xm)	2	0	6(0/2/0)	3,4
(d_{32},An,Xm) or (d_{32},PC,Xm)	1	1	7(0/3/0)	1,3,4
(d_{16},An,Xm.Sz × Sc) or (d_{16},PC,Xm.Sz × Sc)	2	0	6(0/2/0)	2,3,4
(d_{32},An,Xm.Sz × Sc) or (d_{32},PC,Xm.Sz × Sc)	1	1	7(0/3/0)	1,2,3,4

NOTES:

1. Replacement fetches overlap the head of the operation by the amount specified in the tail.
2. Size and scale of the index register do not affect execution time.
3. The PC may be substituted for the base address register An.
4. When adjusting the prefetch time for slower buses, extra clocks may be subtracted from the head until the head reaches zero, at which time additional clocks must be added to both the tail and cycle counts.
5. Timing is calculated with the CPU32+ in 16-bit mode

5.8.3.3 MOVE INSTRUCTION. The MOVE instruction table indicates the number of clock periods needed for the processor to calculate the destination EA and to perform a MOVE or MOVEA instruction. For entries with CEA or FEA, refer to the appropriate table to calculate that portion of the instruction time.

Destination EAs are divided by their formats. The total number of clock cycles is outside the parentheses. The numbers inside parentheses (r/p/w) are included in the total clock cycle number. All timing data assumes two-clock reads and writes.

When using this table, begin at the top and move downward. Use the first entry that matches both source and destination addressing modes.

Instruction	Head	Tail	Cycles
MOVE Rn, Rn	0	0	2(0/1/0)
MOVE ⟨FEA⟩, Rn	0	0	2(0/1/0)
MOVE Rn, (Am)	0	2	4(0/1/X)
MOVE Rn, (Am)+	1	1	5(0/1/X)
MOVE Rn, –(Am)	2	2	6(0/1/X)
MOVE Rn, ⟨CEA⟩	1	3	5(0/1/X)
MOVE ⟨FEA⟩, (An)	2	2	6(0/1/X)
MOVE ⟨FEA⟩, (An)+	2	2	6(0/1/X)
MOVE ⟨FEA⟩, –(An)	2	2	6(0/1/X)
MOVE #, ⟨CEA⟩	2	2	6(0/1/X)*
MOVE ⟨CEA⟩, ⟨FEA⟩	2	2	6(0/1/X)

X = There is one bus cycle for byte and word operands and two bus cycles for long-word operands. For long-word bus cycles, add two clocks to the tail and to the number of cycles.

Timing is calculated with the CPU32+ in 16-bit mode.

* = An # fetch EA time must be added for this instruction: ⟨FEA⟩+⟨CEA⟩+⟨OPER⟩

NOTE: For instructions not explicitly listed, use the MOVE ⟨CEA⟩, ⟨FEA⟩ entry. The source EA is calculated by the calculate EA table, and the destination EA is calculated by the fetch EA table, even though the bus cycle is for the source EA.

5.8.3.4 SPECIAL-PURPOSE MOVE INSTRUCTION. The special-purpose MOVE instruction table indicates the number of clock periods needed for the processor to fetch, calculate, and perform the special-purpose MOVE operation on control registers or a specified EA. Footnotes indicate when to account for the appropriate EA times. The total number of clock cycles is outside the parentheses. The numbers inside parentheses (r/p/w) are included in the total clock cycle number. All timing data assumes two-clock reads and writes.

Instruction		Head	Tail	Cycles
EXG	Rn, Rm	2	0	4(0/1/0)
MOVEC	Cr, Rn	10	0	14(0/2/0)
MOVEC	Rn, Cr	12	0	14-16(0/1/0)
MOVE	CCR, Dn	2	0	4(0/1/0)
MOVE	CCR, ⟨CEA⟩	0	2	4(0/1/1)
MOVE	Dn, CCR	2	0	4(0/1/0)
MOVE	⟨FEA⟩, CCR	0	0	4(0/1/0)
MOVE	SR, Dn	2	0	4(0/1/0)
MOVE	SR, ⟨CEA⟩	0	2	4(0/1/1)
MOVE	Dn, SR	4	–2	10(0/3/0)
MOVE	⟨FEA⟩, SR	0	–2	10(0/3/0)
MOVEM.W	⟨CEA⟩, RL	1	0	$8 + n \times 4(n + 1, 2, 0)^*$
MOVEM.W	RL, ⟨CEA⟩	1	0	$8 + n \times 4(0, 2, n)^*$
MOVEM.L	⟨CEA⟩, RL	1	0	$12 + n \times 4(2n + 2, 2, 0)$
MOVEM.L	RL, ⟨CEA⟩	1	2	$10 + n \times 4(0, 2, 2n)$
MOVEP.W	Dn, (d_{16}, An)	2	0	10(0/2/2)
MOVEP.W	(d_{16}, An), Dn	1	2	11(2/2/0)
MOVEP.L	Dn, (d_{16}, An)	2	0	14(0/2/4)
MOVEP.L	(d_{16}, An), Dn	1	2	19(4/2/0)
MOVES (Save)	⟨CEA⟩, Rn	1	1	3(0/1/0)
MOVES (Op)	⟨CEA⟩, Rn	7	1	11(X/1/0)
MOVES (Save)	Rn, ⟨CEA⟩	1	1	3(0/1/0)
MOVES (Op)	Rn, ⟨CEA⟩	9	2	12(0/1/X)
MOVE	USP, An	0	0	2(0/1/0)
MOVE	An, USP	0	0	2(0/1/0)
SWAP	Dn	4	0	6(0/1/0)

X = There is one bus cycle for byte and word operands and two bus cycles for long operands. For long bus cycles, add two clocks to the tail and to the number of cycles.

* = Each bus cycle may take up to four clocks without increasing total execution time.

Cr = Control registers USP, VBR, SFC, and DFC

n = Number of registers to transfer

RL = Register List

< = Maximum time (certain data or mode combinations may execute faster).

NOTES:

1. The MOVES instruction has an additional save step that other instructions do not have. To calculate the total instruction time, calculate the save, the EA, and the operation execution times, and combine in the order listed, using the equations given in **5.8.1.6 Instruction Execution Time Calculation**.
2. Timing is calculated with the CPU32+ in 16-bit mode.

5.8.3.5 ARITHMETIC/LOGIC INSTRUCTIONS. The arithmetic/logic instruction table indicates the number of clock periods needed to perform the specified arithmetic/logical instruction using the specified addressing mode. Footnotes indicate when to account for the appropriate EA times. The total number of clock cycles is outside the parentheses. The numbers inside parentheses (r/p/w) are included in the total clock cycle number. All timing data assumes two-clock reads and writes.

Instruction		Head	Tail	Cycles
ADD(A)	Rn, Rm	0	0	2(0/1/0)
ADD(A)	⟨FEA⟩, Rn	0	0	2(0/1/0)
ADD	Dn, ⟨FEA⟩	0	3	5(0/1/x)
AND	Dn, Dm	0	0	2(0/1/0)
AND	⟨FEA⟩, Dn	0	0	2(0/1/0)
AND	Dn, ⟨FEA⟩	0	3	5(0/1/x)
EOR	Dn, Dm	0	0	2(0/1/0)
EOR	Dn, ⟨FEA⟩	0	3	5(0/1/x)
OR	Dn, Dm	0	0	2(0/1/0)
OR	⟨FEA⟩, Dn	0	0	2(0/1/0)
OR	Dn, ⟨FEA⟩	0	3	5(0/1/x)
SUB(A)	Rn, Rm	0	0	2(0/1/0)
SUB(A)	⟨FEA⟩, Rn	0	0	2(0/1/0)
SUB	Dn, ⟨FEA⟩	0	3	5(0/1/x)
CMP(A)	Rn, Rm	0	0	2(0/1/0)
CMP(A)	⟨FEA⟩, Rn	0	0	2(0/1/0)
CMP2 (Save) *	⟨FEA⟩, Rn	1	1	3(0/1/0)
CMP2 (Op)	⟨FEA⟩, Rn	2	0	16-18(X/1/0)
MUL(su).W	⟨FEA⟩, Dn	0	0	26(0/1/0)
MUL(su).L (Save) *	⟨FEA⟩, Dn	1	1	3(0/1/0)
MUL(su).L (Op)	⟨FEA⟩, Dl	2	0	46-52(0/1/0)
MUL(su).L (Op)	⟨FEA⟩, Dn:Dl	2	0	46(0/1/0)
DIVU.W	⟨FEA⟩, Dn	0	0	32(0/1/0)
DIVS.W	⟨FEA⟩, Dn	0	0	42(0/1/0)
DIVU.L (Save)*	⟨FEA⟩, Dn	1	1	3(0/1/0)
DIVU.L (Op)	⟨FEA⟩, Dn	2	0	<46(0/1/0)
DIVS.L (Save) *	⟨FEA⟩, Dn	1	1	3(0/1/0)
DIVS.L (Op)	⟨FEA⟩, Dn	2	0	<62(0/1/0)
TBL(su)	Dn:Dm, Dp	26	0	28-30(0/2/0)
TBL(su) (Save) *	⟨CEA⟩, Dn	1	1	3(0/1/0)
TBL(su) (Op)	⟨CEA⟩, Dn	6	0	33-35(2X/1/0)
TBLSN	Dn:Dm, Dp	30	0	30-34(0/2/0)
TBLSN (Save)*	⟨CEA⟩, Dn	1	1	3(0/1/0)
TBLSN (Op)	⟨CEA⟩, Dn	6	0	35-39(2X/1/0)
TBLUN	Dn:Dm, Dp	30	0	34-40(0/2/0)
TBLUN (Save)*	⟨CEA⟩, Dn	1	1	3(0/1/0)
TBLUN (Op)	⟨CEA⟩, Dn	6	0	39-45(2X/1/0)

X = There is one bus cycle for byte and word operands and two bus cycles for long operands. For long bus cycles, add two clocks to the tail and to the number of cycles.

Timing is calculated with the CPU32+ in 16-bit mode.

< = Maximum time (certain data or mode combinations may execute faster).

su = The execution time is identical for signed or unsigned operands.

* = These instructions have an additional save operation that other instructions do not have. To calculate total instruction time, calculate save, ⟨ea⟩, and operation execution times, then combine in the order listed, using equations in **5.8.1.6 Instruction Execution Time Calculations**. A save operation is not run for long-word divide and multiply instructions when ⟨FEA⟩ = Dn.

5.8.3.6 IMMEDIATE ARITHMETIC/LOGIC INSTRUCTIONS. The immediate arithmetic/logic instruction table indicates the number of clock periods needed for the processor to fetch the source immediate data value and to perform the specified arithmetic/logic instruction using the specified addressing mode. Footnotes indicate when to account for the appropriate fetch effective or fetch immediate EA times. The total number of clock cycles is outside the parentheses. The numbers inside parentheses (r/p/w) are included in the total clock cycle number. All timing data assumes two-clock reads and writes.

Instruction		Head	Tail	Cycles
MOVEQ	#, Dn	0	0	2(0/1/0)
ADDQ	#, Rn	0	0	2(0/1/0)
ADDQ	#, ⟨FEA⟩	0	3	5(0/1/x)
SUBQ	#, Rn	0	0	2(0/1/0)
SUBQ	#, ⟨FEA⟩	0	3	5(0/1/x)
ADDI	#, Rn	0	0	2(0/1/0)*
ADDI	#, ⟨FEA⟩	0	3	5(0/1/x)*
ANDI	#, Rn	0	0	2(0/1/0)*
ANDI	#, ⟨FEA⟩	0	3	5(0/1/x)*
EORI	#, Rn	0	0	2(0/1/0)*
EORI	#, ⟨FEA⟩	0	3	5(0/1/x)*
ORI	#, Rn	0	0	2(0/1/0)*
ORI	#, ⟨FEA⟩	0	3	5(0/1/x)*
SUBI	#, Rn	0	0	2(0/1/0)*
SUBI	#, ⟨FEA⟩	0	3	5(0/1/x)*
CMPI	#, Rn	0	0	2(0/1/0)*
CMPI	#, ⟨FEA⟩	0	3	5(0/1/x)*

X = There is one bus cycle for byte and word operands and two bus cycles for long-word operands. For long-word bus cycles, add two clocks to the tail and to the number of cycles.

Timing is calculated with the CPU32+ in 16-bit mode.

* = An # fetch EA time must be added for this instruction: ⟨FEA⟩ + ⟨FEA⟩ + ⟨OPER⟩

5.8.3.7 BINARY-CODED DECIMAL AND EXTENDED INSTRUCTIONS. The BCD and extended instruction table indicates the number of clock periods needed for the processor to perform the specified operation using the specified addressing mode. No additional tables are needed to calculate total effective execution time for these instructions. The total number of clock cycles is outside the parentheses. The numbers inside parentheses (r/p/w) are included in the total clock cycle number. All timing data assumes two-clock reads and writes.

Instruction		Head	Tail	Cycles
ABCD	Dn, Dm	2	0	4(0/1/0)
ABCD	–(An), –(Am)	2	2	12(2/1/1)
SBCD	Dn, Dm	2	0	4(0/1/0)
SBCD	–(An), –(Am)	2	2	12(2/1/1)
ADDX	Dn, Dm	0	0	2(0/1/0)
ADDX	–(An), –(Am)	2	2	10(2/1/1)
SUBX	Dn, Dm	0	0	2(0/1/0)
SUBX	–(An), –(Am)	2	2	10(2/1/1)
CMPM	(An)+, (Am)+	1	0	8(2/1/0)

5.8.3.8 SINGLE OPERAND INSTRUCTIONS. The single operand instruction table indicates the number of clock periods needed for the processor to perform the specified operation using the specified addressing mode. The total number of clock cycles is outside the parentheses. The numbers inside parentheses (r/p/w) are included in the total clock cycle number. All timing data assumes two-clock reads and writes.

Instruction		Head	Tail	Cycles
CLR	Dn	0	0	2(0/1/0)
CLR	⟨CEA⟩	0	2	4(0/1/X)
NEG	Dn	0	0	2(0/1/0)
NEG	⟨FEA⟩	0	3	5(0/1/X)
NEGX	Dn	0	0	2(0/1/0)
NEGX	⟨FEA⟩	0	3	5(0/1/X)
NOT	Dn	0	0	2(0/1/0)
NOT	⟨FEA⟩	0	3	5(0/1/X)
EXT	Dn	0	0	2(0/1/0)
NBCD	Dn	2	0	4(0/1/0)
NBCD	⟨FEA⟩	0	2	6(0/1/1)
Scc	Dn	2	0	4(0/1/0)
Scc	⟨CEA⟩	2	2	6(0/1/1)
TAS	Dn	4	0	6(0/1/0)
TAS	⟨CEA⟩	1	0	10(0/1/1)
TST	⟨FEA⟩	0	0	2(0/1/0)

X = There is one bus cycle for byte and word operands and two bus cycles for long-word operands. For long-word bus cycles, add two clocks to the tail and to the number of cycles.

Timing is calculated with the CPU32+ in 16-bit mode

5.8.3.9 SHIFT/ROTATE INSTRUCTIONS. The shift/rotate instruction table indicates the number of clock periods needed for the processor to perform the specified operation on the given addressing mode. Footnotes indicate when to account for the appropriate EA times. The number of bits shifted does not affect the execution time, unless noted. The total number of clock cycles is outside the parentheses. The numbers inside parentheses (r/p/w) are included in the total clock cycle number. All timing data assumes two-clock reads and writes.

Instruction		Head	Tail	Cycles	Note
LSd	Dn, Dm	–2	0	(0/1/0)	1
LSd	#, Dm	4	0	6(0/1/0)	—
LSd	⟨FEA⟩	0	2	6(0/1/1)	—
ASd	Dn, Dm	–2	0	(0/1/0)	1
ASd	#, Dm	4	0	6(0/1/0)	—
ASd	⟨FEA⟩	0	2	6(0/1/1)	—
ROd	Dn, Dm	–2	0	(0/1/0)	1
ROd	#, Dm	4	0	6(0/1/0)	—
ROd	⟨FEA⟩	0	2	6(0/1/1)	—
ROXd	Dn, Dm	–2	0	(0/1/0)	2
ROXd	#, Dm	–2	0	(0/1/0)	3
ROXd	⟨FEA⟩	0	2	6(0/1/1)	—

d = Direction (left or right)

NOTES:

1. Head and cycle times can be derived from the following table or calculated as follows:
   Max (3 + (n/4) + mod(n,4) + mod (((n/4) + mod (n,4) + 1,2), 6)
2. Head and cycle times are calculated as follows: (count ≤ 63): max (3 + n + mod (n + 1,2), 6).
3. Head and cycle times are calculated as follows: (count ≤ 8): max (2 + n + mod (n,2), 6).
4. Timing is calculated with the CPU32+ in 16-bit mode.

Clocks	Shift Counts									
6	0	1	2	3	4	5	6	8	9	12
8	7	10	11	13	14	16	17	20		
10	15	18	19	21	22	24	25	28		
12	23	26	27	29	30	32	33	36		
14	31	34	35	37	38	40	41	44		
16	39	42	43	45	46	48	49	52		
18	47	50	51	53	54	56	57	60		
20	55	58	59	61	62					
22	63									

5.8.3.10 BIT MANIPULATION INSTRUCTIONS. The bit manipulation instruction table indicates the number of clock periods needed for the processor to perform the specified operation on the given addressing mode. The total number of clock cycles is outside the parentheses. The numbers inside parentheses (r/p/w) are included in the total clock cycle number. All timing data assumes two-clock reads and writes.

Instruction	Head	Tail	Cycles
BCHG #, Dn	2	0	6(0/2/0)*
BCHG Dn, Dm	4	0	6(0/1/0)
BCHG #, ⟨FEA⟩	1	2	8(0/2/1)*
BCHG Dn, ⟨FEA⟩	2	2	8(0/1/1)
BCLR #, Dn	2	0	6(0/2/0)*
BCLR Dn, Dm	4	0	6(0/1/0)
BCLR #, ⟨FEA⟩	1	2	8(0/2/1)*
BCLR Dn, ⟨FEA⟩	2	2	8(0/1/1)
BSET #, Dn	2	0	6(0/2/0)*
BSET Dn, Dm	4	0	6(0/1/0)
BSET #, ⟨FEA⟩	1	2	8(0/2/1)*
BSET Dn, ⟨FEA⟩	2	2	8(0/1/1)
BTST #, Dn	2	0	4(0/2/0)*
BTST Dn, Dm	2	0	4(0/1/0)
BTST #, ⟨FEA⟩	1	0	4(0/2/0)*
BTST Dn, ⟨FEA⟩	2	0	8(0/1/0)

* = An # fetch EA time must be added for this instruction: ⟨FEA⟩ + ⟨FEA⟩ + ⟨OPER⟩

Timing is calculated with the CPU32+ in 16-bit mode.

5.8.3.11 CONDITIONAL BRANCH INSTRUCTIONS. The conditional branch instruction timing table indicates the number of clock periods needed for the processor to perform the specified branch on the given branch size, with complete execution times given. No additional tables are needed to calculate total effective execution time for these instructions. The total number of clock cycles is outside the parentheses. The numbers inside parentheses (r/p/w) are included in the total clock cycle number. All timing data assumes two-clock reads and writes.

Instruction		Head	Tail	Cycles
Bcc	(taken)	2	–2	8(0/2/0)
Bcc.B	(not taken)	2	0	4(0/1/0)
Bcc.W	(not taken)	0	0	4(0/2/0)
Bcc.L	(not taken)	0	0	6(0/3/1)
DBcc	(T, not taken)	1	1	4(0/2/0)
DBcc	(F, –1, not taken)	2	0	6(0/2/0)
DBcc	(F, not –1, taken)	6	–2	10(0/2/0)
DBcc	(T, not taken)	4	0	6(0/1/0)*
DBcc	(F, –1, not taken)	6	0	8(0/1/0)*
DBcc	(F, not –1, taken)	6	0	10(0/0/0)*

* = In loop mode

Timing is calculated with the CPU32+ in 16-bit mode.

5.8.3.12 CONTROL INSTRUCTIONS. The control instruction table indicates the number of clock periods needed for the processor to perform the specified operation on the given addressing mode. Footnotes indicate when to account for the appropriate EA times. The total number of clock cycles is outside the parentheses. The numbers inside parentheses (r/p/w) are included in the total clock cycle number. All timing data assumes two-clock reads and writes.

Instruction		Head	Tail	Cycles
ANDI	#, SR	0	–2	12(0/2/0)
EORI	#, SR	0	–2	12(0/2/0)
ORI	#, SR	0	–2	12(0/2/0)
ANDI	#, CCR	2	0	6(0/2/0)
EORI	#, CCR	2	0	6(0/2/0)
ORI	#, CCR	2	0	6(0/2/0)
BSR.B		3	–2	13(0/2/2)
BSR.W		3	–2	13(0/2/2)
BSR.L		1	–2	13(0/2/2)
CHK	⟨FEA⟩, Dn (no ex)	2	0	8(0/1/0)
CHK	⟨FEA⟩, Dn (ex)	2	–2	42(2/2/6)
CHK2 (Save)	⟨FEA⟩, Dn (no ex)	1	1	3(0/1/0)
CHK2 (Op)	⟨FEA⟩, Dn (no ex)	2	0	18(X/0/0)
CHK2 (Save)	⟨FEA⟩, Dn (ex)	1	1	3(0/1/0)
CHK2 (Op)	⟨FEA⟩, Dn (ex)	2	–2	52(X + 2/1/6)
JMP	⟨CEA⟩	0	–2	6(0/2/0)
JSR	⟨CEA⟩	3	–2	13(0/2/2)
LEA	⟨CEA⟩, An	0	0	2(0/1/0)
LINK.W	An, #	2	0	10(0/2/2)
LINK.L	An, #	0	0	10(0/3/2)
NOP		0	0	2(0/1/0)
PEA	⟨CEA⟩	0	0	8(0/1/2)
RTD	#	1	–2	12(2/2/0)
RTR		1	–2	14(3/2/0)
RTS		1	–2	12(2/2/0)
UNLK	An	1	0	9(2/1/0)

X = There is one bus cycle for byte and word operands and two bus cycles for long-word operands. For long-word bus cycles, add two clocks to the tail and to the number of cycles.

Timing is calculated with the CPU32+ in 16-bit mode.

NOTE: The CHK2 instruction involves a save step that other instructions do not have. To calculate the total instruction time, calculate the save, the EA, and the operation execution times; then combine in the order listed using the equations given in **5.8.1.6 Instruction Execution Time Calculation**.

5.8.3.13 EXCEPTION-RELATED INSTRUCTIONS AND OPERATIONS. The exception-related instructions and operations table indicates the number of clock periods needed for the processor to perform the specified exception-related actions. No additional tables are needed to calculate total effective execution time for these instructions. The total number of clock cycles is outside the parentheses. The numbers inside parentheses (r/p/w) are included in the total clock cycle number. All timing data assumes two-clock reads and writes.

Instruction	Head	Tail	Cycles
BKPT (Acknowledged)	0	0	14(1/0/0)
BKPT (Bus Error)	0	–2	35(3/2/4)
Breakpoint (Acknowledged)	0	0	10(1/0/0)
Breakpoint (Bus Error)	0	–2	42(3/2/6)
Interrupt	0	–2	30(3/2/4)*
RESET	0	0	518(0/1/0)
STOP	2	0	12(0/1/0)
LPSTOP	3	–2	25(0/3/1)
Divide-by-Zero	0	–2	36(2/2/6)
Trace	0	–2	36(2/2/6)
TRAP #	4	–2	29(2/2/4)
ILLEGAL	0	–2	25(2/2/4)
A-line	0	–2	25(2/2/4)
F-line (First word illegal)	0	–2	25(2/2/4)
F-line (Second word illegal) ea = Rn	1	–2	31(2/3/4)
F-line (Second word illegal) ea ≠ Rn (Save)	1	1	3(0/1/0)
F-line (Second word illegal) ea ≠ Rn (Op)	4	–2	29(2/2/4)
Privileged	0	–2	25(2/2/4)
TRAPcc (trap)	2	–2	38(2/2/6)
TRAPcc (no trap)	2	0	4(0/1/0)
TRAPcc.W (trap)	2	–2	38(2/2/6)
TRAPcc.W (no trap)	0	0	4(0/2/0)
TRAPcc.L (trap)	0	–2	38(2/2/6)
TRAPcc.L (no trap)	0	0	6(0/3/0)
TRAPV (trap)	2	–2	38(2/2/6)
TRAPV (no trap)	2	0	4(0/1/0)

* = Minimum interrupt acknowledge cycle time is assumed to be three clocks.

Timing is calculated with the CPU32+ in 16-bit mode.

NOTE: The F-line (second word illegal) operation involves a save step which other operations do not have. To calculate the total operation time, calculate the save, the calculate EA, and the operation execution times, and combine in the order listed, using the equations given in **5.8.1.6 Instruction Execution Time Calculation**.

5.8.3.14 SAVE AND RESTORE OPERATIONS. The save and restore operations table indicates the number of clock periods needed for the processor to perform the specified state save or return from exception. Complete execution times and stack length are given. No additional tables are needed to calculate total effective execution time for these instructions. The total number of clock cycles is outside the parentheses. The numbers inside parentheses (r/p/w) are included in the total clock cycle number. All timing data assumes two-clock reads and writes.

Instruction	Head	Tail	Cycles
BERR on instruction	0	–2	<58(2/2/12)
BERR on exception	0	–2	48(2/2/12)
RTE (four-word frame)	1	–2	24(4/2/0)
RTE (six-word frame)	1	–2	26(4/2/0)
RTE (BERR on instruction)	1	–2	50(12/12/Y)
RTE (BERR on four-word frame)	1	–2	66(10/2/4)
RTE (BERR on six-word frame)	1	–2	70(12/2/6)

Y = If a bus error occurred during a write cycle, the cycle is rerun by the RTE.

< = Maximum time is indicated (certain data or mode combinations execute faster).

Timing is calculated with the CPU32+ in 16-bit mode.

SECTION 6
QUAD DATA MEMORY MODULE

The quad data memory module (QDMM) is part of the CPU030 and provides 4 Kbytes of general purpose data storage on the intermodule bus (IMB). This section provides a functional description, followed by some general application ideas and a description of the register interface.

6.1 FUNCTIONAL DESCRIPTION

The QDMM consists of four independent 1-Kbyte blocks of static random access memory (SRAM), each of which can be independently mapped to any 1-Kbyte boundary in the 4-Gbyte address range. Each block can be programmed to respond to either supervisor-only or supervisor and user code and data accesses. By restricting a block to supervisor-only accesses, the data or code resources stored in the block (such as system interrupt handlers) can be protected from access by user tasks. Additional protection is provided by the ability to write-protect each block—this can prevent changes to the block when it contains code or fixed data structures.

The QDMM is accessed via the IMB. This bus interface supports accesses by both the CPU32+ and the DMA controller. Since the MC68349 does not support a slave mode, the QDMM is not directly accessible to off-chip bus masters. The QDMM always provides 32 bits of data to the IMB for a read access and accepts byte, word, three-byte, or long-word data for a write access. From the perspective of an IMB master, the access timing is identical to that of an external fast termination bus cycle, and all accesses complete in two clocks. However, no external bus cycle occurs when accessing the QDMM (unless show cycles are enabled). Accesses to the QDMM can occur even when the external bus is arbitrated away.

Since CPU32+ accesses to the QDMM complete in the same number of clocks as accesses to the CIC instruction cache, instruction fetches from the QDMM are implicitly non-cacheable and do not allocate in the instruction cache. This maximizes availability of the cache storage for caching slower external instruction fetches.

QDMM blocks can be mapped anywhere in the address map of the IMB, including address ranges which overlap the SIM49 chip selects or other externally decoded memory. Accesses to the QDMM take priority over external accesses, and $\overline{AS}$ (as well as $\overline{CSx}$ for a matching chip select) do not assert for the overlaid memory. Care should be taken to prevent mapping enabled blocks to the same address range; data returned in this case is undefined. The QDMM blocks also should not overlap any CIC blocks programmed as SRAM, or any portion of the 4-Kbyte module register block mapped by the MC68349 system integration module (SIM49) module base address register (MBAR).

6.2 APPLICATION AREAS

The functionality of the SRAM blocks in the QDMM lends it to a number of possible application areas. One obvious use is for scratchpad random access memory (RAM) to hold frequently used variables. A similar use is as a high-speed buffer for storage of data frames for the DMA controller.

The reprogrammable nature of the QDMM modules aids their use as fast overlay memory for speeding up accesses to slower main memory. One block could overlay the stack space for a task which is stack-intensive, for instance. On completion of the task and deallocation of its corresponding user stack, the associated QDMM memory block could be reassigned to another task.

Although its name (Quad *Data* Memory Module) implies use specifically for data storage, the QDMM SRAM can be used to store code as well. For instance, critical interrupt service routines (ISRs), which require a fast, deterministic execution time, can be copied from slower main memory into the QDMM.

During initial prototype debug, the QDMM memory can serve as a small program and data area for initial test diagnostics downloaded using background debug mode (BDM). A functional core based on the MC68349 can be brought up immediately by verifying correct operation of basic processor resources such as the clock, reset, power and ground connections, default pullups/pulldowns, and BDM interface. Using this initial debug platform with an appropriate debug tool (such as the M68ICD32 controller), code can be downloaded directly to the QDMM to assist debugging of the remainder of the system.

6.3 PROGRAMMING MODEL

Figure 6-1 is a programming model (register map) of the registers in the QDMM. The ADDR (address) column indicates the offset of the register from the address stored in the SIM49 MBAR. The FC (function code) column indicates whether a register is restricted to supervisor access (S) or programmable to exist in either supervisor or user space (S/U).

ADDR	FC	15 — 8 | 7 — 0
C00	S	Module Configuration Register (MCR)
C10	S	QDMM Base Address Register 0 (QBAR0)
C14	S	QDMM Base Address Register 1 (QBAR1)
C18	S	QDMM Base Address Register 2 (QBAR2)
C1C	S	QDMM Base Address Register 3 (QBAR3)

Figure 6-1. Programming Model for the QDMM

For the registers discussed in the following paragraphs, the number in the upper right-hand corner indicates the offset of the register from the address stored in the SIM49 MBAR. The numbers on the top line of the register represent the bit position in the register. The second line contains the mnemonic for the bit. The numbers below the register represent the bit value after a hardware reset. The access privilege is indicated in the lower, right-hand corner.

6.3.1 Module Configuration Register (MCR)

The MCR is a 16-bit register which contains the QDMM configuration information.

MCR $C00

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

Supervisor Only

STP—Stop Enable Bit

The STP bit enables a low-power stop state for the QDMM by allowing the clocks to be stopped locally. When the STP bit is set, accesses to the QDMM array are ignored and the QDMM only responds to an access to its MCR or a reset.

Bits 14–0—Reserved by Motorola

6.3.2 QDMM Base Address Registers (QBAR3–QBAR0)

There are four QDMM base address registers, one for each of the 1-Kbyte blocks of SRAM. The QBARs are 32-bit registers which contain the base address and configuration for each block.

QBAR $C10, $C14, $C18, $C1C

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
BASE ADDRESS (UPPER)															
RESET:															
U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
BASE ADDRESS (LOWER)						0	0	0	0	0	0	0	S/U	WP	V
RESET:															
U	U	U	U	U	U	0	0	0	0	0	0	0	U	U	0

Supervisor/User

Base Address—

Bits 31–10 contain the most significant bits of the base address of the corresponding 1-Kbyte SRAM block. The blocks may be mapped to any address on 1-Kbyte boundaries. The least significant 10 bits of the base address are considered to be zero.

S/U—Supervisor/User

This bit determines whether the corresponding block will respond to supervisor space (FC = 6, 5) or supervisor and user space (FC = 1, 2) accesses.

1 = This block responds to supervisor and user space accesses.
0 = This block responds only to supervisor space accesses.

WP—Write Protect

This bit determines if the corresponding block is write protected.

1 = A write access to this block returns a bus error.
0 = Writes are permitted to this block.

V—Valid

1 = The base address and configuration data for the corresponding block are valid and accesses to the block are permitted.
0 = The corresponding block is not accessible.

This bit is cleared at reset to allow configuration of the QBAR before accesses are permitted to the corresponding block.

SECTION 7
DMA CONTROLLER MODULE

The direct memory access (DMA) controller module provides for high-speed transfer capability to/from an external peripheral or for memory-to-memory data transfer. The DMA module, shown in Figure 7-1, provides two channels that allow byte, word, or long-word operand transfers. These transfers can be either single or dual address and to either on- or off-chip devices. The DMA contains the following features:

- Two Independent, Fully Programmable DMA Channels
- Single-Address Transfers and Dual-Address Transfers
- 32-Bit Address and 32-Bit Data Capability
- Two 32-Bit Transfer Counters
- Four 32-Bit Address Pointers That Can Increment or Remain Constant
- Operand Packing and Unpacking for Dual-Address Transfers
- Supports All Bus-Termination Modes
- Provides Two-Clock-Cycle Internal Module Access
- Provides Two-Clock-Cycle External Access Using MC68349 Chip Selects
- Provides Full DMA Handshake for Burst Transfers and Cycle Steal
- Programmable Interrupt Level Allows Central Processing Unit (CPU) to Preempt DMA Activity

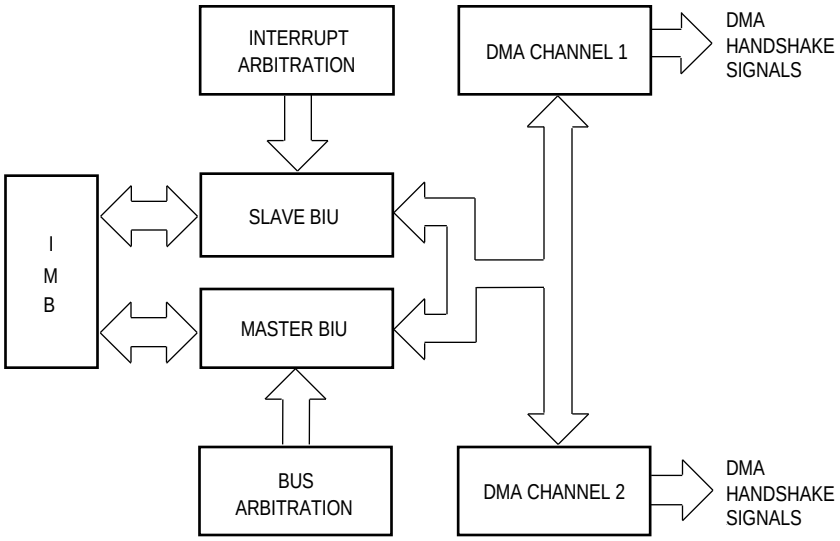


Figure 7-1. DMA Block Diagram

7.1 DMA MODULE OVERVIEW

The main purpose of the DMA controller module is to transfer data at very high rates, usually much faster than the CPU32+ under software control can handle. The term DMA is used to refer to the ability of a peripheral device to access memory in a system in the same manner as a microprocessor does. DMA operations can greatly increase overall system performance.

The MC68349 DMA module consists of two independent, programmable channels. The term DMA is used throughout this section to reference either channel 1 or channel 2 since the two are functionally equivalent. Each channel has independent request, acknowledge, and done signals. However, both channels cannot own the bus at the same time. Therefore, it is impossible to implicitly address both DMA channels at the same time. The MC68349 on-chip peripherals do not support the single-address transfer mode.

DMA requests may be internally generated by the channel or externally generated by a device. For an internal request, the amount of bus bandwidth allocated for the DMA can be programmed. The DMA channels support two external request modes: burst mode and cycle steal mode.

Each DMA channel has a configurable interrupt service mask (ISM) level which causes the channel to temporarily suspend DMA activity when the CPU interrupt service level exceeds the ISM value. This feature can be used to minimize the effects of DMA activity or time-critical interrupt sources.

The DMA controller supports single- and dual-address transfers. In single-address mode, a channel supports 32 bits of address and 32 bits of data. Only an external request can be used to start a transfer in the single-address mode. The DMA provides address and control signals during a single-address transfer. The requesting device either sends or receives data to or from the specified address (see Figure 7-2). In dual-address mode, a channel supports 32 bits of address and 32 bits of data. The dual-address transfers can be started by either the internal request mode or by an external device using the request signal. In this mode, two bus transfers occur, one from a source device and the other to a destination device (see Figure 7-3). In dual-address mode, operands are packed or unpacked according to port sizes and addresses.

Any operation involving the DMA will follow the same basic steps: channel initialization, data transfer, and channel termination. In the channel initialization step, the DMA channel registers are loaded with control information, address pointers, and a byte transfer count. The channel is then started. During the data transfer step, the DMA accepts requests for operand transfers and provides addressing and bus control for the transfers. The channel termination step occurs after operation is complete. The channel indicates the status of the operation in the channel status register.

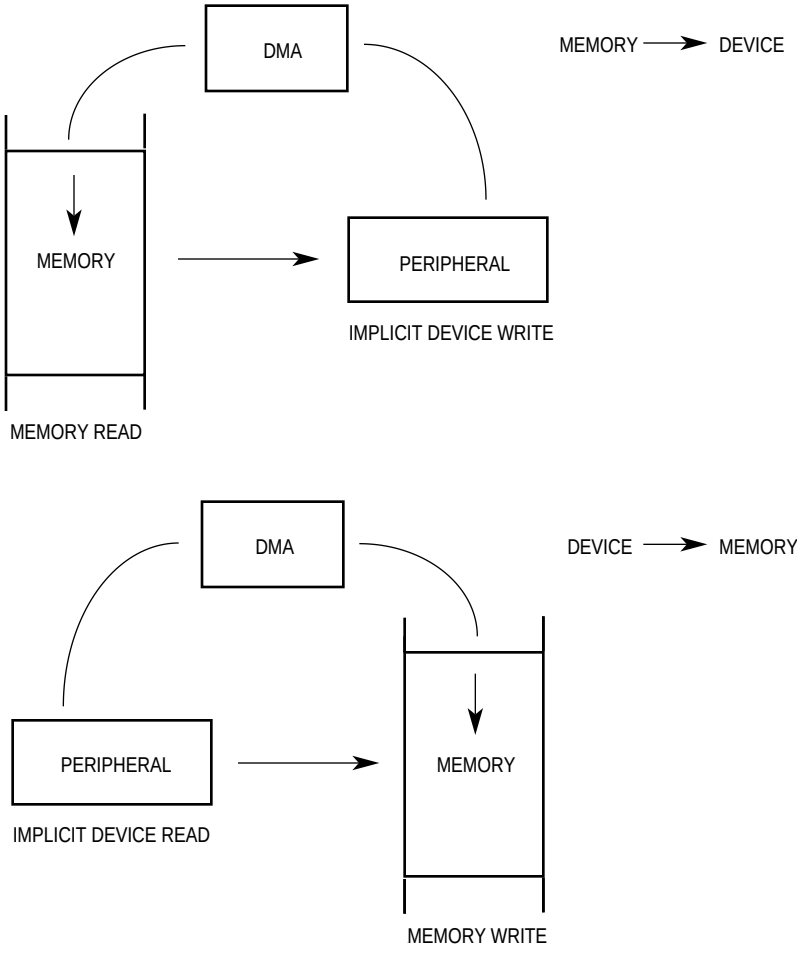


Figure 7-2. Single-Address Transfers

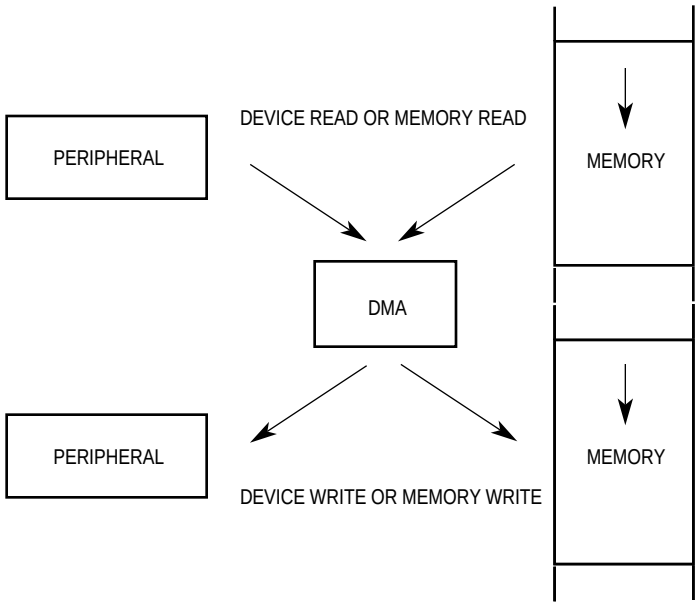


Figure 7-3. Dual-Address Transfer

7.2 DMA MODULE SIGNAL DEFINITIONS

This section contains a brief description of the DMA module signals used to provide handshake control for either a source or destination external device.

NOTE

The terms *assertion* and *negation* are used throughout this section to avoid confusion when dealing with a mixture of active-low and active-high signals. The term *assert* or *assertion* indicates that a signal is active or true, independent of the level represented by a high or low voltage. The term *negate* or *negation* indicates that a signal is inactive or false.

7.2.1 DMA Request ($\overline{\text{DREQ1}}$, $\overline{\text{DREQ2}}$)

This active-low input is asserted by a peripheral device to request an operand transfer between that peripheral and memory. The assertion of $\overline{\text{DREQx}}$ starts the DMA process. The assertion level in external burst mode is level sensitive; in external cycle steal mode, it is falling-edge sensitive.

7.2.2 DMA Acknowledge ($\overline{\text{DACK1}}$, $\overline{\text{DACK2}}$)

This active-low output is asserted by the DMA to signal to a peripheral that an operand is being transferred in response to a previous transfer request.

7.2.3 DMA Done ($\overline{\text{DONE1}}$, $\overline{\text{DONE2}}$)

This active-low bidirectional signal is asserted by the DMA or a peripheral device during any DMA bus cycle to indicate that the last data transfer is being performed. $\overline{\text{DONEx}}$ is an active input in any mode. As an output, $\overline{\text{DONEx}}$ is only active in external request mode. An external pullup resistor is required even if operating only in the internal request mode.

7.3 TRANSFER REQUEST GENERATION

The DMA channel supports two types of request generation methods: internal and external. Internally generated requests can be programmed to limit the amount of bus utilization. Externally generated requests can be either burst mode or cycle steal mode. The request generation method used for the channel is programmed by the REQ field in the channel control register (CCR).

7.3.1 Internal Request Generation

The channel is started as soon as the STR bit in the CCR is set. The channel immediately requests the bus and begins transferring data. Only internal requests can limit the amount of bus utilization. The percentage of the bus bandwidth that the DMA channel uses during a transfer is selected by the CCR bus bandwidth (BB) field.

7.3.1.1 INTERNAL REQUEST, MAXIMUM RATE. Internal generation using 100% of the internal bus always has a transfer request pending for the channel until the block transfer is complete. As soon as the channel is started, the DMA will arbitrate for the internal bus and begin to transfer data when it becomes bus master. If no exceptions occur, all operands in the data block will be transferred in one burst so that the DMA will use 100% of the available bus bandwidth.

7.3.1.2 INTERNAL REQUEST, LIMITED RATE. To guarantee that the DMA does not use all of the available bus bandwidth during a transfer, the bus bandwidth allocated to the DMA can be limited. There are three programmed constants in the CCR used to monitor the bus activity and allow the DMA to use a percentage of the bus bandwidth. Options are 25%, 50%, and 75% of 1024 clock periods. See Table 7-5 for more information.

7.3.2 External Request Generation

To control the transfer of operands to or from memory in an orderly manner, a peripheral device uses the $\overline{\text{DREQx}}$ input signal to request service. If the channel is programmed for external request and the CCR STR bit is set, an external request ($\overline{\text{DREQx}}$) signal must be asserted before the channel requests the bus and begins a transfer. The DMA supports external burst mode and external cycle steal mode.

The generation of the request from the source or destination is specified by the ECO bit of the CCR. The external requests can be for either single- or dual-address transfers.

7.3.2.1 EXTERNAL BURST MODE. For external devices that require very high data transfer rates, the burst request mode allows the DMA channel to use all of the bus bandwidth under control of the external device. In burst mode, the $\overline{\text{DREQx}}$ input to the DMA is level sensitive and is sampled at certain points to determine when a valid request is asserted by the device. The device requests service by asserting $\overline{\text{DREQx}}$ and leaving it asserted. In response, the DMA arbitrates for the bus and performs an operand transfer. During each operand transfer, the DMA asserts DMA acknowledge ($\overline{\text{DACKx}}$) to indicate to the device that a request is being serviced. $\overline{\text{DACKx}}$ is asserted on the cycle of either the source or destination device, depending on which one generated the request as programmed by the CCR ECO bit.

To allow more than one transfer to be recognized, $\overline{\text{DREQx}}$ must meet the asynchronous setup and hold times while $\overline{\text{DACKx}}$ is asserted in the DMA bus cycle. Upon completion of a request, $\overline{\text{DREQx}}$ should be held asserted (bursting) into the following DMA bus cycle to allow another transfer to occur. The recognized request will immediately be serviced. If $\overline{\text{DREQx}}$ is negated before $\overline{\text{DACKx}}$ is asserted, a new request is not recognized, and the DMA channel releases ownership of the bus.

7.3.2.2 EXTERNAL CYCLE STEAL MODE. For external devices that generate a pulsed signal for each operand to be transferred, the cycle steal request mode uses the $\overline{\text{DREQx}}$ signal as a falling-edge-sensitive input. The $\overline{\text{DREQx}}$ pulse generated by the device must be asserted during two consecutive falling edges of the clock to be recognized as valid. Therefore, if a peripheral generates it asynchronously, it must be at least two clock periods long.

The DMA channel responds to cycle steal requests the same as all other requests. However, if subsequent $\overline{\text{DREQx}}$ pulses are generated before $\overline{\text{DACKx}}$ is asserted in response to each request, they are ignored. If $\overline{\text{DREQx}}$ is asserted after the DMA channel asserts $\overline{\text{DACKx}}$ for the previous request but before $\overline{\text{DACKx}}$ is negated, then the new request is serviced before bus ownership is released. If a new request is not generated by the time $\overline{\text{DACKx}}$ is negated, the bus is released.

7.3.2.3 EXTERNAL REQUEST WITH OTHER MODULES. The DMA controller can be externally connected to the serial module and used in conjunction with the serial module to send or receive data. The DMA takes the place of a separate service routine for accessing or storing data that is sent or received by the serial module. Using the DMA also lowers the CPU32+ overhead required to handle the data transferred by the serial module. Figure 7-4 shows the external connections required for using the DMA with the serial module.

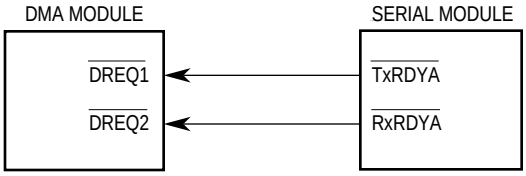


Figure 7-4. DMA External Connections to Serial Module

For serial receive, the DMA reads data from the serial receive buffer register (when the serial module has filled the buffer on input) and writes data to memory. For serial transmit, the DMA reads data from memory and writes data to the serial transmit buffer register. Only dual-address mode can be used with the serial module. The MC68349 on-chip peripherals do not support single-address transfers.

7.4 DATA TRANSFER MODES

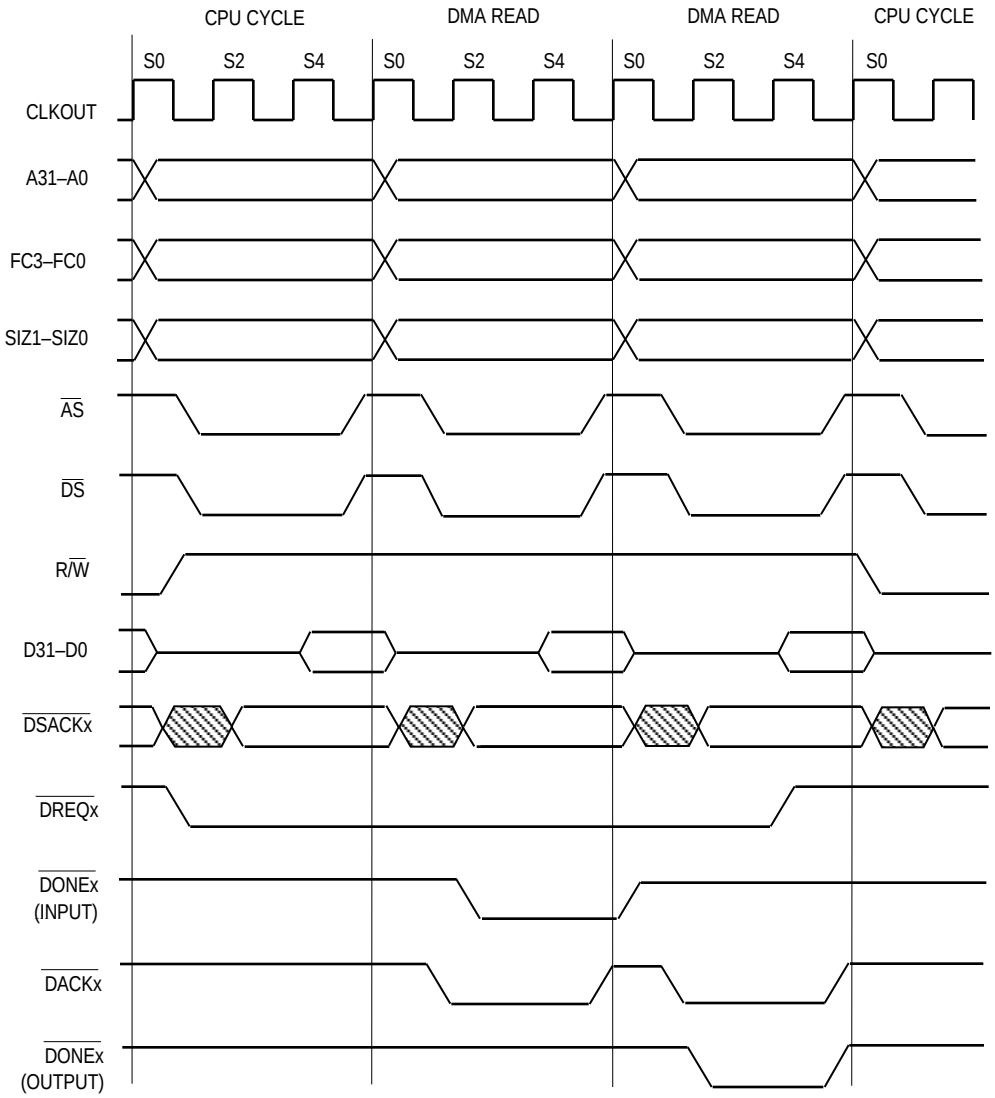
The DMA channel supports single- and dual-address transfers. The single-address transfer mode consists of one DMA bus cycle, which allows either a read or a write cycle to occur. The dual-address transfer mode consists of a source operand read and a destination operand write. Two DMA bus cycles are executed for the dual-address mode: a DMA read cycle and a DMA write cycle.

7.4.1 Single-Address Mode

The single-address DMA bus cycle allows data to be transferred directly between a device and memory without going through the DMA. In this mode, the operand transfer takes place in one bus cycle, where only the memory is explicitly addressed. The DMA bus cycle may be either a read or a write cycle. The DMA provides the address and control signals required for the operation. The requesting device either sends or receives data to or from the specified address. Only external requests can be used to start a transfer when the single-address mode is selected. An external device uses $\overline{\text{DREQx}}$ to request a transfer.

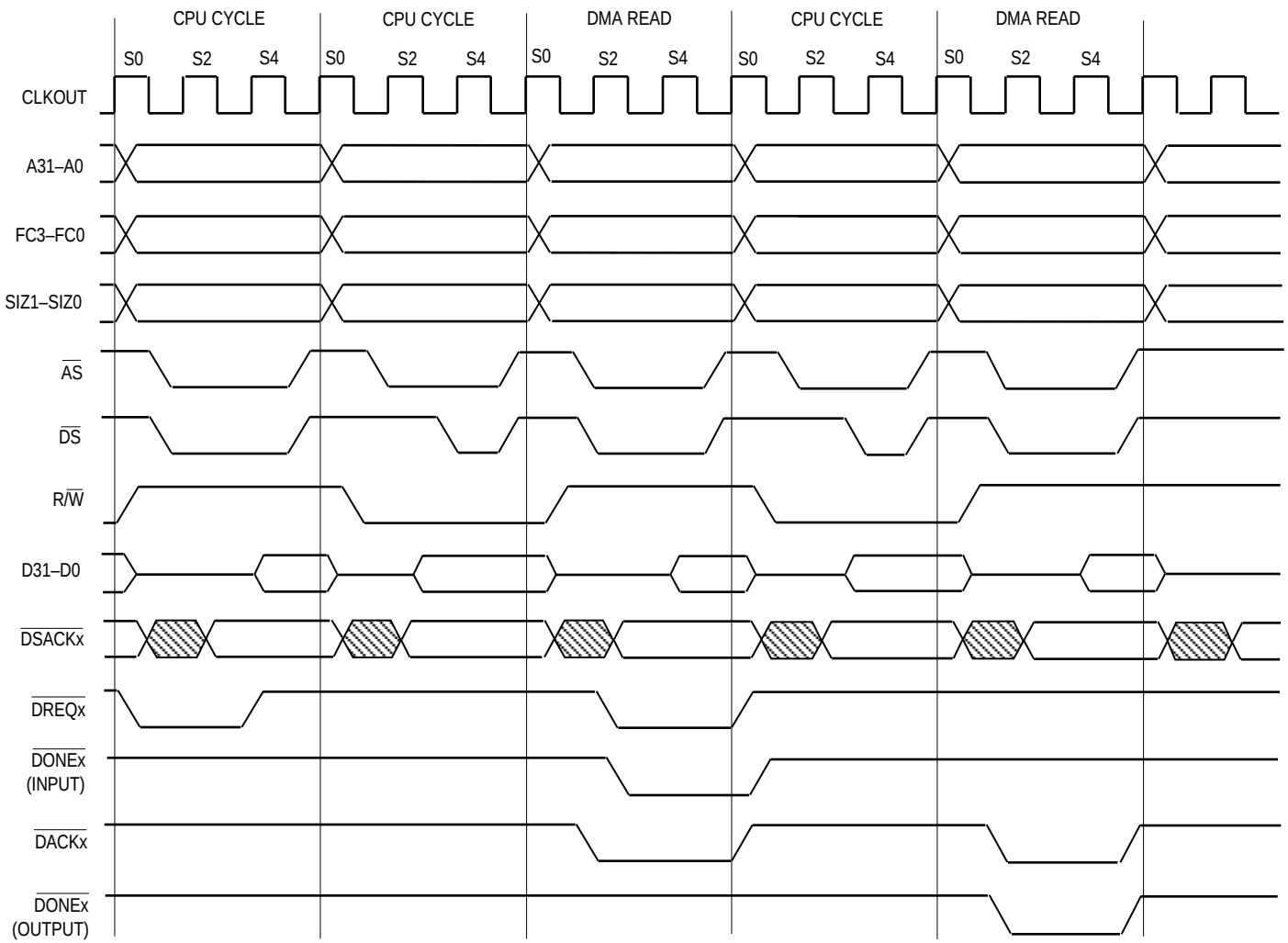
Each DMA channel can be independently programmed to provide single-address transfers. The CCR ECO bit controls whether a source read or a destination write cycle occurs on the data bus. If the ECO bit is set, the external handshake signals are used with the source operand and a single-address source read occurs. If the ECO bit is cleared, the external handshake signals are used with the destination operand, and a single-address destination write occurs. The channel can be programmed to operate in either burst transfer mode or cycle steal mode. See **7.7 Register Description** for more information.

7.4.1.1 SINGLE-ADDRESS READ. During the single-address source (read) cycle, the DMA controls the transfer of data from memory to a device. The memory selected by the address specified in the source address register (SAR), the source function codes in the function code register (FCR), and the source size in the CCR provides the data and control signals on the data bus. This bus cycle operates like a normal read bus cycle. The DMA control signals ($\overline{\text{DACKx}}$ and $\overline{\text{DONEx}}$) are asserted in the source (read) cycle. See Figures 7-5 and 7-6 for timing diagrams single-address read for external burst and cycle steal modes.



NOTES:
1. Timing to generate more than one DMA request.
2. DACKx and DONEx (DMA control signals) are asserted in the source (read) DMA cycle.
3. DREQx must be asserted while DACKx is asserted and meet the setup and hold times for more than one DMA transfer to be recognized.

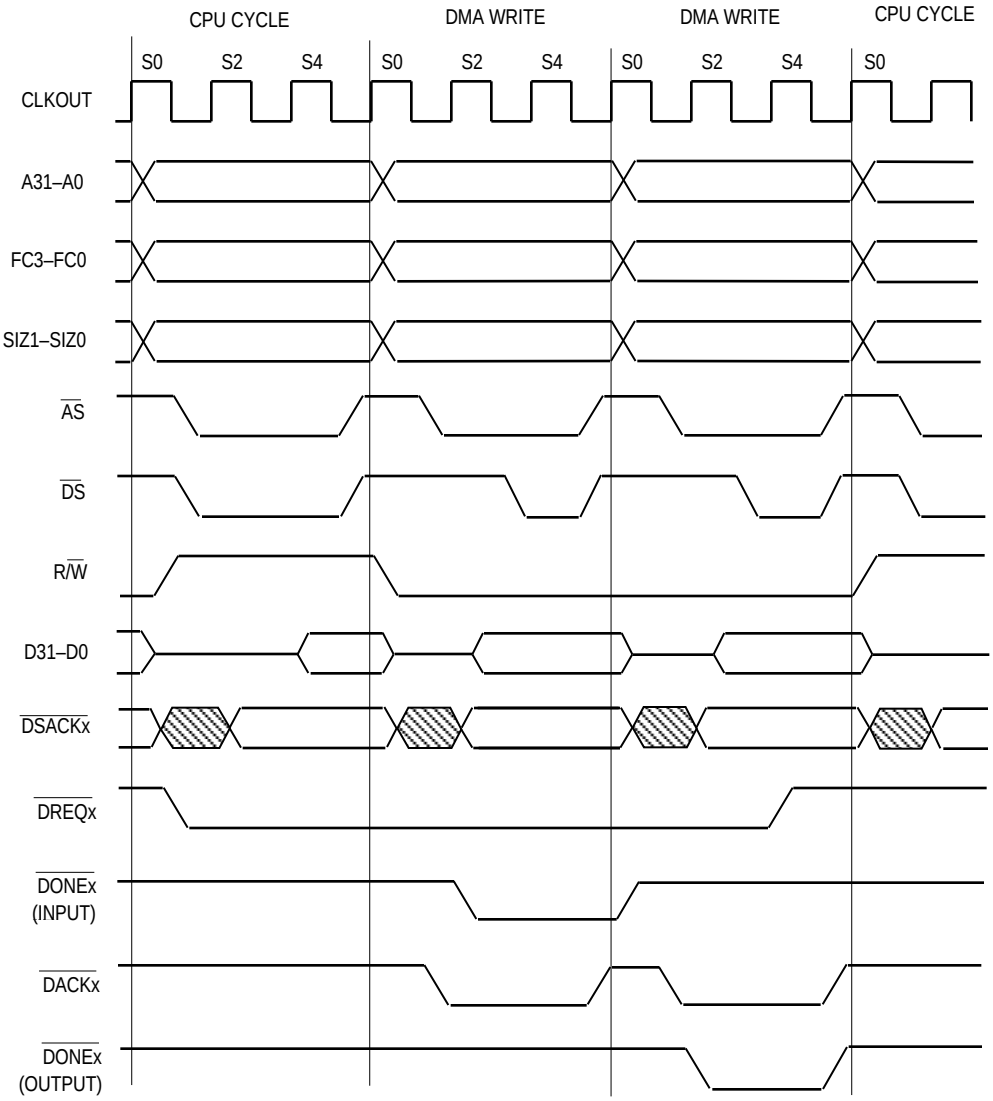
Figure 7-5. Single-Address Read Timing (External Burst)



NOTES:
1. DREQx must be active for two consecutive clocks for a DMA request to be recognized.
2. To cause another DMA transfer, DREQx is asserted after DACKx is asserted and before DACKx is negated.
3. DACKx and DONEx (DMA control signals) are asserted in the source (read) DMA cycle.

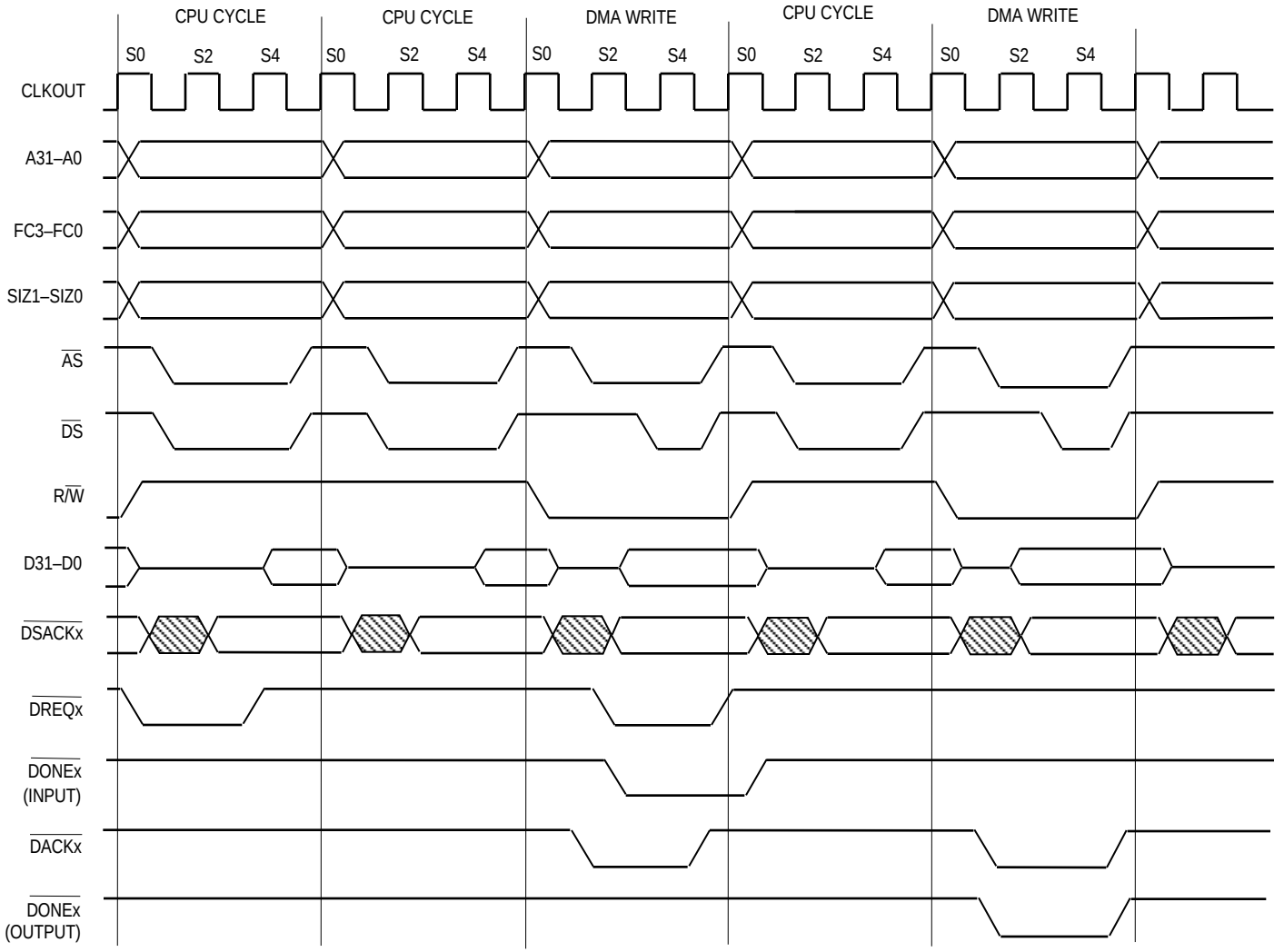
Figure 7-6. Single-Address Read Timing (Cycle Steal)

7.4.1.2 SINGLE-ADDRESS WRITE. During the single-address destination (write) cycle, the DMA controls the transfer of data from a device to memory. The data is written to memory selected by the address specified in the destination address register (DAR), the destination function codes in the FCR, and the size in the CCR. The destination (write) DMA bus cycle has timing identical to a write bus cycle. The DMA control signals ($\overline{\text{DACKx}}$ and $\overline{\text{DONEx}}$) are asserted in the destination (write) cycle. See Figures 7-7 and 7-8 for timing diagrams of single-address write for external burst and cycle steal modes.



NOTES:
1. Timing to generate more than one DMA request.
2. DACKx and DONEx (DMA control signals) are asserted in the source (read) DMA cycle.
3. DREQx must be asserted while DACKx is asserted, and meet the setup and hold times for more than one DMA transfer to be recognized.

Figure 7-7. Single-Address Write Timing (External Burst)



NOTES:
1. DREQx must be active for two consecutive clocks for a DMA request to be recognized.
2. To cause another DMA transfer, DREQx is asserted after DACKx is asserted and before DACKx is negated.
3. DACKx and DONEx (DMA control signals) are asserted in the destination (write) DMA cycle.

Figure 7-8. Single-Address Write Timing (Cycle Steal)

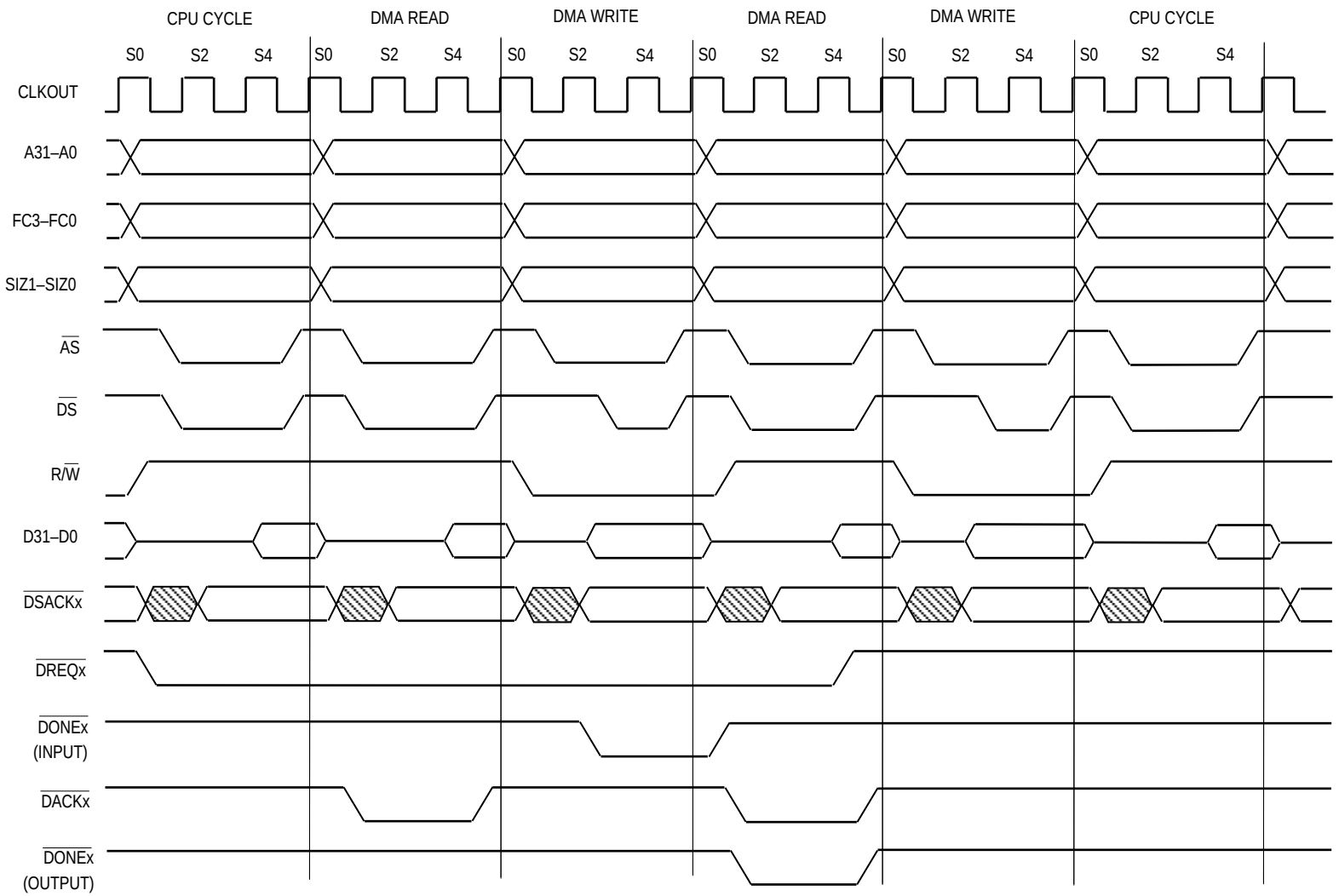
7.4.2 Dual-Address Mode

The dual-address DMA bus cycle transfers data between a device or memory and the DMA internal data holding register (DHR). In this mode, any operand transfer takes place in two DMA bus cycles, one where a device is addressed and one where memory is addressed. The data transferred during a dual-address operation is either read from the data bus into the DHR or written from the DHR to the data bus.

Each DMA channel can be programmed to operate in the dual-address transfer mode. In this mode, the operand is read from the source address specified in the SAR and placed in the DHR. The operand read may take up to four bus cycles to complete because of differences in operand sizes of the source and destination. The operand is then written to the address specified in the DAR. This transfer may also be up to four bus cycles long. In this manner, various combinations of peripheral, memory, and operand sizes may be used. See **7.7 Register Description** for more information.

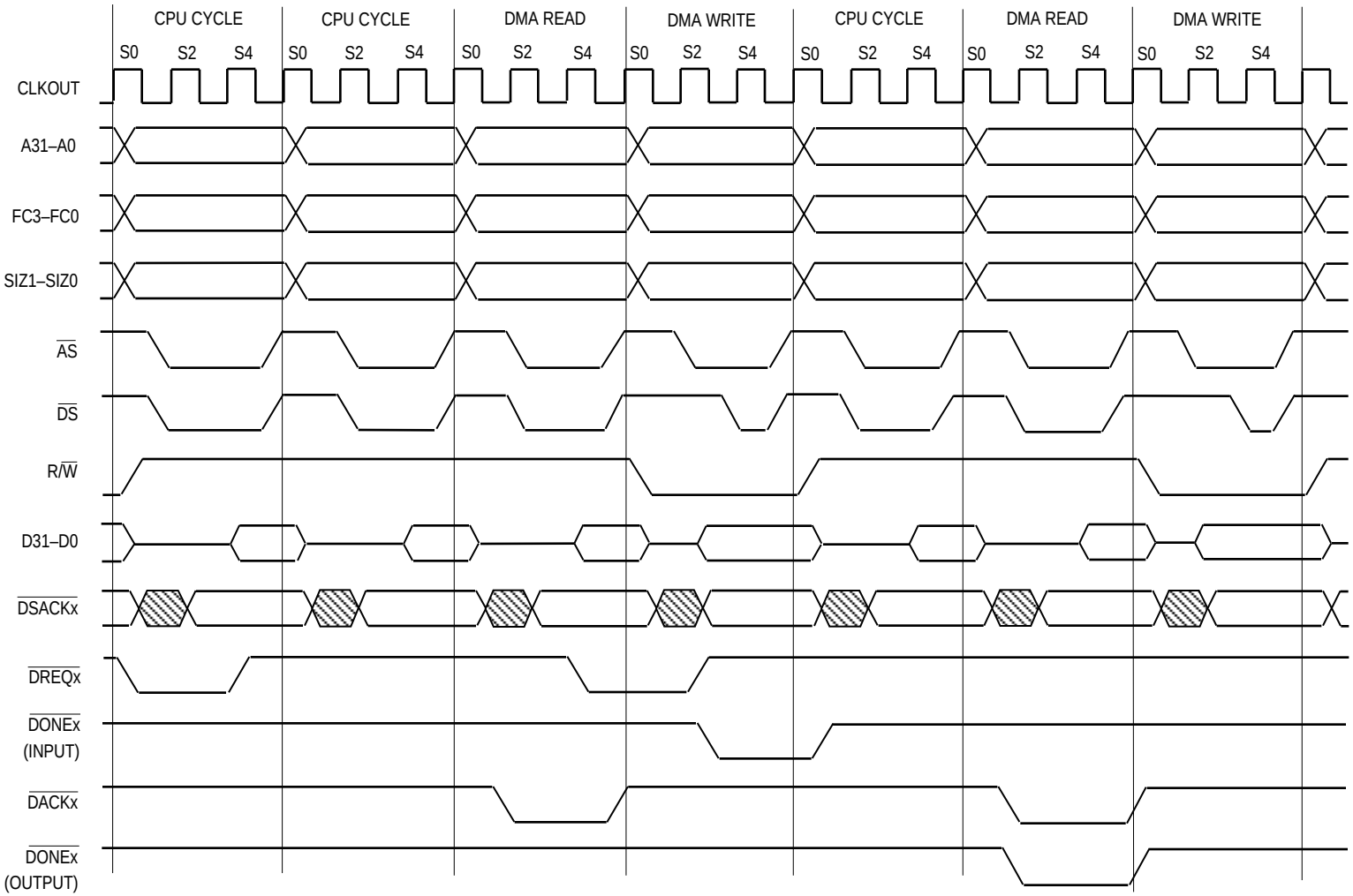
The dual-address transfers can be started by either the internal request mode or by an external device using the $\overline{\text{DREQx}}$ input signal. When the external device uses $\overline{\text{DREQx}}$, the channel can be programmed to operate in either burst transfer mode or cycle steal mode.

7.4.2.1 DUAL-ADDRESS READ. During the dual-address read cycle, the DMA reads data from a device or memory into the internal DHR. The device or memory is selected by the address specified in the SAR, the source function codes in the FCR, and the source size in the CCR. Data is read from the memory or peripheral and placed in the DHR when the bus cycle is terminated. When the complete operand has been read, the SAR is incremented by 0, 1, 2, or 4, according to the size and increment information specified by the SSIZE and SAPI bits of the CCR. The DMA control signals ($\overline{\text{DACKx}}$ and $\overline{\text{DONEx}}$) are asserted in the source (read) cycle when the source device makes a request. See Figures 7-9 and 7-10 for timing diagrams of dual-address reads for external burst and cycle steal modes.



NOTES:
1. Timing to generate more than one DMA transfer.
2. DACKx and DONEx (DMA control signals) are asserted in the source (read) DMA cycle.
3. DREQx must be asserted while DACKx is asserted and meet the setup and hold times for more than one DMA transfer to be recognized.
4. DONEx (input) can be asserted in either the read or write DMA bus cycle to indicate that the next DMA transfer will be the last one.

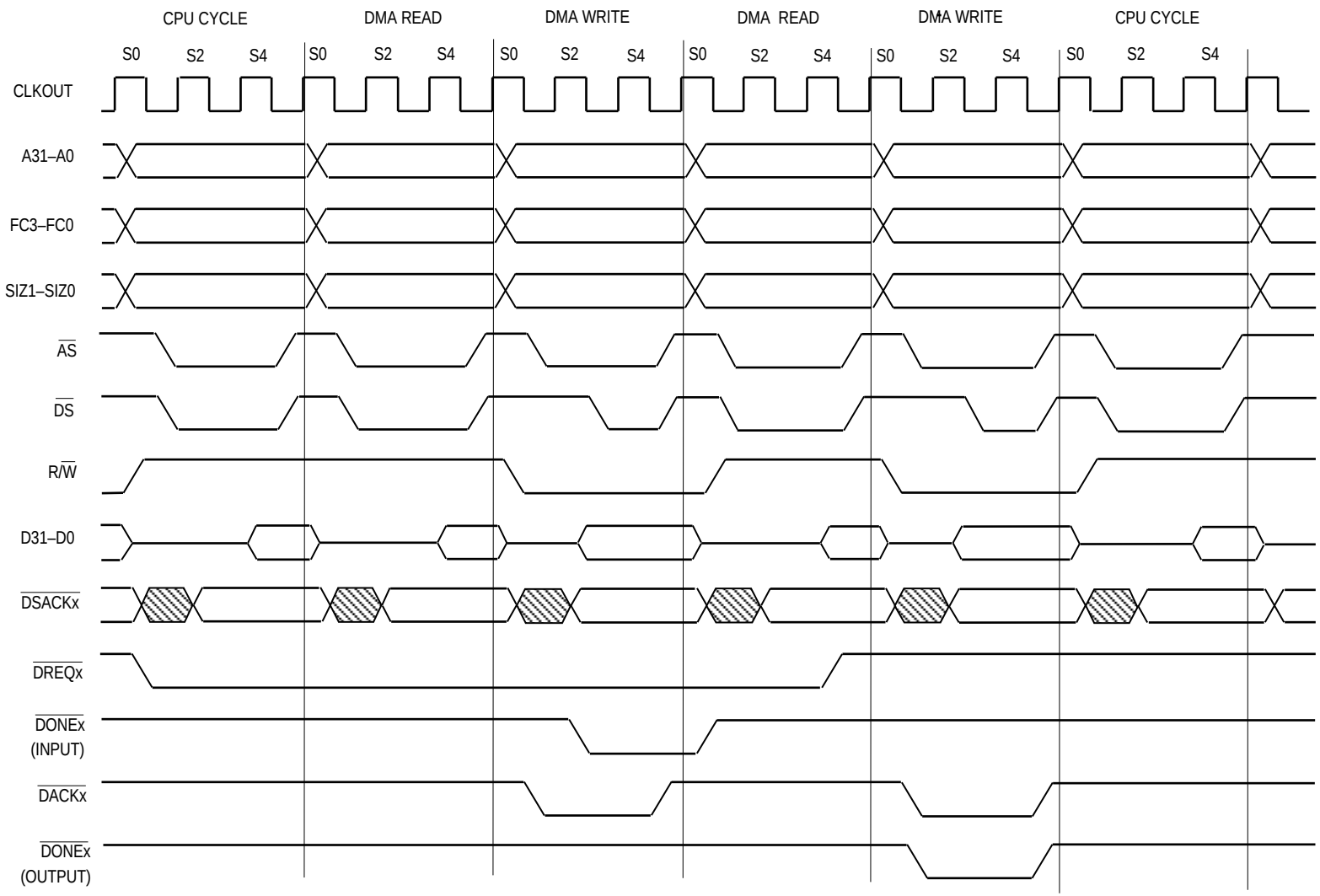
Figure 7-9. Dual Address Read Timing (External Burst–Source Requesting)



NOTES:
1. DREQx must be active for two consecutive clocks for a DMA request to be recognized.
2. To cause another DMA transfer, the DREQx is asserted after DACKx is asserted and before DACKx is negated.
3. DACKx and DONEx (DMA control signals) are asserted in the source (read) DMA cycle.
4. DONEx (input) can be asserted in either the read or write DMA bus cycle to indicate that the next DMA transfer will be the last one.

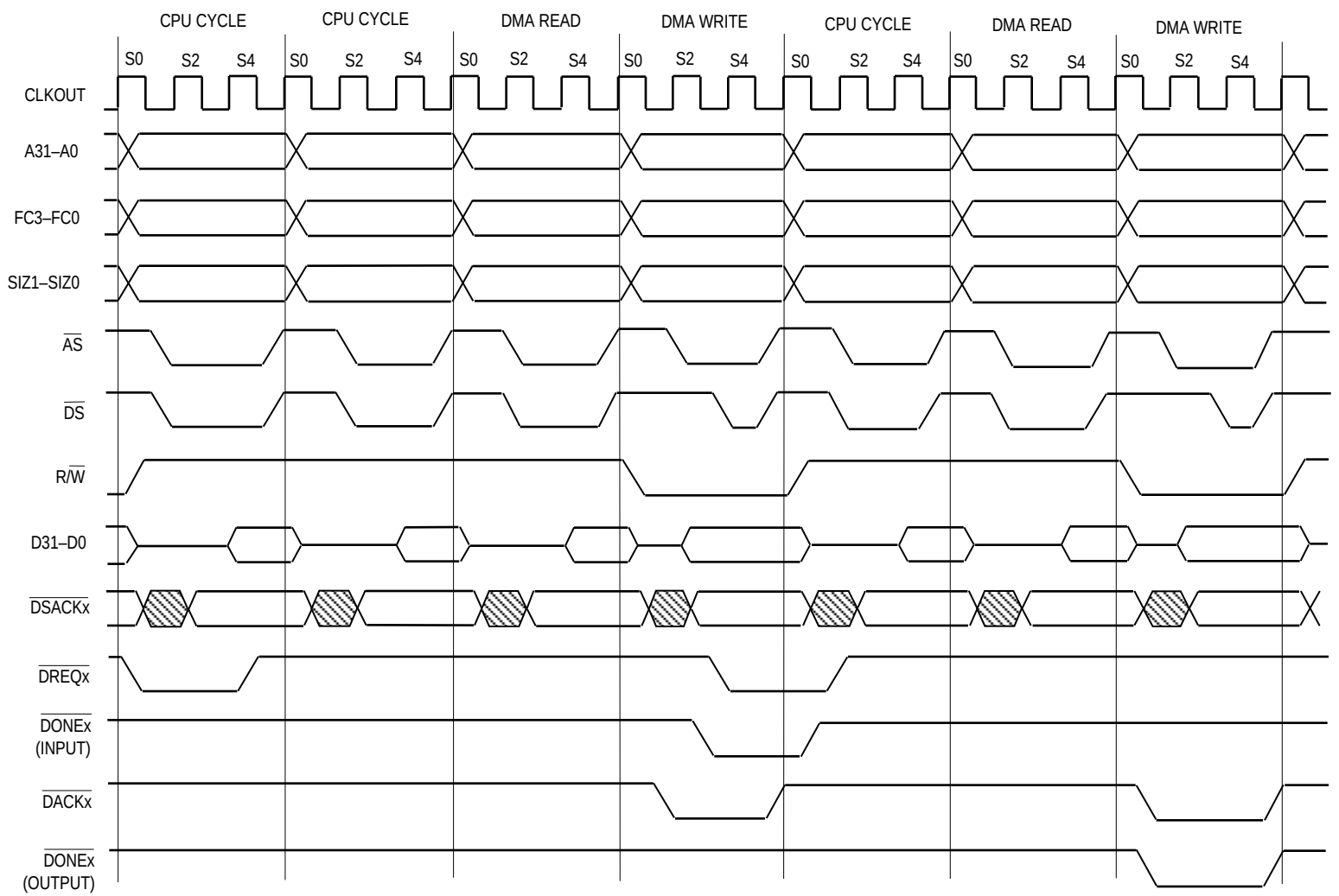
Figure 7-10. Dual-Address Read Timing (Cycle Steal–Source Requesting)

7.4.2.2 DUAL-ADDRESS WRITE. During the dual-address write cycle, the DMA writes data to a device or memory from the internal DHR. The data in the DHR is written to the device or memory selected by the address in the DAR, the destination function codes in the FCR, and the size in the CCR. When the complete operand is written, the DAR is incremented by 0, 1, 2, or 4, according to the increment and size information specified by the DAPI and DSIZE bits of the CCR, and the byte transfer count register (BTC) is decremented by the number of bytes transferred. If the BTC is equal to zero and there were no errors, the channel status register (CSR) DONE bit is set, and the $\overline{\text{DONEx}}$ signal for the DMA handshake is asserted. The DMA control signals ($\overline{\text{DACKx}}$ and $\overline{\text{DONEx}}$) are asserted in the destination (write) cycle when the destination device makes a request. See Figures 7-11 and 7-12 for timing diagrams of dual-address writes for external burst and cycle steal modes.



NOTES:
1. Timing to generate more than one DMA transfer.
2. DACKx and DONEx (DMA control signals) are asserted in the destination (write) DMA cycle.
3. DREQx must be asserted while DACKx is asserted and meet the setup and hold times for more than one DMA transfer to be recognized.
4. DONEx (input) can be asserted in either the read or write DMA bus cycle to indicate that the next DMA transfer will be the last one.

Figure 7-11. Dual Address Write Timing (External Burst–Destination Requesting)



NOTES:
1. DREQx must be active for two consecutive clocks for a DMA request to be recognized.
2. To cause another DMA transfer, DREQx is asserted after DACKx is asserted and before DACKx is negated.
3. DACKx and DONEx (DMA control signals) are asserted in the destination (write) DMA cycle.
4. DONEx (Input) can be asserted in either the read or write DMA bus cycle to indicate that the next DMA transfer will be the last one.

Figure 7-12. Dual-Address Write Timing (Cycle Steal–Destination Requesting)

7.5 BUS ARBITRATION

The DMA controller module uses the M68000 bus arbitration protocol to request bus mastership for DMA transfers. Each channel arbitrates for the bus independently. The source (read) DMA bus cycle has timing identical to a read bus cycle. The destination (write) DMA bus cycle has timing identical to a write bus cycle. However, the DMA channel transfers are unique in that the FC3 signal can be asserted during the source operand bus cycle and remain asserted until the end of the destination operand bus cycle.

For internal request generation, as soon as the CCR STR bit is set, the DMA channel arbitrates for the bus and begins to transfer data when it becomes bus master. For external request generation, the STR bit must be set and a $\overline{\text{DREQx}}$ signal must be asserted before the channel arbitrates for the bus and begins a transfer.

7.6 DMA CHANNEL OPERATION

The following paragraphs describe the programmable channel functions available for the DMA channel, the data transfer operations, and behavior during cycle termination. This description applies to both channels.

Any DMA channel operation adheres to the following basic sequence:

1. Channel Initialization and Startup—The channel registers are initialized. The channel is then started by setting the CCR STR bit. The first operand transfer request (either internally or externally generated) is recognized.
2. Data Transfer—After a channel is started, it transfers one operand in response to each request until an entire data block is transferred.
3. Channel Termination—The channel can terminate by normal completion or from an error. The CSR indicates the status of the operation.

7.6.1 Channel Initialization and Startup

Before starting a block transfer operation, the channel registers must be initialized with information describing the channel configuration, request generation method, and data block. This initialization is accomplished by programming the appropriate information into the channel registers.

The SAR is loaded with the source (read) address. If the transfer is from a peripheral device to memory, the source address is the location of the peripheral data register. If the transfer is from memory to a peripheral device or memory to memory, the source address is the starting address of the data block. This address may be any byte address. In the single-address mode with the destination (write) device requesting mode of operation, this register is not used.

The DAR should contain the destination (write) address. If the transfer is from a peripheral device to memory or memory to memory, the DAR is loaded with the starting address of the data block to be written. If the transfer is from memory to a peripheral device, the DAR is

loaded with the address of the peripheral data register. This address may be any byte address. In the single-address mode with the source (read) device requesting mode of operation, this register is not used.

The manner in which the SAR and DAR change after each cycle depends upon the values in the CCR SSIZE and DSIZE fields and SAPI and DAPI bits, and the starting address in the SAR and DAR. If programmed to increment, the increment value is 1, 2, or 4 for byte, word, or long-word operands, respectively. If the address register is programmed to remain unchanged (no count), the register is not incremented after the operand transfer. The SAR and DAR are incremented if a bus error terminates the transfer. Therefore, either the SAR or the DAR contain the next address after the one that caused the bus error.

The BTC must be loaded with the number of byte transfers that are to occur. This register is decremented by 1, 2, or 4 at the end of each transfer. The FCR must be loaded with the source and destination function codes. Although these function codes may not be used in the address decode for the memory or peripheral, they are provided if needed. The CSR must be cleared for channel startup.

Once the channel has been initialized, it is started by writing a one to the STR bit in the CCR. Programming the channel for internal request causes the channel to request the bus and start transferring data immediately. If the channel is programmed for external request, $\overline{\text{DREQx}}$ must be asserted before the channel requests the bus. The $\overline{\text{DREQx}}$ input is ignored until the channel is started, since the channel does not recognize transfer requests until it is active.

If any fields in the CCR are modified while the channel is active, that change is effective immediately. To avoid any problems with changing the setup for the DMA channel, a zero should be written to the STR bit in the CCR to halt the DMA channel at the end of the current bus cycle.

7.6.2 Data Transfers

Each operand transfer requires from one to five bus cycles to complete. Once a bus request is recognized and the operand transfer begins, both the source (read) cycle and/or the destination (write) cycle occur before a new bus request may be honored, even if the new bus request is of higher priority.

7.6.2.1 INTERNAL REQUEST TRANSFERS. The percentage of bus bandwidth utilization can be limited for internal request transfers.

7.6.2.2 EXTERNAL REQUEST TRANSFERS. In single-address mode, only one bus cycle is run for each request. Since the operand size must be equal to the device port size in single-address mode, the number of normally terminated bus cycles executed during a transfer operation is always equal to the value programmed into the corresponding size field of the CCR. The sequencing of the address bus follows the programming of the CCR and address register (SAR or DAR) for the channel.

Each operand transfer in dual-address mode requires from two to five bus cycles in response to each operand transfer request. If the source and destination operands are the

same size, two cycles will transfer the complete operand. If the source and destination operands are different sizes, the number of cycles will vary. If the source is a long-word and the destination is a byte, there would be one bus cycle for the read and four bus cycles for the write. Once the DMA channel has started a dual-address operand transfer, it must complete that transfer before releasing ownership of the bus or servicing a request for another channel of equal or higher priority, unless one of the bus cycles is terminated with a bus error during the transfer.

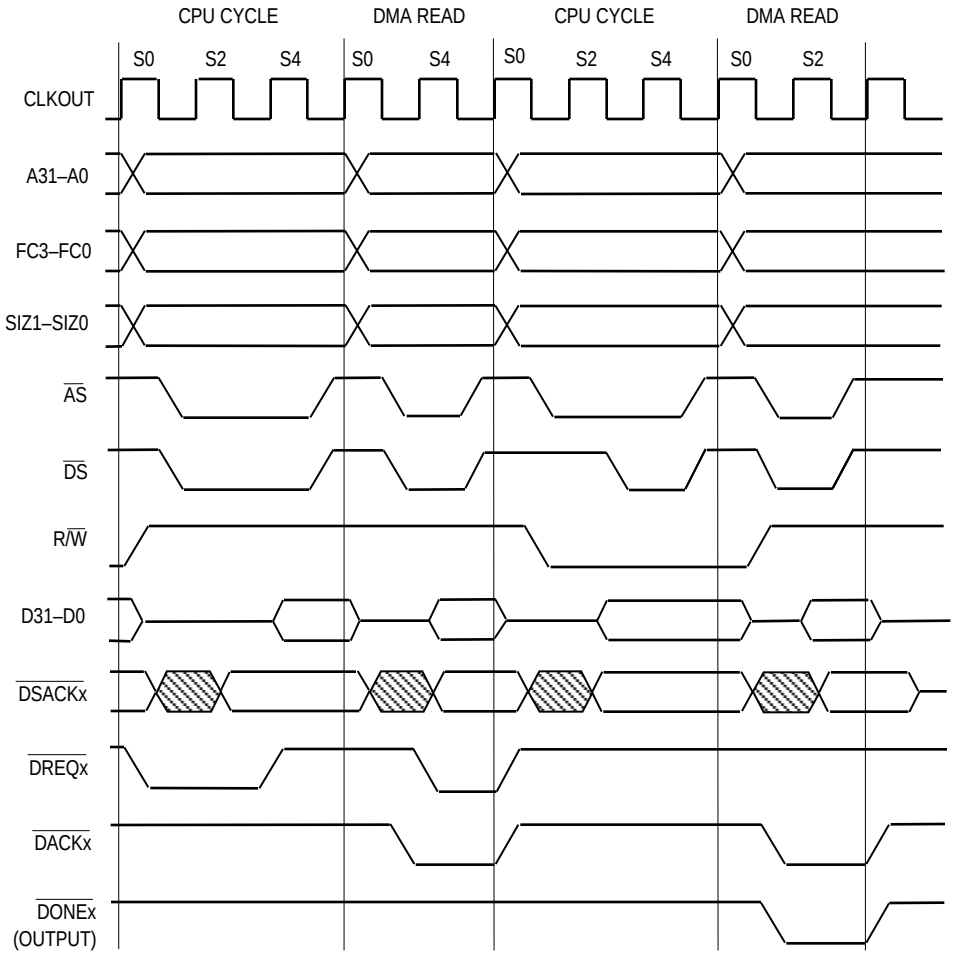
7.6.3 Channel Termination

The channel can terminate by normal completion or as the result of an error. The status of a DMA operation can be determined by reading the CSR. The DMA channel can also interrupt the processor to inform it of errors, normal transfer completion, or breakpoints. The fast termination option can also be used to provide a two-clock access for external requests.

7.6.3.1 CHANNEL TERMINATION. The channel operation can be terminated for several reasons: the BTC is decremented to zero, a peripheral device asserts $\overline{\text{DONE}}$x during an operand transfer, the STR bit is cleared in the CCR, a bus cycle is terminated with a bus error, or a reset occurs.

7.6.3.2 INTERRUPT OPERATION. Interrupts can be generated by error termination of a bus cycle or by normal channel completion. Specifically, if the CCR interrupt error (INTE) bit is set and a bus error on source (CCR BES) bit, bus error on destination (CCR BED) bit, or configuration error (CCR CONF) bit is set, the CCR IRQ bit is set. In this case, clearing the INTE, BES, BED, or CONF bits causes the IRQ bit to be cleared. If the interrupt normal (CCR INTN) bit is set and the CCR DONE bit is set, the IRQ bit is set. In this case, clearing the INTN or the DONE bit causes the IRQ bit to be cleared. If the interrupt breakpoint (CCR INTB) and the CSR BRKP bits are set, the IRQ bit is set. Clearing INTB or BRKP clears IRQ.

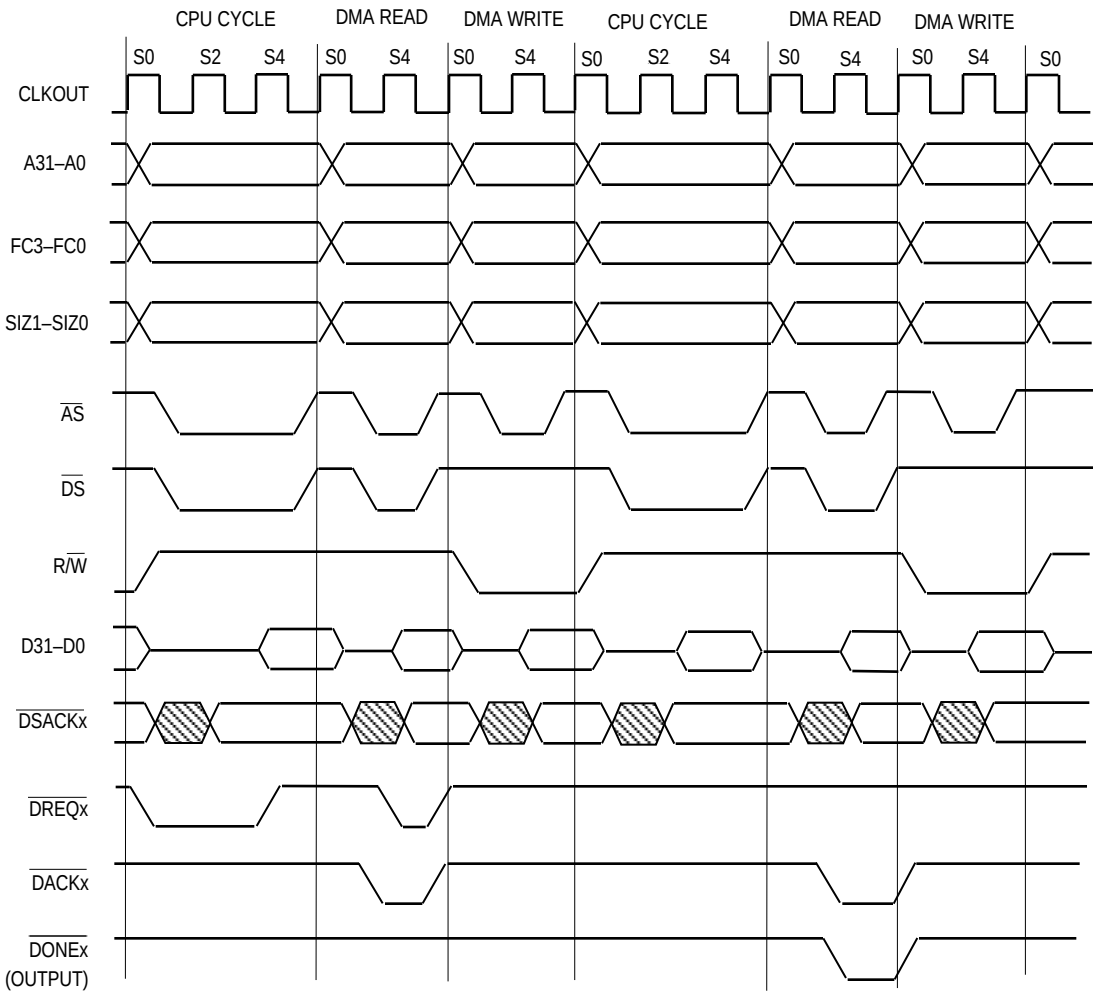
7.6.3.3 FAST TERMINATION OPTION. Using the system integration module (SIM49) chip select logic, the fast termination option (Figure 7-13) can be employed to give a fast bus access of two clock cycles rather than the standard three-cycle access time for external requests. The fast termination option is described in **Section 3 Bus Operation** and **Section 4 System Integration Module**.



NOTES:
1. To cause another DMA transfer, DREQx is asserted after DACKx is asserted and before DACKx is negated.
2. DACKx and DONEx (DMA control signals) are asserted in the source (read) DMA cycle.

Figure 7-13. Fast Termination Option Timing (Cycle Steal)

If the fast termination option is used with external burst request mode (Figure 7-14), an extra DMA cycle may result on every burst transfer. Normally, DREQx is negated when DACKx is returned. In the burst mode with fast termination selected, a new cycle starts even if DREQx is negated simultaneously with DACKx assertion.



NOTES:
1. To cause another DMA transfer, the DREQx is asserted after DACKx is asserted and before DACKx is negated.
2. DACKx and DONEx (DMA control signals) are asserted in the source (read) DMA cycle.

Figure 7-14. Fast Termination Option Timing (External Burst–Source Requesting)

7.7 REGISTER DESCRIPTION

The following paragraphs contain a detailed description of each register and its specific function. Figure 7-15 is a programmer's model (register map) of all registers in the DMA module. Each channel has an independent set of registers. For more information about a particular register, refer to the individual register description. The ADDRESS column indicates the offset of the register from the base address of the DMA channel. The FC column designation of S indicates that register access is restricted to supervisor only. A designation of S/U indicates that access is governed by the SUPV bit in the module configuration register (MCR).

Unimplemented memory locations return logic zero when accessed. All registers support byte, word, and long-word transfers. The register interface from the IMB is 16 bits, which forces long-word accesses to complete as two successive word accesses.

ADDRESS		FC		
CH1	CH2		15 — 8	7 — 0
780	7A0	S	MODULE CONFIGURATION REGISTER	
782	7A2	S	RESERVED	
784	7A4	S	INTERRUPT REGISTER	
786	7A6	S/U	RESERVED	
788	7A8	S/U	CHANNEL CONTROL REGISTER	
78A	7AA	S/U	CHANNEL STATUS REGISTER	FUNCTION CODE REGISTER
78C	7AC	S/U	SOURCE ADDRESS REGISTER MSBs	
78E	7AE	S/U	SOURCE ADDRESS REGISTER LSBs	
790	7B0	S/U	DESTINATION ADDRESS REGISTER MSBs	
792	7B2	S/U	DESTINATION ADDRESS REGISTER LSBs	
794	7B4	S/U	BYTE TRANSFER COUNTER MSBs	
796	7B6	S/U	BYTE TRANSFER COUNTER LSBs	
798	7B8	S/U	RESERVED	
79A	7BA	S/U	RESERVED	
79C	7BC	S/U	RESERVED	
79E	7BE	S/U	RESERVED	

Figure 7-15. DMA Module Programming Model

The registers are discussed in the following paragraphs in alphabetical order. The numbers in the upper right-hand corner of the register diagrams indicate the offset of the register from the base address specified by the module base address register (MBAR) in the SIM49. The first number is the offset for channel 1; the second number is the offset for channel 2. The numbers above the register represent the bit position in the register. The register contains the mnemonic for the bit. The value of these bits after a hardware reset is shown below the register. The access privilege is shown in the lower right-hand corner.

NOTE

A CPU32+ RESET instruction will not affect the MCR but will reset all other registers in the DMA module as though a hardware reset occurred. The term DMA is used to reference either channel 1 or channel 2, since the two are functionally equivalent.

7.7.1 Byte Transfer Counter Register (BTC)

The BTC is a 32-bit register that contains the number of bytes left to transfer in a given block. This register is accessible in either supervisor or user space. The BTC can always be read or written to when the DMA module is enabled (i.e., the STP bit in the MCR is cleared).

BTC1, BTC2 $794, $7B4

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
A31	A30	A29	A28	A27	A26	A25	A24	A23	A22	A21	A20	A19	A18	A17	A16
RESET:															
U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
A15	A14	A13	A12	A11	A10	A9	A8	A7	A6	A5	A4	A3	A2	A1	A0
RESET:															
U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U

U = Unaffected by reset Supervisor/User

This register is decremented by 1, 2, or 4 for each successful operand transfer from source to destination locations. When the BTC decrements to zero and no error has occurred, the CSR DONE bit is set. In the external request mode, the $\overline{\text{DONEx}}$ handshake line is also asserted when the BTC is decremented to zero.

If the operand size is byte, then the register is always decremented by 1. If the operand size is word and the starting count is even word, the register is decremented by 2. If the operand size is word and the byte count is not a multiple of 2, the CSR CONF bit is set, and a transfer does not occur. If the operand size is long word and the count is a multiple of four, then the register is decremented by 4. If the operand size is long word and the byte count is not a multiple of 4, the CSR CONF bit is set, and a transfer does not occur. If the STR bit is set with a zero count in the BTC, the CONF bit is set, and the STR bit is cleared.

When read, this register always contains the count for the next access. If a bus error terminates the transfer, this register contains the count for the next access that would have been run had the error not occurred.

7.7.2 Channel Control Register (CCR)

The CCR controls the configuration of the DMA channel. This register is accessible in either supervisor or user space. The CCR can always be read or written to when the DMA module is enabled (i.e., the STP bit in the MCR is cleared).

CCR1, CCR2 $788, $7A8

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
INTB	INTN	INTE	ECO	SAPI	DAPI	SSIZE		DSIZE		REQ		BB		S/D	STR
RESET:															
U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	0

U = Unaffected by reset Supervisor/User

INTB—Interrupt Breakpoint

Setting the interrupt breakpoint bit sets the BRKP bit in the CSR. The logic AND of INTB and BRKP generates an interrupt request.

1 = Enables an $\overline{\text{IRQx}}$ when a breakpoint is recognized and the channel is the bus master.
0 = Does not enable an $\overline{\text{IRQx}}$ when a breakpoint is recognized and the channel is the bus master.

INTN—Interrupt Normal

1 = Enables $\overline{\text{IRQx}}$ when the channel finishes a transfer without an error condition (CSR DONE bit is set).
0 = Does not enable $\overline{\text{IRQx}}$ when the channel finishes a transfer without an error condition.

INTE—Interrupt Error

1 = Enables $\overline{\text{IRQx}}$ when the channel encounters an error on source read (CSR BES bit is set), destination write (CSR BED bit is set), or configuration for channel setup (CSR CONF bit is set).
0 = Does not enable $\overline{\text{IRQx}}$ when the channel encounters an error on source read, destination write, or configuration for channel setup.

ECO—External Control Option

If request generation is programmed to be internal (REQ bits = 00), this bit has no effect.

Single-Address Mode—This bit defines the direction of transfer.

1 = If request generation is programmed to be external (REQ = 1x), the requesting device receives the data (read from memory), and the control signals ($\overline{\text{DREQx}}$, $\overline{\text{DACKx}}$, and $\overline{\text{DONEx}}$) are used by the requesting device to write data during the source (read) portion of the transfer.
0 = If request generation is programmed to be external (REQ = 1x), the requesting device provides the data (write to memory), and the control signals ($\overline{\text{DREQx}}$, $\overline{\text{DACKx}}$, and $\overline{\text{DONEx}}$) are used by the requesting device to provide data during the destination (write) portion of the transfer.

Dual-Address Mode—This bit defines which device generates requests.

1 = If request generation is programmed to be external (REQ = 1x), the source device generates the request, and the control signals ($\overline{\text{DREQx}}$, $\overline{\text{DACKx}}$, and $\overline{\text{DONEx}}$) are part of the source (read) portion of the transfer.
0 = If request generation is programmed to be external (REQ = 1x), the destination device generates the request, and the control signals ($\overline{\text{DREQx}}$, $\overline{\text{DACKx}}$, and $\overline{\text{DONEx}}$) are part of the destination (write) portion of the transfer.

SAPI—Source Address Pointer Increment

1 = The SAR is incremented by 1, 2, or 4 after each transfer, according to the source size. The address that is written into the SAR points to a memory block and is incremented to complete the data transfer.

0 = The SAR is not incremented during operand transfer. The address that is written into the SAR points to a peripheral device and is used for the complete data transfer.

DAPI—Destination Address Pointer Increment

1 = The DAR is incremented by 1, 2, or 4 after each transfer, according to the source size. The address that is written into the DAR points to a memory block and is incremented to complete the data transfer.

0 = The DAR is not incremented during operand transfer. The address that is written into the DAR points to a peripheral device and is used for the complete data transfer.

SSIZE—Source Size Control Field

This field controls the size of the source (read) bus cycle that the DMA channel is running. Table 7-1 defines these bits.

Table 7-1. SSIZEx Encoding

Bit 9	Bit 8	Definition
0	0	Long Word
0	1	Byte
1	0	Word
1	1	Not Used

DSIZE—Destination Size Control Field

This field controls the size of the destination (write) bus cycle that the DMA channel is running. Table 7-2 defines these bits.

Table 7-2. DSIZEx Encoding

Bit 7	Bit 6	Definition
0	0	Long Word
0	1	Byte
1	0	Word
1	1	Not Used

REQ—Request Generation Field

This field controls the mode of operation the DMA channel uses to make an operand transfer request. Table 7-3 defines these bits.

Table 7-3. REQx Encoding

Bit 5	Bit 4	Definition
0	0	Internal Request at Programmable Rate
0	1	Reserved
1	0	External Request Burst Transfer Mode
1	1	External Request Cycle Steal

BB—Bus Bandwidth Field

This field controls the percentage of 1024 clock periods of the IMB that the DMA channel can use during internal requests only (REQx = 00). Table 7-4 defines these bits.

Table 7-4. BBx Encoding and Bus Bandwidth

BB Field			Bus Bandwidth
Bit 3	Bit 2	Definition	(Clock Periods)
0	0	25%	256
0	1	50%	512
1	0	75%	768
1	1	100%	1024

S/D—Single-/Dual-Address Transfer

1 = The DMA channel runs single-address transfers from a peripheral to memory or from memory to a peripheral. The destination holding register is not used for these transfers because the data is transferred directly into the destination location. The MC68349 on-chip peripherals do not support single-address transfers.

0 = The DMA channel runs dual-address transfers.

STR—Start

This bit is cleared by a hardware/software reset, writing a logic zero, or setting one of the following CSR bits: DONE, BES, BED, CONF, or BRKP. The STR bit cannot be set when the CSR IRQ bit is set. The DMA channel cannot be started until the CSR DONE, BES, BED, CONF, and BRKP bits are cleared.

Internal Request Mode:

1 = The DMA transfer starts as soon as this bit is set.

0 = The DMA transfer can be stopped by clearing this bit.

External Request Mode:

1 = Setting this bit allows the DMA to start the transfer when a $\overline{\text{DREQx}}$ input is received from an external device.

0 = The DMA transfer can be stopped by clearing this bit.

NOTE

If any fields in the CCR are modified while the channel is active, that change is effective immediately. To avoid any problems with changing the setup for the DMA channel, a zero should be written to the STR bit in the CCR to halt the DMA channel at the end of the current bus cycle.

7.7.3 Channel Status Register (CSR)

The CSR contains the channel status information. This register is accessible in either supervisor or user space. The CSR can always be read or written to when the DMA module is enabled (i.e., the STP bit in the MCR is cleared).

CSR1, CSR2 $78A, $7AA

7	6	5	4	3	2	1	0
IRQ	DONE	BES	BED	CONF	BRKP	0	0
RESET							
0	0	0	0	0	0	0	0

Supervisor/User

IRQ—Interrupt Request

This bit is the logical OR of the DONE, BES, BED, CONF, and BRKP bits and is cleared when they are all cleared. IRQ is positioned to allow conditional testing as a signed binary integer. The state of this bit is not affected by the interrupt enable bits in the CCR. The STR bit in the CCR cannot be set when this bit is set; all error status bits, except the BRKP bit, must be cleared before the STR bit can be set.

1 = An interrupt condition has occurred.
0 = An interrupt condition has not occurred.

DONE—DMA Done

1 = The DMA channel has terminated normally.
0 = The DMA channel has not terminated normally. This bit is cleared by writing a logic one or by a hardware reset. Writing a zero has no effect.

BES—Bus Error on Source

1 = The DMA channel has terminated with a bus error during the read bus cycle.
0 = The DMA channel has not terminated with a bus error during the read bus cycle. This bit is cleared by writing a logic one or by a hardware reset. Writing a zero has no effect.

BED—Bus Error on Destination

1 = The DMA channel has terminated with a bus error during the write bus cycle.
0 = The DMA channel has not terminated with a bus error during the write bus cycle. This bit is cleared by writing a logic one or by a hardware reset. Writing a zero has no effect.

CONF—Configuration Error

A configuration error results when either the SAR or the DAR contains an address that does not match the port size specified in the CCR and the BTC register does not match the larger port size or is zero.

1 = The CCR STR bit is set, and a configuration error is present.
0 = The CCR STR bit is set, and no configuration error exists. This bit is cleared by writing a logic one or by a hardware reset. Writing a zero has no effect.

BRKP—Breakpoint

1 = The breakpoint signal was set during a DMA transfer.
0 = The breakpoint signal was not set during a DMA transfer. This bit is cleared by writing a logic one or by a hardware reset. Writing a zero has no effect.

Bits 1, 0—Reserved by Motorola

NOTE

The CSR is cleared by writing $7C to its location. The DMA channel cannot be started until the CSR DONE, BES, BED, CONF and BRKP bits are cleared.

7.7.4 Destination Address Register (DAR)

The DAR is a 32-bit register that contains the address of the destination operand used by the DMA to write to memory or peripheral registers. This register is accessible in either supervisor or user space. The DAR can always be read or written to when the DMA module is enabled (i.e., the STP bit in the MCR is cleared).

DAR1, DAR2 $790, $7B0

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
A31	A30	A29	A28	A27	A26	A25	A24	A23	A22	A21	A20	A19	A18	A17	A16
RESET:															
U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
A15	A14	A13	A12	A11	A10	A9	A8	A7	A6	A5	A4	A3	A2	A1	A0
RESET:															
U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U

U = Unaffected by reset Supervisor/User

During the DMA write cycle, this register drives the address on the address bus. This register can be programmed to increment (CCR DAPI bit set) or remain constant (CCR DAPI bit cleared) after each operand transfer.

The register is incremented using unsigned arithmetic and will roll over if overflow occurs. For example, if a register contains $FFFFFFFF and is incremented by 1, it will roll over to $00000000. This register can be incremented by 1, 2, or 4, depending on the size of the operand and the starting address. If the operand size is byte, the register is always incremented by 1. If the operand size is word and the starting address is word aligned, the

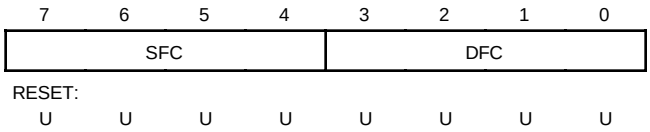
register is incremented by 2. If the operand size is long word and the address is word aligned, the register is incremented by 4. The DAR value must be aligned to an word boundary if the transfer size is word or long word; otherwise, the CSR CONF bit is set, and the transfer does not occur.

When read, this register always contains the next destination address. If a bus error terminates the transfer, this register contains the next destination address that would have been run had the error not occurred.

7.7.5 Function Code Register (FCR)

The FCR contains the source and destination function codes for the channel. This register is accessible in either supervisor or user space. The FCR can always be read or written to when the DMA module is enabled (i.e., the STP bit in the MCR is cleared).

FCR1, FCR2 $78B, $7AB



U = Unaffected by reset Supervisor/User

SFC—Source Function Code Field

This field specifies the source access to a certain address space type. The source function code bits 3–0 are defined in Table 7-5.

DFC—Destination Function Code Field

This field specifies the destination access to a certain address space type. The destination function code bits 3–0 are defined in Table 7-5.

Table 7-5. Address Space Encoding

Source/Destination Function Code Bits				
3	**2**	**1**	**0**	**Address Spaces**
0	0	0	0	Reserved (Motorola)
0	0	0	1	User Data Space
0	0	1	0	User Program Space
0	0	1	1	Reserved (User)
0	1	0	0	Reserved (Motorola)
0	1	0	1	Supervisor Data Space
0	1	1	0	Supervisor Program Space
0	1	1	1	CPU Space
1	x	x	x	DMA Space

NOTE

Although SFC3/DFC3 can be set for DMA transfers to distinguish the source or destination space from other data or program spaces, it is not required to be set. Since the CPU32+ currently has only 3-bit SFC and DFC capability, it cannot emulate SFC3 = 1 or DFC3 = 1 at this time. However, it is recommended that SFC3/DFC3 be set to one to distinguish DMA and CPU accesses during debug.

7.7.6 Interrupt Register (INTR)

The INTR contains the priority level for the channel interrupt request and the 8-bit vector number of the interrupt. The register can be read or written to at any time while in supervisor mode and while the DMA module is enabled (i.e., the STP bit in the MCR is cleared).

INTR1, INTR2 $784, $7A4

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	INTL			INTV							
RESET:															
0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1

Supervisor Only

Bits 15–11—Reserved by Motorola

INTL—Interrupt Level Bits

Each module that can generate interrupts has an interrupt level field. The interrupt level field contains the priority level of the interrupt for its associated channel. The priority level encoded in these bits is sent to the CPU32+ on the appropriate $\overline{IRQx}$ signal. The CPU32+ uses this value to determine servicing priority. See **Section 5 CPU030** for more information.

INTV—Interrupt Vector Bits

Each module that can generate interrupts has an interrupt vector field. The interrupt vector field contains the vector number of the interrupt for its associated channel. This 8-bit field indicates the offset from the base of the vector table where the address of the exception handler for the specified interrupt is located. The INTV field is reset to $0F, which indicates an uninitialized interrupt condition. See **Section 5 CPU030** for more information.

7.7.7 Module Configuration Register (MCR)

The MCR controls the DMA channel configuration. Each DMA channel has an MCR. This register can be either read or written when the channel is enabled and is in the supervisor state. The MCR is not affected by a CPU32+ RESET instruction.

MCR1, MCR2 $780, $7A0

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
STP	FRZ1	FRZ0	SE	0	ISM			SUPV	MAID			IARB			
RESET:															
0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0

Supervisor Only

STP—Stop Bit

1 = Setting the STP bit stops all clocks within the DMA module except for the clock from the IMB. The clock from the IMB remains active to allow the CPU32+ access to the MCR. The clock stops on the low phase of the clock and remains stopped until the STP bit is cleared by the CPU32+ or a hardware reset. Accesses to DMA module registers while in stop mode produce a bus error. The DMA module should be disabled in a known state before setting the STP bit. The STP bit should be set prior to executing the LPSTOP instruction to reduce overall power consumption.

0 = The channel operates in normal mode.

NOTE

The DMA module uses only one STP bit for both channels. A read or write to either MCR accesses the same STP control bit.

FRZ1, FRZ0—Freeze

These bits determine the action taken when the FREEZE signal is asserted on the IMB when the CPU32+ has entered background debug mode. The DMA module negates $\overline{BR}$ and keeps it negated until FREEZE is negated or reset. Table 7-6 lists the action taken for each bit combination.

Table 7-6. FRZx Control Bits

FRZ1	FRZ0	Action
0	0	Ignore FREEZE
0	1	Reserved
1	0	Freeze on Boundary*
1	1	Reserved

*The boundary is defined as any bus cycle by the DMA module.

NOTE

The DMA module uses only one set of FRZx bits for both channels. A read or write to either MCR accesses the same FRZx control bits.

SE—Single-Address Enable

This bit is implemented for future M68300 family compatibility.

1 = In single-address mode, the external data bus is driven during a DMA transfer.
0 = In single-address mode, the external data bus remains in a high-impedance state during a DMA transfer (used for intermodule DMA).

In dual-address mode, the SE bit has no effect.

Bit 11—Reserved by Motorola

ISM2–ISM0—Interrupt Service Mask

These bits contain the interrupt service mask level for the channel. When the interrupt service level on the IMB is greater than the interrupt service mask level, the DMA vacates the bus and negates $\overline{BR}$ until the interrupt service level is less than or equal to the interrupt service mask level.

NOTE

When the CPU32+ status register (SR) interrupt priority mask bits I2–I0 are at a higher level than the DMA ISM bits, the DMA channel will not start. The channel will begin operation when the level of the SR I2–I0 bits is less than or equal to the level of the DMA ISM bits.

SUPV—Supervisor/User

The value of this bit has no effect on registers permanently defined as supervisor-only access.

1 = The DMA channel registers defined as supervisor/user reside in supervisor data space and are only accessible from supervisor programs.
0 = The DMA channel registers defined as supervisor/user reside in user data space and are accessible from either supervisor or user programs.

MAID—Master Arbitration ID

These bits establish bus arbitration priority level among modules that have the capability of becoming bus master. For the MC68349, the MAID bits are used to arbitrate between DMA channel 1 and channel 2. If both channels are programmed with the same MAID level, channel 1 will have priority. These bits are implemented for future M68300 family compatibility. In the MC68349, only the SIM and the DMA can be bus masters. However, future versions of the M68300 family may incorporate other modules that may also be bus masters. For these devices, the MAID bits will be required. For the MAID bits, zero is the lowest priority and seven is the highest priority.

IARB — Interrupt Arbitration ID

Each module that generates interrupts has an IARB field. These bits are used to arbitrate for the bus in the case that two or more modules simultaneously generate an interrupt at the same priority level. No two modules can share the same IARB value. The reset value of the IARB field is $0, which prevents the DMA module from arbitrating during the interrupt acknowledge cycle. The system software should initialize the IARB field to a value from $F (highest priority) to $1 (lowest priority).

NOTE

The DMA module uses only one set of IARB bits for both channels. A read or write to either MCR accesses the same IARB control bits.

7.7.8 Source Address Register (SAR)

The SAR is a 32-bit register that contains the address of the source operand used by the DMA to access memory or peripheral registers. This register is accessible in either supervisor or user space. The SAR can always be read or written to when the DMA module is enabled (i.e., the STP bit in the MCR is cleared).

SAR1, SAR2 $78C, $7AC

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
A31	A30	A29	A28	A27	A26	A25	A24	A23	A22	A21	A20	A19	A18	A17	A16
RESET:															
U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
A15	A14	A13	A12	A11	A10	A9	A8	A7	A6	A5	A4	A3	A2	A1	A0
RESET:															
U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U

U = Unaffected by reset Supervisor/User

During the DMA read cycle, the SAR drives the address on the address bus. This register can be programmed to increment (CCR SAPI bit set) or remain constant (CCR SAPI bit cleared) after each operand transfer.

The register is incremented using unsigned arithmetic and will roll over if overflow occurs. For example, if the register contains $FFFFFFFF and is incremented by 1, it will roll over to $00000000. This register is incremented by 1, 2, or 4, depending on the size of the operand and the memory starting address. If the operand size is byte, then the register is always incremented by 1. If the operand size is word and the starting address is word aligned, then the register is incremented by 2. If the operand size is long word and the address is word aligned, then the register is incremented by 4. The SAR value must be aligned to an word boundary if the transfer size is word or long word; otherwise, the CSR CONF bit is set, and the transfer does not occur.

When read, this register always contains the next source address. If a bus error terminates the transfer, this register contains the next source address that would have been run had the error not occurred.

7.8 DATA PACKING

The internal DHR is a 32-bit register that can serve as a buffer register for the data being transferred during dual-address DMA cycles. No address is specified since this register can not be addressed by the programmer. The DHR allows the data to be packed and unpacked by the DMA during the dual-address transfer. For example, if the source operand size is byte and the destination operand size is word, then for each DMA request two byte read cycles occur, followed by one word write cycle (see Figure 7-16). The two bytes of data are buffered in the DHR until the destination (write) word cycle occurs. The DHR allows for packing and unpacking of operands for the following sizes: bytes to words, bytes to long words, words to long words, words to bytes, long words to bytes, and long words to words.

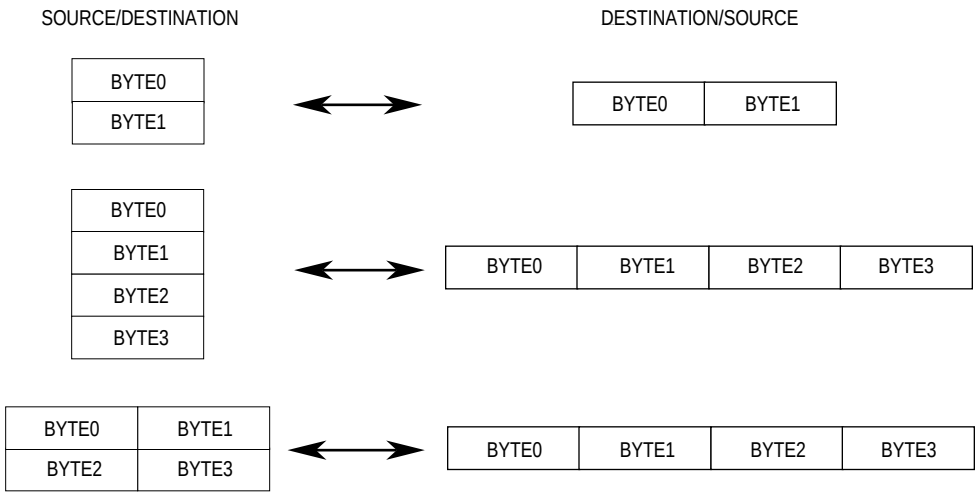


Figure 7-16. Packing and Unpacking of Operands

For normal transfers aligned with the size and address, only two bus cycles are required for each transfer: a read from the source and a write to the destination.

7.9 DMA CHANNEL INITIALIZATION SEQUENCE

The following paragraphs describe DMA channel initialization and operation. If the DMA capability of the MC68349 is being used, the initialization steps should be performed during the part initialization sequence. The mode operation steps should be performed to start a DMA transfer. The $\overline{\text{DONEx}}$ pin requires an external pullup resistor even if operating only in the internal request mode.

7.9.1 DMA Channel Configuration

The following steps can be accomplished in any order when initializing the DMA channel. These steps need to be performed for each channel used.

Module Configuration Register (MCR)

- Clear the stop bit (STP) for normal operation. (Only one STP bit exists for both channels.)
- Select whether to respond to or ignore FREEZE (FRZx bits). (Only one set of FRZx bits exists for both channels.)
- If desired, enable the external data bus operation in single-address mode (SE bit).
- Program the interrupt service mask to set the level below which interrupts are ignored during a DMA transfer (ISM bits). The channel will begin operation when the level of the CPU32+ SR I2-I0 bits is less than or equal to the level of the DMA ISM bits.
- Select the access privilege for the supervisor/user registers (SUPV bit).
- Program the master arbitration ID (MAID) to establish priority on the IMB between both DMA channels. Note that the two DMA channels should have distinct MAIDs if both channels are being used. (If they are programmed the same, channel 1 has priority.)
- Select the interrupt arbitration level for the DMA channel (IARB bits). (Only one set of IARB bits exists for both channels.)

Interrupt Register (INTR)

- Program the interrupt priority level for the channel interrupt (INTL bits).
- Program the vector number for the channel interrupt (INTV bits).

Channel Control Register (CCR)

- If desired, enable the interrupt when breakpoint is recognized and the channel is the bus master (INTB bit).
- If desired, enable the interrupt when done without an error condition (INTN bit).
- If desired, enable the interrupt when the channel encounters an error (INTE bit).
- Select the direction of transfer if in single-address mode (ECO bit), or select which device generates requests if in dual-address mode.

7.9.1.1 DMA CHANNEL OPERATION IN SINGLE-ADDRESS MODE. The following steps are required to begin a DMA transfer in single-address mode.

Channel Control Register (CCR)

- Write a zero to the start bit (STR) to prevent the channel from starting the transfer prematurely.
- Select the amount by which to increment the source address for a read cycle (SAPI bit) or the destination address for a write cycle (DAPI bit).
- Define the transfer size by selecting the source size for a read cycle (SSIZE field) or by selecting the destination size for a write cycle (DSIZE field).
- Select external burst request mode or external cycle steal request mode (REQ field).
- Set the S/D bit for signal-address transfer.

Channel Status Register (CSR)

- Clear the CSR by writing $7C into it. The DMA cannot be started until the DONE, BES, BED, CONF, and BRKP bits are cleared.

Function Code Register (FCR)

- Encode the source function code for a read cycle or the destination function code for a write cycle.

Address Register (SAR or DAR)

- Write the source address for a read cycle or the destination address for a write cycle.

Byte Transfer Counter (BTC)

- Encode the number of bytes to be transferred.

Channel Control Register (CCR)

- Write a one to the start bit (STR) to allow the transfer to begin.

7.9.1.2 DMA CHANNEL OPERATION IN DUAL-ADDRESS MODE. The following steps are required to begin a DMA transfer in dual-address mode.

Channel Control Register (CCR)

- Write a zero to the start bit (STR) to prevent the channel from starting the transfer prematurely.
- Select the amount by which to increment the source and destination addresses (SAPI and DAPI bits).
- Select the source and destination sizes (SSIZE and DSIZE fields).
- Select internal request, external burst request mode, or external cycle steal request mode (REQ field).
- If using internal request, select the amount of bus bandwidth to be used by the DMA (BB field).
- Clear the S/D bit for dual-address transfer.

Channel Status Register (CSR)

- Clear the CSR by writing $7C into it. The DMA cannot be started until the DONE, BES, BED, CONF, and BRKP bits are cleared.

Function Code Register (FCR)

- Encode the source and destination function codes.

Address Registers (SAR and DAR)

- Write the source and destination addresses.

Byte Transfer Counter (BTC)

- Encode the number of bytes to be transferred.

Channel Control Register (CCR)

- Write a one to the start bit (STR) to allow the transfer to begin.

7.9.2 DMA Channel Example Configuration Code

The following are examples of configuration sequences for a DMA channel in single- and dual-addressing modes.

```
*****************************************************************************
* MC68349 basic DMA channel register initialization example code.
* This code is used to initialize the 68349's internal DMA channel
* registers, providing basic functions for operation.
* The code sets up channel 1 for external burst request generation,
* single-address mode, long word size transfers.
* Control signals are asserted on the DMA read cycle.
*****************************************************************************
```

Example 1: External Burst Request Generation, Single-Address Transfers.

```
*****************************************************************************
* SIM49 equates
*****************************************************************************
MBAR        EQU  $0003FF00  Address of SIM49 Module Base Address Reg.
MODBASE     EQU  $FFFFF000  SIM49 MBAR address value

*****************************************************************************
* DMA Channel 1 equates
DMACH1      EQU  $780       Offset from MBAR for channel 1 regs
DMAMCR1     EQU  $0         MCR for channel 1

* Channel 1 register offsets from channel 1 base address
DMAINT1     EQU  $4         interrupt register channel 1
DMACCR1     EQU  $8         control register channel 1
DMACSR1     EQU  $A         status register channel 1
DMAFCR1     EQU  $B         function code register channel 1
DMASAR1     EQU  $C         source address register channel 1
DMADAR1     EQU  $10        destination address register channel 1
DMABTC1     EQU  $14        byte transfer count register channel 1
SARADD      EQU  $10000     source address
NUMBYTE     EQU  $C         number of bytes to transfer

*****************************************************************************
*****************************************************************************
* Initialize DMA Channel 1
*****************************************************************************
        LEA    MODBASE+DMACH1,A0   Pointer to channel 1

* Initialize DMA channel 1 MCR
* Normal Operation, ignore FREEZE, single-address mode. ISM field at 2. Make
* sure CPU32+ SR I2-I0 bits are less than or equal to ISM bits for channel startup.
* Supervisor/user reg. unrestricted, MAID field at 7. IARB priority at 1.
        MOVE.W      #$1271,(A0)
```

```
* Clear channel control reg.
* Clear STR (start) bit to prevent the channel from starting a transfer early.
        CLR.W       DMACCR1(A0)

* Initialize interrupt reg.
* Interrupt priority at 7, interrupt vector at $42.
        MOVE.W      #$0742,DMAINT1(A0)

* Initialize channel status reg.
* Clear the DONE, BES, BED, CONF and BRKP bits to allow channel to startup.
        MOVE.B      #$7C,DMACSR1(A0)

* Initialize function code reg.
* DMA space, user data space for source.
        MOVE.B      #$99,DMAFCR1(A0)

* Initialize source operand address
* Source address is equal to $10000.
        MOVE.L      SARADD,DMASAR1(A0)

* Initialize the byte transfer count reg.
* The number of bytes to be transferred is $C or 3 long words
        MOVE.L      NUMBYTE,DMABTC1(A0)

* Channel control reg. init. and Start DMA transfers
* No interrupts are enabled, source (read) cycle. Increment source
* address, source size is long word, REQ is external burst request.
* Single-address mode, start the DMA transfers.
        MOVE.W      #$1823,DMACCR1(A0)

*************************************************************************
        END
*************************************************************************
```

Example 2: Internal Request Generation, Memory to Memory Transfers.

```
*************************************************************************
* MC68349 basic DMA channel register initialization example code.
* This code is used to initialize the 68349's internal DMA channel
* registers, providing basic functions for operation.
* The code sets up channel 1 for internal request generation
* memory to memory transfers.
*************************************************************************
*************************************************************************
* SIM49 equates
```

```
*************************************************************************
MBAR        EQU   $0003FF00   Address of SIM49 Module Base Address Reg.
MODBASE     EQU   $FFFFF000   SIM49 MBAR address value

*************************************************************************
* DMA Channel 1 equates
DMACH1      EQU   $780        Offset from MBAR for channel 1 regs
DMAMCR1     EQU   $0          MCR for channel 1

* Channel 1 register offsets from channel 1 base address
DMAINT1     EQU   $4          interrupt register channel 1
DMACCR1     EQU   $8          control register channel 1
DMACSR1     EQU   $A          status register channel 1
DMAFCR1     EQU   $B          function code register channel 1
DMASAR1     EQU   $C          source address register channel 1
DMADAR1     EQU   $10         destination address register channel 1
DMABTC1     EQU   $14         byte transfer count register channel 1
SARADD      EQU   $6000             source address
DARADD      EQU   $8000             destination address
NUMBYTE     EQU   $E          number of bytes to transfer

*************************************************************************
*************************************************************************
* Initialize DMA Channel 1
*************************************************************************
        LEA    MODBASE+DMACH1,A0   Pointer to channel 1

* Initialize DMA channel 1 MCR
* Normal Operation, ignore FREEZE, dual-address mode. ISM field at 3. Make
* sure CPU32+ SR I2-I0 bits are less than or equal to ISM bits for channel startup.
* Supervisor/user reg. unrestricted, MAID field at 3. IARB priority at 4.
        MOVE.W      #$0334,(A0)

* Clear channel control reg.
* Clear STR (start) bit to prevent the channel from starting a transfer early.
        CLR.W       DMACCR1(A0)

* Initialize interrupt reg.
* Interrupt priority at 7, interrupt vector at $42.
        MOVE.W      #$0742,DMAINT1(A0)

* Initialize channel status reg.
* Clear the DONE, BES, BED, CONF and BRKP bits to allow channel to startup.
        MOVE.B      #$7C,DMACSR1(A0)
```

```
* Initialize function code reg.
* DMA space, supervisor data space for source and destination.
        MOVE.B      #$DD,DMAFCR1(A0)

* Initialize source operand address
* Source address is equal to $6000.
        MOVE.L      SARADD,DMASAR1(A0)

* Initialize destination operand address
* Destination address is equal to $8000.
        MOVE.L      DARADD,DMADAR1(A0)

* Initialize the byte transfer count reg.
* The number of bytes to be transferred is $E or 7 words
        MOVE.L      NUMBYTE,DMABTC1(A0)

* Channel control reg. init. and Start DMA transfers
* No interrupts are enabled, destination (write) cycle. Increment source and
* destination addresses,source size is word, destination size is word.
* REQ is internal. 100% of bus bandwidth, dual-address transfers,
* start the DMA transfers.
        MOVE.W      #$0E8D,DMACCR1(A0)

*************************************************************************
        END
*************************************************************************
```

Example 3: Internal Request Generation, Memory Block Initialization.

```
*************************************************************************
* MC68349 basic DMA channel register initialization example code.
* This code is used to initialize the 68349's internal DMA channel
* registers, providing basic functions for operation.
* The code sets up channel 1 for internal request generation
* to perform a memory block initialization for 100 bytes.
*************************************************************************
*************************************************************************
* SIM49 equates
*************************************************************************
MBAR        EQU   $0003FF00   Address of SIM49 Module Base Address Reg.
MODBASE     EQU   $FFFFF000   SIM49 MBAR address value

*************************************************************************
* DMA Channel 1 equates
DMACH1      EQU   $780        Offset from MBAR for channel 1 regs
DMAMCR1     EQU   $0          MCR for channel 1
```

```
* Channel 1 register offsets from channel 1 base address
DMAINT1     EQU   $4          interrupt register channel 1
DMACCR1     EQU   $8          control register channel 1
DMACSR1     EQU   $A          status register channel 1
DMAFCR1     EQU   $B          function code register channel 1
DMASAR1     EQU   $C          source address register channel 1
DMADAR1     EQU   $10         destination address register channel 1
DMABTC1     EQU   $14         byte transfer count register channel 1
SARADD      EQU   $6000       source address
DARADD      EQU   $8000       destination address
NUMBYTE     EQU   $64         number of bytes to transfer

*****************************************************************************
*****************************************************************************
* Initialize DMA Channel 1
*****************************************************************************
        LEA    MODBASE+DMACH1,A0   Pointer to channel 1

* Initialize DMA channel 1 MCR
* Normal Operation, ignore FREEZE, dual-address mode. ISM field at 3. Make
* sure CPU32+ SR I2-I0 bits are less than or equal to ISM bits for channel
* startup.Supervisor/user reg. unrestricted, MAID field at 3.
* IARB priority at 4.
        MOVE.W      #$0334,(A0)

* Clear channel control reg.
* Clear STR (start) bit to prevent the channel from starting a transfer early.
        CLR.W       DMACCR1(A0)

* Initialize interrupt reg.
* Interrupt priority at 7, interrupt vector at $42.
        MOVE.W      #$0742,DMAINT1(A0)

* Initialize channel status reg.
* Clear the DONE, BES, BED, CONF and BRKP bits to allow channel to startup.
        MOVE.B      #$7C,DMACSR1(A0)

* Initialize function code reg.
* DMA space, supervisor data space for source and destination.
        MOVE.B      #$DD,DMAFCR1(A0)

* Initialize source operand address
* Source address is equal to $6000.
        MOVE.L      SARADD,DMASAR1(A0)
```

```
* Initialize destination operand address
* Destination address is equal to $8000.
        MOVE.L      DARADD,DMADAR1(A0)

* Initialize the byte transfer count register
* The number of bytes to be transferred is $64 or 50 words
        MOVE.L      NUMBYTE,DMABTC1(A0)

* Channel control reg. init. and Start DMA transfers
* No interrupts are enabled, destination (write) cycle.
* Source address is not incremented. Increment the destination address.
* Source size is word, destination size is word. REQ is internal.
* 100% of bus bandwidth, dual-address transfers, start the DMA transfers.
        MOVE.W      #$068D,DMACCR1(A0)

*************************************************************************
        END
*************************************************************************
```

Example 4: Cycle Steal Request Generation, Dual-Address Transfers.

```
*************************************************************************
* MC68349 basic DMA channel register initialization example code.
* This code is used to initialize the 68349's internal DMA channel
* registers, providing basic functions for operation.
* The code sets up channel 1 for external cycle steal request generation,
* dual-address transfers. DMA 16-bit wide data from an odd address to an
* even address. Control signals are asserted on the DMA read cycle.
*************************************************************************
*************************************************************************
* SIM49 equates
*************************************************************************
MBAR        EQU  $0003FF00  Address of SIM49 Module Base Address Reg.
MODBASE     EQU  $FFFFF000  SIM49 MBAR address value

*************************************************************************
* DMA Channel 1 equates
DMACH1      EQU  $780       Offset from MBAR for channel 1 regs
DMAMCR1     EQU  $0         MCR for channel 1

* Channel 1 register offsets from channel 1 base address
DMAINT1     EQU  $4         interrupt register channel 1
DMACCR1     EQU  $8         control register channel 1
DMACSR1     EQU  $A         status register channel 1
DMAFCR1     EQU  $B         function code register channel 1
DMASAR1     EQU  $C         source address register channel 1
DMADAR1     EQU  $10        destination address register channel 1
DMABTC1     EQU  $14        byte transfer count register channel 1
```

```
SARADD    EQU  $6001        source address is an ODD address
DARADD    EQU  $10000       destination address is and EVEN address
NUMBYTE   EQU  $14          number of bytes to transfer

*************************************************************************
*************************************************************************
* Initialize DMA Channel 1
*************************************************************************
        LEA    MODBASE+DMACH1,A0   Pointer to channel 1

* Initialize DMA channel 1 MCR
* Normal Operation, ignore FREEZE, dual-address mode. ISM field at 0. Make
* CPU32+ SR I2-I0 bits are less than or equal to ISM bits for channel startup.
* Supervisor/user reg. unrestricted, MAID field at 4. IARB priority at 8.
        MOVE.W      #$00C8,(A0)

* Clear channel control reg.
* Clear STR (start) bit to prevent the channel from starting a transfer early.
        CLR.W       DMACCR1(A0)

* Initialize interrupt reg.
* Interrupt priority at 7, interrupt vector at $42.
        MOVE.W      #$0742,DMAINT1(A0)

* Initialize channel status reg.
* Clear the DONE, BES, BED, CONF and BRKP bits to allow channel to startup.
        MOVE.B      #$7C,DMACSR1(A0)

* Initialize function code reg.
* DMA space, supervisor data space for source and destination.
        MOVE.B      #$DD,DMAFCR1(A0)

* Initialize source operand address
* Source address is equal to $6001, and odd address.
        MOVE.L      SARADD,DMASAR1(A0)

* Initialize destination operand address
* Destination address is equal to $10000, and even address.
        MOVE.L      DARADD,DMADAR1(A0)

* Initialize the byte transfer count register
* The number of bytes to be transferred is $14 or 20 bytes
        MOVE.L      NUMBYTE,DMABTC1(A0)
```

```
* Channel control reg. init. and Start DMA transfers
* No interrupts are enabled, source (read) cycle.
* Increment the source and destination addresses.
* Source size is byte, destination size is word. REQ is external cycle steal.
* dual-address transfers, start the DMA transfers.
        MOVE.W      #$1DB1,DMACCR1(A0)

*******************************************************************************
        END
*******************************************************************************
```

SECTION 8
SERIAL MODULE

The MC68349 serial module is a dual universal asynchronous/synchronous receiver/transmitter that interfaces directly to the CPU32+ processor via the intermodule bus (IMB). The serial module, shown in Figure 8-1, consists of the following major functional areas:

- Two Independent Serial Communication Channels (A and B)
- Baud Rate Generator Logic
- Internal Channel Control Logic
- Interrupt Control Logic

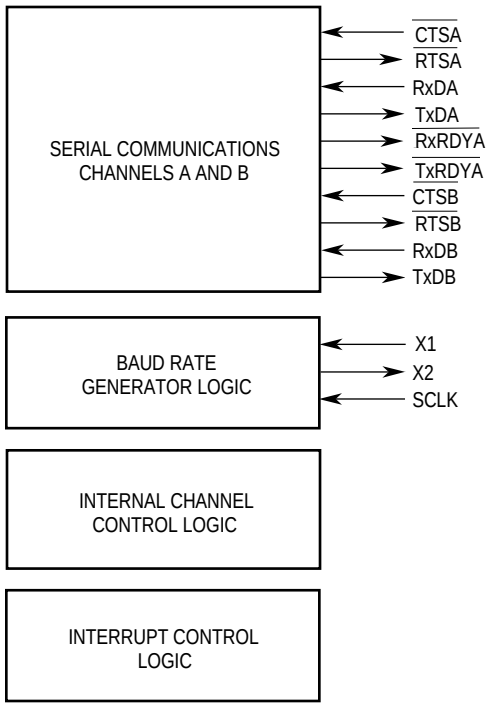


Figure 8-1. Simplified Block Diagram

8.1 MODULE OVERVIEW

Features of the serial module are as follows:

- Two, Independent, Full-Duplex Asynchronous/Synchronous Receiver/Transmitter Channels
- Maximum Data Transfer Rate:
  - —1× mode: 9.8 Mbps @25 MHz CLKOUT
  - —16× mode: 612 kbps @25 MHz CLKOUT
- Quadruple-Buffered Receiver
- Double-Buffered Transmitter
- Independently Programmable Baud Rate for Each Receiver and Transmitter Selectable from:
  - —19 Fixed Rates: 50 to 76.8k Baud
  - —External 1× Clock or 16× Clock
- Programmable Data Format:
  - —Five to Eight Data Bits Plus Parity
  - —Odd, Even, No Parity, or Force Parity
  - —Nine-Sixteenths to Two Stop Bits Programmable in One-Sixteenth Bit Increments
- Programmable Channel Modes:
  - —Normal (Full Duplex)
  - —Automatic Echo
  - —Local Loopback
  - —Remote Loopback
- Automatic Wakeup Mode for Multidrop Applications
- Seven Maskable Interrupt Conditions
- Parity, Framing, and Overrun Error Detection
- False-Start Bit Detection
- Line-Break Detection and Generation
- Detection of Breaks Originating in the Middle of a Character
- Start/End Break Interrupt/Status
- On-Chip Crystal Oscillator

8.1.1 Serial Communication Channels A and B

Each communication channel provides a full-duplex asynchronous/synchronous receiver and transmitter using an operating frequency independently selected from a baud rate generator or an external clock input.

The transmitter accepts parallel data from the IMB, converts it to a serial bit stream, inserts the appropriate start, stop, and optional parity bits, then outputs a composite serial data stream on the channel transmitter serial data output (TxDx). Refer to **8.3.2.1 Transmitter** for additional information.

The receiver accepts serial data on the channel receiver serial data input (RxDx), converts it to parallel format, checks for a start bit, stop bit, parity (if any), or break condition, and transfers the assembled character onto the IMB during read operations. Refer to **8.3.2.2 Receiver** for additional information.

8.1.2 Baud Rate Generator Logic

The crystal oscillator operates directly from a 3.6864-MHz crystal connected across the X1 input and the X2 output or from an external clock of the same frequency connected to X1. The clock serves as the basic timing reference for the baud rate generator and other internal circuits.

The baud rate generator operates from the oscillator or external TTL clock input and is capable of generating 19 commonly used data communication baud rates ranging from 50 to 76.8k by producing internal clock outputs at 16 times the actual baud rate. Refer to **8.2 Serial Module Signal Definitions** and **8.3.1 Baud Rate Generator** for additional information.

The external clock input (SCLK), which bypasses the baud rate generator, provides a synchronous clock mode of operation when used as a divide-by-1 clock and an asynchronous clock mode when used as a divide-by-16 clock. The external clock input allows the user to use SCLK as the only clock source for the serial module if multiple baud rates are not required.

8.1.3 Internal Channel Control Logic

The serial module receives operation commands from the host and, in turn, issues appropriate operation signals to the internal serial module control logic. This mechanism allows the registers within the module to be accessed and various commands to be performed. Refer to **8.4 Register Description and Programming** for additional information.

8.1.4 Interrupt Control Logic

Seven interrupt request ($\overline{IRQ7}$–$\overline{IRQ1}$) signals are provided to notify the CPU32+ that an interrupt has occurred. These interrupts are described in **8.4 Register Description and Programming**. The interrupt status register (ISR) is read by the CPU32+ to determine all

currently active interrupt conditions. The interrupt enable register (IER) is programmable to mask any events that can cause an interrupt.

8.1.5 Comparison of the Serial Module to the MC68681

The serial module is code compatible with the MC68681 with some modifications. The following paragraphs describe the differences.

The programming model is slightly altered. The supervisor/user block in the MC68349 closely follows the MC68681. The supervisor-only block has the following changes:

- The interrupt vector register is moved from supervisor/user to supervisor only at a new address.
- MR2A and MR2B are moved from a hidden address location to a location at the bottom of the programming model.

The timer/counter is eliminated as well as all associated command and status registers.

Only certain output port pins are available.

There are no IP pins on the MC68349.

RxRTS and TxRTS are more automated on the MC68349.

The XTAL_RDY bit in the ISR should be polled until it is cleared to prevent an unstable frequency from being applied to the baud rate generator. The following pseudocode is an example:

```
if (XTAL_RDY==0)
   begin
     write CSR
   end
else
   begin
      wait
      jump loop
   end
```

8.2 SERIAL MODULE SIGNAL DEFINITIONS

The following paragraphs contain a brief description of the serial module signals. Figure 8-2 shows both the external and internal signal groups.

NOTE

The terms *assertion* and *negation* are used throughout this section to avoid confusion when dealing with a mixture of active-low and active-high signals. The term *assert* or *assertion* indicates that a signal is active or true, independent of the level represented by a high or low voltage. The term *negate* or *negation* indicates that a signal is inactive or false.

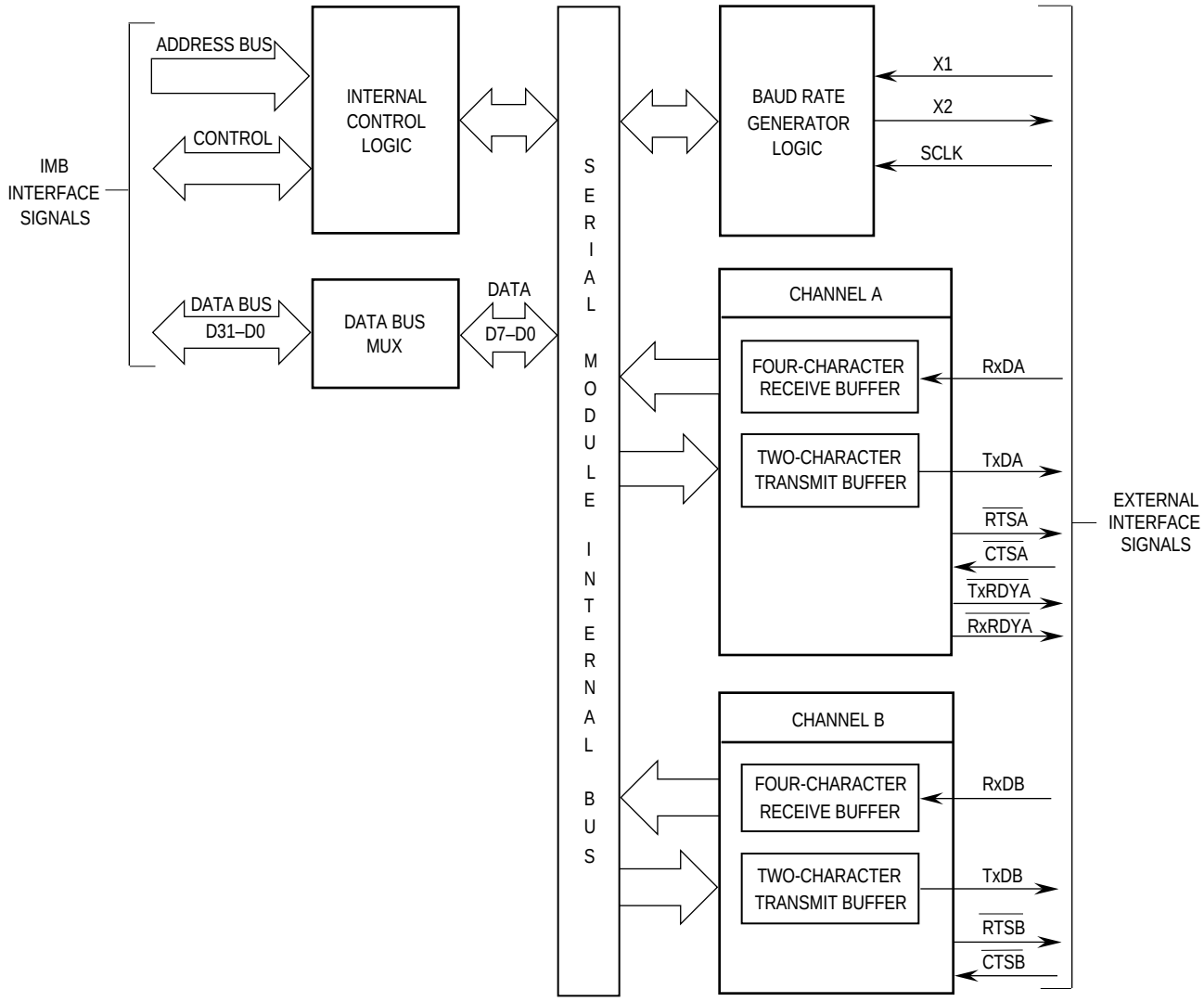


Figure 8-2. External and Internal Interface Signals

8.2.1 Crystal Input or External Clock (X1)

This input is one of two connections to a crystal or a single connection to an external clock. A crystal or an external clock signal, at 3.6864 MHz, must be supplied when using the baud rate generator. If a crystal is used, a capacitor of approximately 10 pF should be connected from this signal to ground. If this input is not used, it must be connected to V_{CC} or GND. Refer to **Section 10 Applications** for an example of a clock driver circuit.

8.2.2 Crystal Output (X2)

This output is the additional connection to a crystal. If a crystal is used, a capacitor of approximately 5 pF should be connected from this signal to ground. If an external TTL-level clock is used on X1, the X2 output must be left open. Refer to **Section 10 Applications** for an example of a clock driver circuit.

8.2.3 External Input (SCLK)

This input can be used as the clock input for channel A and/or channel B and is programmable in the clock-select registers (CSR). When used as the receiver clock, received data is sampled on the rising edge of the clock. When used as the transmitter clock, data is output on the falling edge of the clock. If this input is not used, it must be connected to V_{CC} or GND.

8.2.4 Channel A Transmitter Serial Data Output (TxDA)

This signal is the transmitter serial data output for channel A. The output is held high ('mark' condition) when the transmitter is disabled, idle, or operating in the local loopback mode. Data is shifted out on this signal on the falling edge of the clock source, with the least significant bit transmitted first.

8.2.5 Channel A Receiver Serial Data Input (RxDA)

This signal is the receiver serial data input for channel A. Data received on this signal is sampled on the rising edge of the clock source, with the least significant bit received first.

8.2.6 Channel B Transmitter Serial Data Output (TxDB)

This signal is the transmitter serial data output for channel B. The output is held high ('mark' condition) when the transmitter is disabled, idle, or operating in the local loopback mode. Data is shifted out on this signal at the falling edge of the clock source, with the least significant bit transmitted first.

8.2.7 Channel B Receiver Serial Data Input (RxDB)

This signal is the receiver serial data input for channel B. Data on this signal is sampled on the rising edge of the clock source, with the least significant bit received first.

8.2.8 Channel A Request-To-Send ($\overline{\text{RTSA}}$)

This active-low output signal is programmable as the channel A request-to-send or as a dedicated parallel output.

$\overline{\text{RTSA}}$

When used for this function, this signal can be programmed to be automatically negated and asserted by either the receiver or transmitter. When connected to the clear-to-send ($\overline{\text{CTSx}}$) input of a transmitter, this signal can be used to control serial data flow.

OP0

When used for this function, this output is controlled by bit 0 in the output port data register (OP).

8.2.9 Channel B Request-To-Send ($\overline{\text{RTSB}}$)

This active-low output signal is programmable as the channel B request-to-send or as a dedicated parallel output.

$\overline{\text{RTSB}}$

When used for this function, this signal can be programmed to be automatically negated and asserted by either the receiver or transmitter. When connected to the $\overline{\text{CTSx}}$ input of a transmitter, this signal can be used to control serial data flow.

OP1

When used for this function, this output is controlled by bit 1 in the OP.

8.2.10 Channel A Clear-To-Send ($\overline{\text{CTSA}}$)

This active-low input is the channel A clear-to-send.

8.2.11 Channel B Clear-To-Send ($\overline{\text{CTSB}}$)

This active-low input is the channel B clear-to-send.

8.2.12 Channel A Transmitter Ready ($\overline{\text{TxRDYA}}$)

This active-low output signal is programmable as the channel A transmitter ready or as a dedicated parallel output and cannot be masked by the IER.

$\overline{\text{TxRDYA}}$

When used for this function, this signal reflects the complement of the status of bit 2 of the channel A status register (SRA). This signal can be used to control parallel data flow by acting as an interrupt to indicate when the transmitter contains a character.

OP6

When used for this function, this output is controlled by bit 6 in the OP.

8.2.13 Channel A Receiver Ready ($\overline{\text{RxRDYA}}$)

This active-low output signal is programmable as the channel A receiver ready, channel A first-in-first-out (FIFO) full indicator, or a dedicated parallel output and cannot be masked by the IER.

$\overline{\text{RxRDYA}}$

When used for this function, this signal reflects the complement of the status of bit 1 of the ISR. This signal can be used to control parallel data flow by acting as an interrupt to indicate when the receiver contains a character.

$\overline{\text{FFULLA}}$

When used for this function, this signal reflects the complement of the status of bit 1 of the ISR. This signal can be used to control parallel data flow by acting as an interrupt to indicate when the receiver FIFO is full.

OP4

When used for this function, this output is controlled by bit 4 in the OP.

8.3 OPERATION

The following paragraphs describe the operation of the baud rate generator, transmitter and receiver, and other functional operating modes of the serial module.

8.3.1 Baud Rate Generator

The baud rate generator consists of a crystal oscillator, baud rate generator, and clock selectors (see Figure 8-3). The crystal oscillator operates directly from a 3.6864-MHz crystal or from an external clock of the same frequency. The SCLK input bypasses the baud rate generator and provides a synchronous clock mode of operation when used as a divide-by-1 clock and an asynchronous clock mode when used as a divide-by-16 clock. The clock is selected by programming the CSR for each channel.

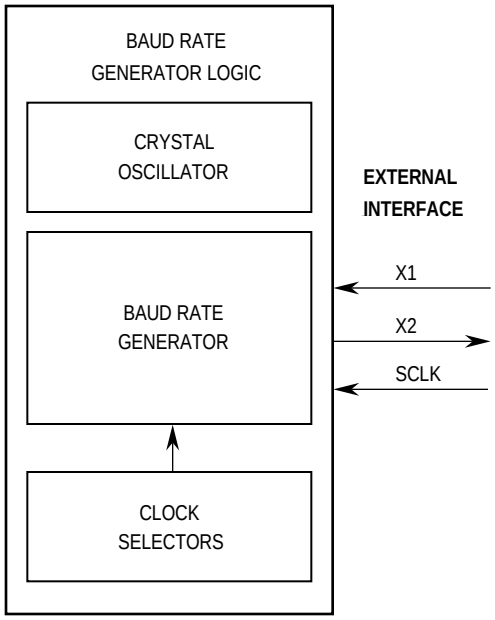
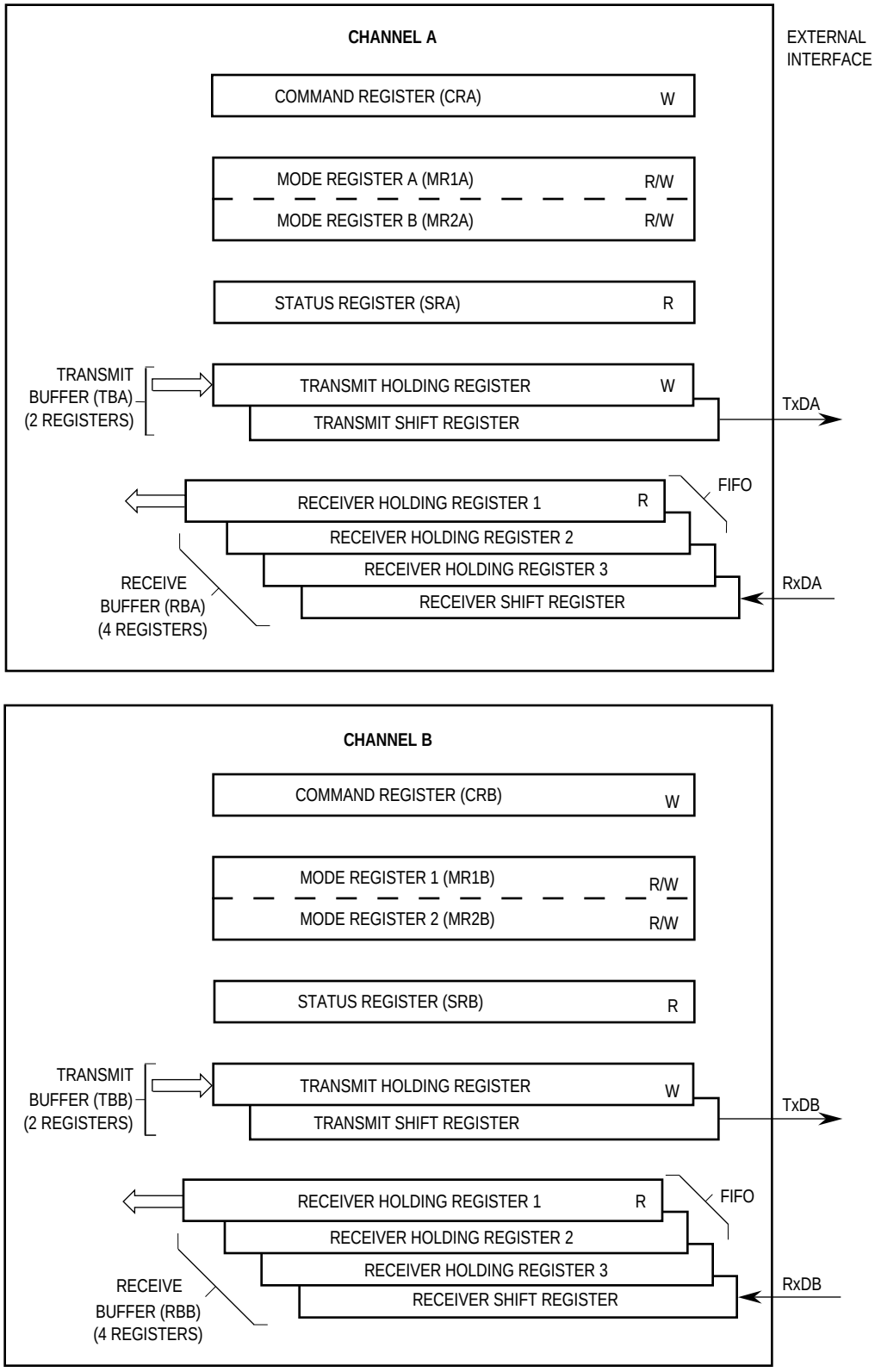


Figure 8-3. Baud Rate Generator Block Diagram

8.3.2 Transmitter and Receiver Operating Modes

The functional block diagram of the transmitter and receiver, including command and operating registers, is shown in Figure 8-4. The paragraphs that follow contain descriptions for both these functions in reference to this diagram. For detailed register information, refer to **8.4 Register Description and Programming**.

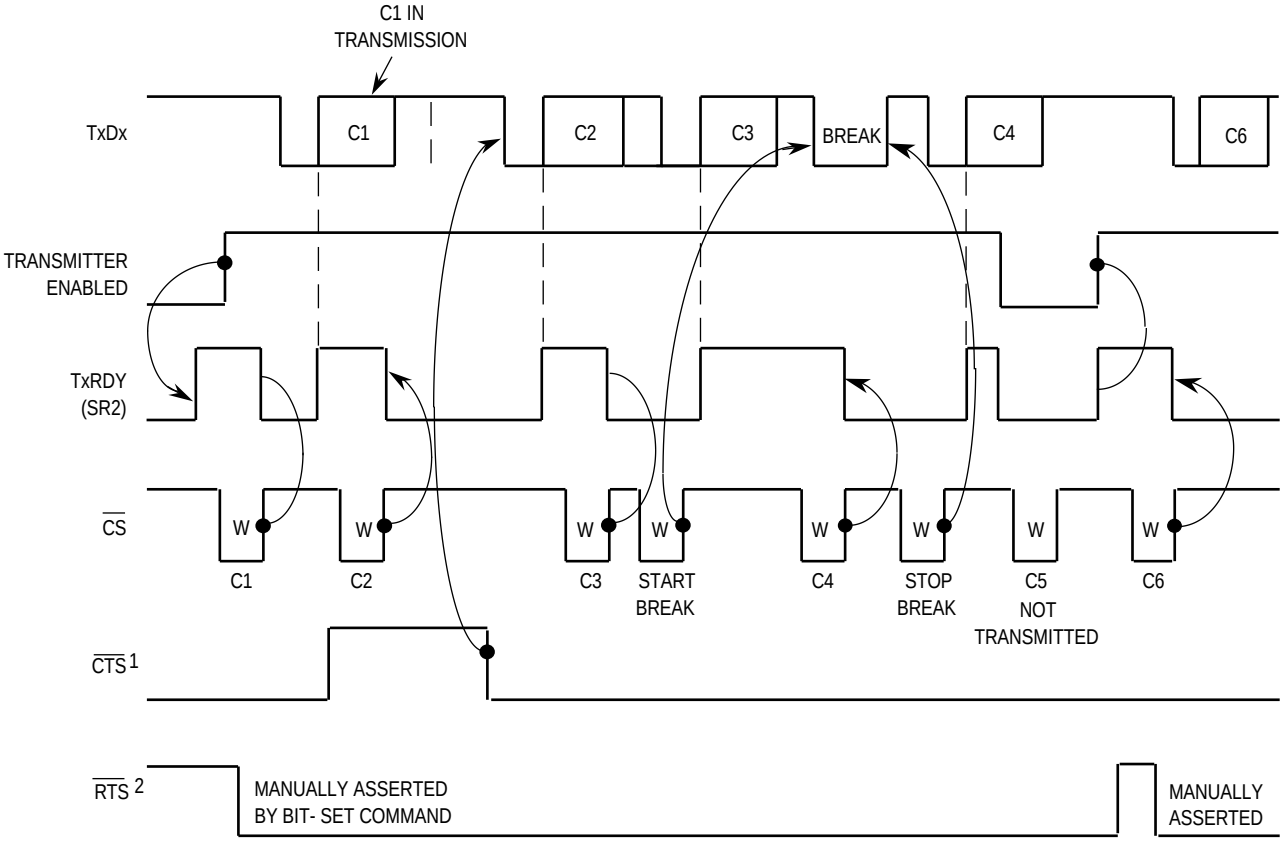


NOTE:
R/W = READ/WRITE
R = READ
W = WRITE

Figure 8-4. Transmitter and Receiver Functional Diagram

8.3.2.1 TRANSMITTER. The transmitters are enabled through their respective command registers (CR) located within the serial module. The serial module signals the CPU32+ when it is ready to accept a character by setting the transmitter-ready bit (TxRDY) in the channel's status register (SR). Functional timing information for the transmitter is shown in Figure 8-5.

The transmitter converts parallel data from the CPU32+ to a serial bit stream on TxDx. It automatically sends a start bit followed by the programmed number of data bits, an optional parity bit, and the programmed number of stop bits. The least significant bit is sent first. Data is shifted from the transmitter output on the falling edge of the clock source.



NOTES:

1. TIMING SHOWN FOR MR2(4) = 1
2. TIMING SHOWN FOR MR2(5) = 1
3. C_N = TRANSMIT CHARACTER
4. W = WRITE

Figure 8-5. Transmitter Timing Diagram

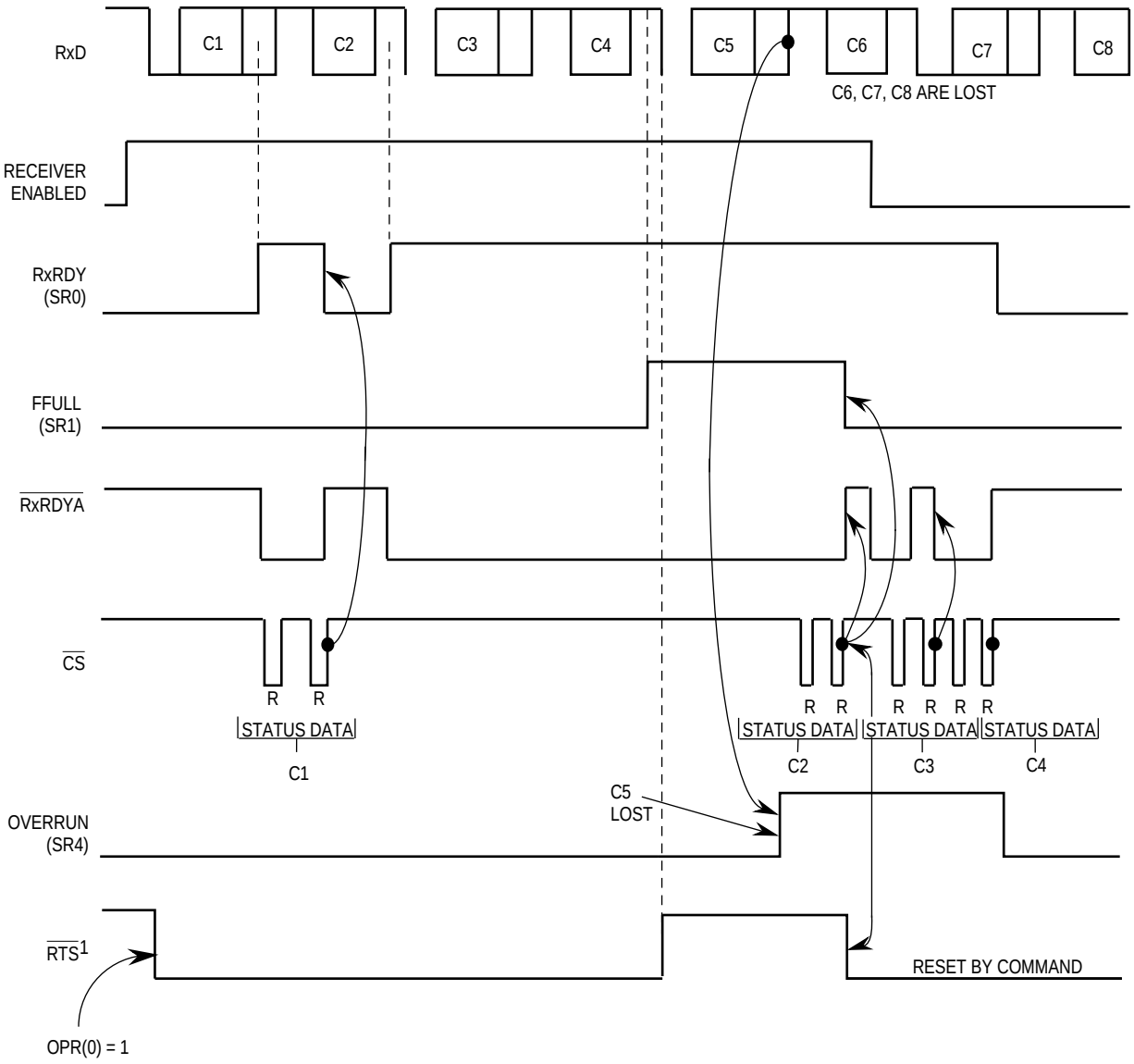
Following transmission of the stop bits, if a new character is not available in the transmitter holding register, the TxDx output remains high ('mark' condition), and the transmitter empty bit (TxEMP) in the SR is set. Transmission resumes and the TxEMP bit is cleared when the CPU32+ loads a new character into the transmitter buffer (TB). If a disable command is sent to the transmitter, it continues operating until the character in the transmit shift register, if any, is completely sent out. If the transmitter is reset through a software command, operation ceases immediately (refer to **8.4.1.3 Command Register (CR)**). The transmitter is re-enabled through the CR to resume operation after a disable or software reset.

If clear-to-send operation is enabled, $\overline{\text{CTSx}}$ must be asserted for the character to be transmitted. If $\overline{\text{CTSx}}$ is negated in the middle of a transmission, the character in the shift register is transmitted, and TxDx remains in the 'mark' state until $\overline{\text{CTSx}}$ is asserted again. If the transmitter is forced to send a continuous low condition by issuing a send break command, the state of $\overline{\text{CTSx}}$ is ignored by the transmitter.

The transmitter can be programmed to automatically negate request-to-send ($\overline{\text{RTSx}}$) outputs upon completion of a message transmission. If the transmitter is programmed to operate in this mode, $\overline{\text{RTSx}}$ must be manually asserted before a message is transmitted. In applications in which the transmitter is disabled after transmission is complete and $\overline{\text{RTSx}}$ is appropriately programmed, $\overline{\text{RTSx}}$ is negated one bit time after the character in the shift register is completely transmitted. The transmitter must be manually re-enabled by reasserting $\overline{\text{RTSx}}$ before the next message is to be sent.

8.3.2.2 RECEIVER. The receivers are enabled through their respective CRs located within the serial module. Functional timing information for the receiver is shown in Figure 8-6. The receiver looks for a high-to-low (mark-to-space) transition of the start bit on RxDx. When a transition is detected, the state of RxDx is sampled each 16× clock for eight clocks, starting one-half clock after the transition (asynchronous operation) or at the next rising edge of the bit time clock (synchronous operation). If RxDx is sampled high, the start bit is invalid, and the search for the valid start bit begins again. If RxDx is still low, a valid start bit is assumed, and the receiver continues to sample the input at one-bit time intervals, at the theoretical center of the bit, until the proper number of data bits and parity, if any, is assembled and one stop bit is detected. Data on the RxDx input is sampled on the rising edge of the programmed clock source. The least significant bit is received first. The data is then transferred to a receiver holding register, and the RxRDY bit in the appropriate SR is set. If the character length is less than eight bits, the most significant unused bits in the receiver holding register are cleared.

After the stop bit is detected, the receiver immediately looks for the next start bit. However, if a nonzero character is received without a stop bit (framing error) and RxDx remains low for one-half of the bit period after the stop bit is sampled, the receiver operates as if a new start bit is detected. The parity error (PE), framing error (FE), overrun error (OE), and received break (RB) conditions (if any) set error and break flags in the appropriate SR at the received character boundary and are valid only when the RxRDY bit in the SR is set.



NOTES:
1. Timing shown for MR1(7) = 1
2. Timing shown for OPCR(4) = 1 and MR1(6) = 0
3. R = Read
4. C_N = Received Character

Figure 8-6. Receiver Timing Diagram

If a break condition is detected (RxDx is low for the entire character including the stop bit), a character of all zeros is loaded into the receiver holding register, and the receive buffer (RB) and RxRDY bits in the SR are set. The RxDx signal must return to a high condition for at least one-half bit time before a search for the next start bit begins.

The receiver detects the beginning of a break in the middle of a character if the break persists through the next character time. When the break begins in the middle of a character, the receiver places the damaged character in the receiver FIFO stack and sets the corresponding error conditions and RxRDY bit in the SR. Then, if the break persists

until the next character time, the receiver places an all-zero character into the receiver FIFO and sets the corresponding RB and RxRDY bits in the SR.

8.3.2.3 FIFO STACK. The FIFO stack is used in each channel's receiver buffer logic. The stack consists of three receiver holding registers. The receive buffer consists of the FIFO and a receiver shift register connected to the RxDx (refer to Figure 8-4). Data is assembled in the receiver shift register and loaded into the top empty receiver holding register position of the FIFO. Thus, data flowing from the receiver to the CPU32+ is quadruple buffered.

In addition to the data byte, three status bits, PE, FE, and RB, are appended to each data character in the FIFO; OE is not appended. By programming the ERR bit in the channel's mode register (MR1), status is provided in character or block modes.

The RxRDY bit in the SR is set whenever one or more characters are available to be read by the CPU32+. A read of the receiver buffer produces an output of data from the top of the FIFO stack. After the read cycle, the data at the top of the FIFO stack and its associated status bits are 'popped', and new data can be added at the bottom of the stack by the receiver shift register. The FIFO-full status bit (FFULL) is set if all three stack positions are filled with data. Either the RxRDY or FFULL bit can be selected to cause an interrupt.

In the character mode, status provided in the SR is given on a character-by-character basis and thus applies only to the character at the top of the FIFO. In the block mode, the status provided in the SR is the logical OR of all characters coming to the top of the FIFO stack since the last reset error command. A continuous logical OR function of the corresponding status bits is produced in the SR as each character reaches the top of the FIFO stack. The block mode is useful in applications where the software overhead of checking each character's error cannot be tolerated. In this mode, entire messages are received, and only one data integrity check is performed at the end of the message. This mode allows a data-reception speed advantage, but does have a disadvantage since each character is not individually checked for error conditions by software. If an error occurs within the message, the error is not recognized until the final check is performed, and no indication exists as to which character in the message is at fault.

In either mode, reading the SR does not affect the FIFO. The FIFO is 'popped' only when the receive buffer is read. The SR should be read prior to reading the receive buffer. If all three of the FIFO's receiver holding registers are full when a new character is received, the new character is held in the receiver shift register until a FIFO position is available. If an additional character is received during this state, the contents of the FIFO are not affected. However, the character previously in the receiver shift register is lost, and the OE bit in the SR is set when the receiver detects the start bit of the new overrunning character.

To support control flow capability, the receiver can be programmed to automatically negate and assert $\overline{\text{RTSx}}$. When in this mode, $\overline{\text{RTSx}}$ is automatically negated by the receiver when a valid start bit is detected and the FIFO stack is full. When a FIFO position

becomes available, $\overline{RTSx}$ is asserted by the receiver. Using this mode of operation, overrun errors are prevented by connecting the $\overline{RTSx}$ to the $\overline{CTSx}$ input of the transmitting device.

If the FIFO stack contains characters and the receiver is disabled, the characters in the FIFO can still be read by the CPU32+. If the receiver is reset, the FIFO stack and all receiver status bits, corresponding output ports, and interrupt request are reset. No additional characters are received until the receiver is re-enabled.

8.3.3 Looping Modes

Each serial module channel can be configured to operate in various looping modes as shown in Figure 8-7. These modes are useful for local and remote system diagnostic functions. The modes are described in the following paragraphs with further information available in **8.4 Register Description and Programming**.

The channel's transmitter and receiver should both be disabled when switching between modes. The selected mode is activated immediately upon mode selection, regardless of whether a character is being received or transmitted.

8.3.3.1 AUTOMATIC ECHO MODE. In this mode, the channel automatically retransmits the received data on a bit-by-bit basis. The local CPU32+-to-receiver communication continues normally, but the CPU32+-to-transmitter link is disabled. While in this mode, received data is clocked on the receiver clock and retransmitted on TxDx. The receiver must be enabled, but the transmitter need not be enabled.

Since the transmitter is not active, the SR TxEMP and TxRDY bits are inactive, and data is transmitted as it is received. Received parity is checked, but not recalculated for transmission. Character framing is also checked, but stop bits are transmitted as received. A received break is echoed as received until the next valid start bit is detected.

8.3.3.2 LOCAL LOOPBACK MODE. In this mode, TxDx is internally connected to RxDx. This mode is useful for testing the operation of a local serial module channel by sending data to the transmitter and checking data assembled by the receiver. In this manner, correct channel operations can be assured. Also, both transmitter and CPU32+-to-receiver communications continue normally in this mode. While in this mode, the RxDx input data is ignored, the TxDx is held marking, and the receiver is clocked by the transmitter clock. The transmitter must be enabled, but the receiver need not be enabled.

8.3.3.3 REMOTE LOOPBACK MODE. In this mode, the channel automatically transmits received data on the TxDx output on a bit-by-bit basis. The local CPU32+-to-transmitter link is disabled. This mode is useful in testing receiver and transmitter operation of a remote channel. While in this mode, the receiver clock is used for the transmitter.

Since the receiver is not active, received data cannot be read by the CPU32+, and the error status conditions are inactive. Received parity is not checked and is not recalculated for transmission. Stop bits are transmitted as received. A received break is echoed as received until the next valid start bit is detected.

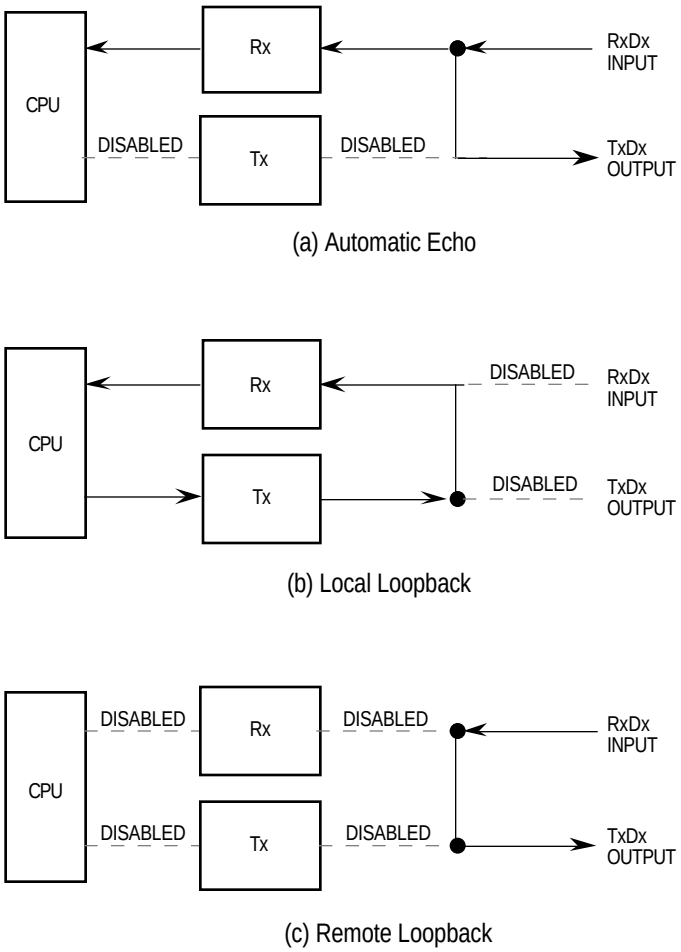


Figure 8-7. Looping Modes Functional Diagram

8.3.4 Multidrop Mode

A channel can be programmed to operate in a wakeup mode for multidrop or multiprocessor applications. Functional timing information for the multidrop mode is shown in Figure 8-8. The mode is selected by setting bits 3 and 4 in mode register 1 (MR1). This mode of operation allows the master station to be connected to several slave stations (maximum of 256). In this mode, the master transmits an address character followed by a block of data characters targeted for one of the slave stations. The slave stations have their channel receivers disabled. However, they continuously monitor the data stream sent out by the master station. When an address character is sent by the master, the slave receiver channel notifies its respective CPU by setting the RxRDY bit in the SR and generating an interrupt (if programmed to do so). Each slave station CPU then compares the received address to its station address and enables its receiver if it wishes to receive the subsequent data characters or block of data from the master station. Slave stations not addressed continue to monitor the data stream for the next address character. Data fields in the data stream are separated by an address character. After a slave receives a block of data, the slave station's CPU disables the receiver and initiates the process again.

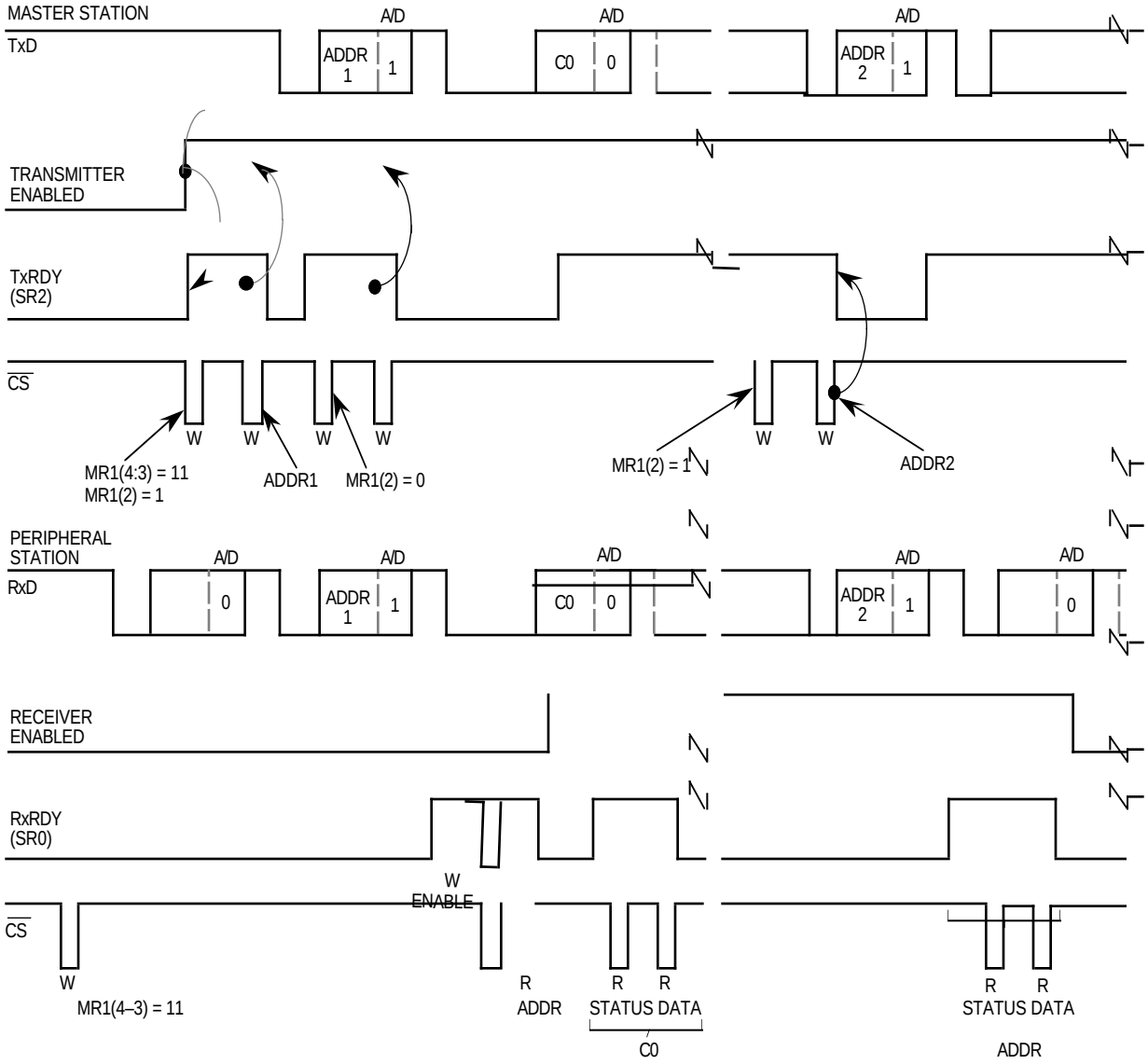


Figure 8-8. Multidrop Mode Timing Diagram

A transmitted character from the master station consists of a start bit, a programmed number of data bits, an address/data (A/D) bit flag, and a programmed number of stop bits. The A/D bit identifies the type of character being transmitted to the slave station. The character is interpreted as an address character if the A/D bit is set or as a data character if the A/D bit is cleared. The polarity of the A/D bit is selected by programming bit 2 of the MR1. The MR1 should be programmed before enabling the transmitter and loading the corresponding data bits into the transmit buffer.

In multidrop mode, the receiver continuously monitors the received data stream, regardless of whether it is enabled or disabled. If the receiver is disabled, it sets the RxRDY bit and loads the character into the receiver holding register FIFO stack provided the received A/D bit is a one (address tag). The character is discarded if the received A/D bit is a zero (data tag). If the receiver is enabled, all received characters are transferred to the CPU32+ via the receiver holding register stack during read operations.

In either case, the data bits are loaded into the data portion of the stack while the A/D bit is loaded into the status portion of the stack normally used for a parity error (SR bit 5). Framing error, overrun error, and break detection operate normally. The A/D bit takes the place of the parity bit; therefore, parity is neither calculated nor checked. Messages in this mode may still contain error detection and correction information. One way to provide error detection, if 8-bit characters are not required, is to use software to calculate parity and append it to the 5-, 6-, or 7-bit character.

8.3.5 Bus Operation

This section describes the operation of the IMB during read, write, and interrupt acknowledge cycles to the serial module. All serial module registers must be accessed as bytes.

8.3.5.1 READ CYCLES. The serial module is accessed by the CPU32+ with no wait states. The serial module responds to byte reads. Reserved registers return logic zero during reads.

8.3.5.2 WRITE CYCLES. The serial module is accessed by the CPU32+ with no wait states. The serial module responds to byte writes. Write cycles to read-only registers and reserved registers complete in a normal manner without exception processing; however, the data is ignored.

8.3.5.3 INTERRUPT ACKNOWLEDGE CYCLES. The serial module is capable of arbitrating for interrupt servicing and supplying the interrupt vector when it has successfully won arbitration. The vector number must be provided if interrupt servicing is necessary; thus, the interrupt vector register (IVR) must be initialized. If the IVR is not initialized, a spurious interrupt exception will be taken if interrupts are generated.

8.4 REGISTER DESCRIPTION AND PROGRAMMING

This section contains a detailed description of each register and its specific function as well as flowcharts of basic serial module programming.

8.4.1 Register Description

The operation of the serial module is controlled by writing control bytes into the appropriate registers. A list of serial module registers and their associated addresses are shown in Figure 8-9. The mode, status, command, and clock-select registers are duplicated for each channel to provide independent operation and control.

NOTE

All serial module registers are only accessible as bytes. The contents of the mode registers (MR1 and MR2), CSR, and the auxiliary control register (ACR) bit 7 should only be changed after the receiver/transmitter is issued a software RESET command—i.e., channel operation must be disabled. Care should also be taken if the register contents are changed during receiver/transmitter operations, as undesirable results may be produced.

The registers of the serial module are discussed in the following paragraphs in alphabetical order. The numbers in the upper right-hand corner indicate the offset of the register from the base address specified in the module base address register (MBAR) in the SIM49. The numbers above the register description represent the bit position in the register. The register description contains the mnemonic for the bit. The values shown below the register description are the values of those register bits after a hardware reset. A value of U indicates that the bit value is unaffected by reset. The read/write status and the access privilege are shown in the last line.

NOTE

A CPU32+ RESET instruction will not affect the MCR, but will reset all the other serial module registers as though a hardware reset had occurred. The module is enabled when the STP bit in the MCR is cleared. The module is disabled when the STP bit in the MCR is set.

Address	FC	Register Read (R/W = 1)	Register Write (R/W = 0)
700	S[1]	MCR (HIGH BYTE)	MCR (HIGH BYTE)
701	S	MCR (LOW BYTE)	MCR (LOW BYTE)
702	S	DO NOT ACCESS[3]	DO NOT ACCESS[3]
703	S	DO NOT ACCESS[3]	DO NOT ACCESS[3]
704	S	INTERRUPT LEVEL (ILR)	INTERRUPT LEVEL (ILR)
705	S	INTERRUPT VECTOR (IVR)	INTERRUPT VECTOR (IVR)
710	S/U[2]	MODE REGISTER 1A (MR1A)	MODE REGISTER 1A (MR1A)
711	S/U	STATUS REGISTER A (SRA)	CLOCK-SELECT REGISTER A (CSRA)
712	S/U	DO NOT ACCESS[3]	COMMAND REGISTER A (CRA)
713	S/U	RECEIVER BUFFER A (RBA)	TRANSMITTER BUFFER A (TBA)
714	S/U	INPUT PORT CHANGE REGISTER (IPCR)	AUXILIARY CONTROL REGISTER (ACR)
715	S/U	INTERRUPT STATUS REGISTER (ISR)	INTERRUPT ENABLE REGISTER (IER)
716	S/U	DO NOT ACCESS[3]	DO NOT ACCESS[3]
717	S/U	DO NOT ACCESS[3]	DO NOT ACCESS[3]
718	S/U	MODE REGISTER 1B (MR1B)	MODE REGISTER 1B (MR1B)
719	S/U	STATUS REGISTER B (SRB)	CLOCK-SELECT REGISTER B (CSRB)
71A	S/U	DO NOT ACCESS[3]	COMMAND REGISTER B (CRB)
71B	S/U	RECEIVER BUFFER B (RBB)	TRANSMITTER BUFFER B (TBB)
71C	S/U	DO NOT ACCESS[3]	DO NOT ACCESS[3]
71D	S/U	INPUT PORT REGISTER (IP)	OUTPUT PORT CONTROL REGISTER (OPCR)
71E	S/U	DO NOT ACCESS[3]	OUTPUT PORT (OP)[4] BIT SET
71F	S/U	DO NOT ACCESS[3]	OUTPUT PORT (OP)[4] BIT RESET
720	S/U	MODE REGISTER 2A (MR2A)	MODE REGISTER 2A (MR2A)
721	S/U	MODE REGISTER 2B (MR2B)	MODE REGISTER 2B (MR2B)

NOTES:
1. S = Register permanently defined as supervisor-only access
2. S/U = Register programmable as either supervisor or user access
3. A read or write to these locations currently has no effect.
4. Address-triggered commands

Figure 8-9. Serial Module Programming Model

8.4.1.1 AUXILIARY CONTROL REGISTER (ACR). The ACR selects which baud rate is used and controls the handshake of the transmitter/receiver. This register can only be written when the serial module is enabled (i.e., the STP bit in the MCR is cleared).

ACR $714

7	6	5	4	3	2	1	0
BRG	0	0	0	0	0	IECB	IECA
RESET: 0	0	0	0	0	0	0	0

Write Only Supervisor/User

BRG—Baud Rate Generator Set Select

1 = Set 2 of the available baud rates is selected.
0 = Set 1 of the available baud rates is selected. Refer to **8.4.1.2 Clock-Select Register (CSR)** for more information on the baud rates.

IECB, IECA—Input Enable Control

1 = ISR bit 7 will be set and an interrupt will be generated when the corresponding bit in the IPCR (COSB or COSA) is set by an external transition on the channel's $\overline{\text{CTSx}}$ input (if bit 7 of the IER) is set to enable interrupts).
0 = Setting the corresponding bit in the IPCR has no effect on ISR bit 7.

8.4.1.2 CLOCK-SELECT REGISTER (CSR). The CSR selects the baud rate clock for the channel receiver and transmitter. This register can only be written when the serial module is enabled (i.e., the STP bit in the MCR is cleared).

NOTE

This register should only be written after the external crystal is stable (XTAL_RDY bit of the ISR is zero).

CSRA, CSRB $711, $719

7	6	5	4	3	2	1	0
RCS3	RCS2	RCS1	RCS0	TCS3	TCS2	TCS1	TCS0
RESET: 0	0	0	0	0	0	0	0

Write Only Supervisor/User

RCS3–RCS0—Receiver Clock Select

These bits select the baud rate clock for the channel receiver from a set of baud rates listed in Table 8-1. The baud rate set selected depends upon the ACR bit 7. Set 1 is selected if ACR bit 7 = 0, and set 2 is selected if ACR bit 7 = 1. The receiver clock is always 16 times the baud rate shown in this list, except when SCLK is used.

Table 8-1. RCSx Control Bits

RCS3	RCS2	RCS1	RCS0	Set 1	Set 2
0	0	0	0	50	75
0	0	0	1	110	110
0	0	1	0	134.5	134.5
0	0	1	1	200	150
0	1	0	0	300	300
0	1	0	1	600	600
0	1	1	0	1200	1200
0	1	1	1	1050	2000
1	0	0	0	2400	2400
1	0	0	1	4800	4800
1	0	1	0	7200	1800
1	0	1	1	9600	9600
1	1	0	0	38.4k	19.2k
1	1	0	1	76.8k	38.4k
1	1	1	0	SCLK/16	SCLK/16
1	1	1	1	SCLK/1	SCLK/1

TCS3–TCS0—Transmitter Clock Select

These bits select the baud rate clock for the channel transmitter from a set of baud rates listed in Table 8-2. The baud rate set selected depends upon ACR bit 7. Set 1 is selected if ACR bit 7 = 0, and set 2 is selected if ACR bit 7 = 1. The transmitter clock is always 16 times the baud rate shown in this list, except when SCLK is used.

Table 8-2 . TCSx Control Bits

TCS3	TCS2	TCS1	TCS0	Set 1	Set 2
0	0	0	0	50	75
0	0	0	1	110	110
0	0	1	0	134.5	134.5
0	0	1	1	200	150
0	1	0	0	300	300
0	1	0	1	600	600
0	1	1	0	1200	1200
0	1	1	1	1050	2000
1	0	0	0	2400	2400
1	0	0	1	4800	4800
1	0	1	0	7200	1800
1	0	1	1	9600	9600
1	1	0	0	38.4k	19.2k
1	1	0	1	76.8k	38.4k
1	1	1	0	SCLK/16	SCLK/16
1	1	1	1	SCLK/1	SCLK/1

8.4.1.3 COMMAND REGISTER (CR). The CR is used to supply commands to the channel. Multiple commands can be specified in a single write to the CR if the commands are not conflicting—e.g., reset transmitter and enable transmitter commands cannot be specified in a single command. This register can only be written when the serial module is enabled (i.e., the STP bit in the MCR is cleared).

CRA, CRB $712, $71A

7	6	5	4	3	2	1	0
MISC3	MISC2	MISC1	MISC0	TC1	TC0	RC1	RC0
RESET:							
0	0	0	0	0	0	0	0

Write Only Supervisor/User

MISC3–MISC0—Miscellaneous Commands

These bits select a single command as listed in Table 8-3.

Table 8-3. MISCx Control Bits

MISC3	MISC2	MISC1	MISC0	Command
0	0	0	0	No Command
0	0	0	1	No Command
0	0	1	0	Reset Receiver
0	0	1	1	Reset Transmitter
0	1	0	0	Reset Error Status
0	1	0	1	Reset Break-Change Interrupt
0	1	1	0	Start Break
0	1	1	1	Stop Break
1	0	0	0	Assert $\overline{RTS}$
1	0	0	1	Negate $\overline{RTS}$
1	0	1	0	No Command
1	0	1	1	No Command
1	1	0	0	No Command
1	1	0	1	No Command
1	1	1	0	No Command
1	1	1	1	No Command

Reset Receiver—The reset receiver command resets the channel receiver. The receiver is immediately disabled, the FFULL and RxRDY bits in the SR are cleared, and the receiver FIFO pointer is reinitialized. All other registers are unaltered. This command should be used in lieu of the receiver disable command whenever the receiver configuration is changed because it places the receiver in a known state.

Reset Transmitter—The reset transmitter command resets the channel transmitter. The transmitter is immediately disabled, and the TxEMP and TxRDY bits in the SR are cleared. All other registers are unaltered. This command should be used in lieu of the transmitter disable command whenever the transmitter configuration is changed because it places the transmitter in a known state.

Reset Error Status—The reset error status command clears the channel's RB, FE, PE, and OE bits (in the SR). This command is also used in the block mode to clear all error bits after a data block is received.

Reset Break-Change Interrupt—The reset break-change interrupt command clears the delta break (DBx) bits in the ISR.

Start Break—The start break command forces the channel's TxDx low. If the transmitter is empty, the start of the break conditions can be delayed up to one bit time. If the transmitter is active, the break begins when transmission of the character is complete. If a character is in the transmitter shift register, the start of the break is delayed until the character is transmitted. If the transmitter holding register has a character, that character is transmitted after the break. The transmitter must be enabled for this command to be accepted. The state of the $\overline{CTSx}$ input is ignored for this command.

Stop Break—The stop break command causes the channel's TxDx to go high (mark) within two bit times. Characters stored in the transmitter buffer, if any, are transmitted.

Assert $\overline{RTS}$—The assert $\overline{RTS}$ command forces the channel's $\overline{RTSx}$ output low.

Negate $\overline{RTS}$—The negate $\overline{RTS}$ command forces the channel's $\overline{RTSx}$ output high.

TC1–TC0—Transmitter Commands

These bits select a single command as listed in Table 8-4.

Table 8-4. TCx Control Bits

TC1	TC0	Command
0	0	No Action Taken
0	1	Enable Transmitter
1	0	Disable Transmitter
1	1	Do Not Use

No Action Taken—The no action taken command causes the transmitter to stay in its current mode. If the transmitter is enabled, it remains enabled; if disabled, it remains disabled.

Transmitter Enable—The transmitter enable command enables operation of the channel's transmitter. The TxEMP and TxRDY bits in the SR are also set. If the transmitter is already enabled, this command has no effect.

Transmitter Disable—The transmitter disable command terminates transmitter operation and clears the TxEMP and TxRDY bits in the SR. However, if a character is being transmitted when the transmitter is disabled, the transmission of the character is completed before the transmitter becomes inactive. If the transmitter is already disabled, this command has no effect.

Do Not Use—Do not use this bit combination because the result is indeterminate.

RC1–RC0—Receiver Commands

These bits select a single command as listed in Table 8-5.

Table 8-5. RCx Control Bits

RC1	RC0	Command
0	0	No Action Taken
0	1	Enable Receiver
1	0	Disable Receiver
1	1	Do Not Use

No Action Taken—The no action taken command causes the receiver to stay in its current mode. If the receiver is enabled, it remains enabled; if disabled, it remains disabled.

Receiver Enable—The receiver enable command enables operation of the channel's receiver. If the serial module is not in multidrop mode, this command also forces the receiver into the search-for-start-bit state. If the receiver is already enabled, this command has no effect.

Receiver Disable—The receiver disable command disables the receiver immediately. Any character being received is lost. The command has no effect on the receiver status bits or any other control register. If the serial module is programmed to operate in the local loopback mode or multidrop mode, the receiver operates even though this command is selected. If the receiver is already disabled, this command has no effect.

Do Not Use—Do not use this bit combination because the result is indeterminate.

8.4.1.4 INPUT PORT CHANGE REGISTER (IPCR). The IPCR shows the current state and the change-of-state for the $\overline{\text{CTSA}}$ and $\overline{\text{CTSB}}$ pins. This register can only be read when the serial module is enabled (i.e., the STP bit in the MCR is cleared).

IPCR $714

7	6	5	4	3	2	1	0
0	0	COSB	COSA	0	0	CTSB	CTSA
RESET: 0	0	0	0	0	0	U	U

Read Only Supervisor/User

Bits 7, 6, 3, 2—Reserved by Motorola

COSB, COSA—Change-of-State

1 = A change-of-state (high-to-low or low-to-high transition), lasting longer than 25–50 μs when using a crystal as the sampling clock or longer than one or two periods when using SCLK, has occurred at the corresponding $\overline{\text{CTSx}}$ input (MCR ICCS bit controls selection of the sampling clock for clear-to-send operation). When these bits are set, the ACR can be programmed to generate an interrupt to the CPU32+.

0 = The CPU32+ has read the IPCR. No change-of-state has occurred. A read of the IPCR also clears the ISR COS bit.

CTSB, CTSA—Current State

Starting two serial clock periods after reset, the $\overline{\text{CTSx}}$ bits reflect the state of the $\overline{\text{CTSx}}$ pins. If a $\overline{\text{CTSx}}$ pin is detected as asserted at that time, the associated COSx bit will be set, which will initiate an interrupt if the corresponding IECx bit of the ACR register is enabled.

1 = The current state of the respective $\overline{\text{CTSx}}$ input is negated.
0 = The current state of the respective $\overline{\text{CTSx}}$ input is asserted.

8.4.1.5 INPUT PORT REGISTER (IP). The IP shows the current state of the $\overline{\text{CTSx}}$ inputs. This register can only be read when the serial module is enabled (i.e., the STP bit in the MCR is cleared).

IP $71D

7	6	5	4	3	2	1	0
0	0	0	0	0	0	CTSB	CTSA
RESET: 0	0	0	0	0	0	U	U

Read Only Supervisor/User

CTSB, CTSA—Current State

1 = The current state of the respective $\overline{\text{CTSx}}$ input is negated.
0 = The current state of the respective $\overline{\text{CTSx}}$ input is asserted.

The information contained in these bits is latched and reflects the state of the input pins at the time that the IP is read.

NOTE

These bits have the same function and value of the IPCR bits 1 and 0.

8.4.1.6 INTERRUPT ENABLE REGISTER (IER). The IER selects the corresponding bits in the ISR that cause an interrupt output ($\overline{IRQx}$). If one of the bits in the ISR is set and the corresponding bit in the IER is also set, the $\overline{IRQx}$ output is asserted. If the corresponding bit in the IER is zero, the state of the bit in the ISR has no effect on the $\overline{IRQx}$ output. The IER does not mask the reading of the ISR. The ISR XTAL_RDY bit cannot be enabled to generate an interrupt. This register can only be written when the serial module is enabled (i.e., the STP bit in the MCR is cleared).

IER $715

7	6	5	4	3	2	1	0
COS	DBB	RxRDYB	TxRDYB	0	DBA	RxRDYA	TxRDYA
RESET: 0	0	0	0	0	0	0	0	0

Write Only Supervisor/User

COS—Change-of-State

1 = Enable interrupt
0 = Disable interrupt

DBB—Delta Break B

1 = Enable interrupt
0 = Disable interrupt

RxRDYB—Channel B Receiver Ready or FIFO full

1 = Enable interrupt
0 = Disable interrupt

TxRDYB—Channel B Transmitter Ready

1 = Enable interrupt
0 = Disable interrupt

Bit 3—Reserved by Motorola

DBA—Delta Break A

1 = Enable interrupt
0 = Disable interrupt

RxRDYA—Channel A Receiver Ready or FIFO full

1 = Enable interrupt
0 = Disable interrupt

TxRDYA—Channel A Transmitter Ready

1 = Enable interrupt
0 = Disable interrupt

8.4.1.7 INTERRUPT LEVEL REGISTER (ILR). The ILR contains the priority level for the serial module interrupt request. When the serial module is enabled (i.e., the STP bit in the MCR is cleared), this register can be read or written to at any time while in supervisor mode.

ILR $704

7	6	5	4	3	2	1	0
0	0	0	0	0	IL2	IL1	IL0
RESET:							
0	0	0	0	0	0	0	0

Read/Write Supervisor Only

Bits 7–3—Reserved by Motorola

IL2–IL0—Interrupt Level Bits

Each module that can generate interrupts has an interrupt level field. The priority level encoded in these bits is sent to the CPU32+ on the appropriate $\overline{\text{IRQx}}$ signal. The CPU32+ uses this value to determine servicing priority. The hardware reset value of $00 will not generate any interrupts. See **Section 5 CPU030** for more information.

8.4.1.8 INTERRUPT STATUS REGISTER (ISR). The ISR provides status for all potential interrupt sources. The contents of this register are masked by the IER. If a flag in the ISR is set and the corresponding bit in IER is also set, the $\overline{\text{IRQx}}$ output is asserted. If the corresponding bit in the IER is cleared, the state of the bit in the ISR has no effect on the output. This register can only be read when the serial module is enabled (i.e., the STP bit in the MCR is cleared).

NOTE

The IER does not mask reading of the ISR. True status is provided regardless of the contents of IER. The contents of ISR are cleared when the serial module is reset.

ISR $715

7	6	5	4	3	2	1	0
COS	DBB	RxRDYB	TxRDYB	XTAL_ RDY	DBA	RxRDYA	TxRDYA
RESET:							
0	0	0	0	1	0	0	0

Read Only Supervisor/User

COS—Change-of-State

1 = A change-of-state has occurred at one of the $\overline{\text{CTSx}}$ inputs and has been selected to cause an interrupt by programming bit 1 and/or bit 0 of the ACR.
0 = The CPU32+ has read the IPCR.

DBB—Delta Break B

1 = The channel B receiver has detected the beginning or end of a received break.
0 = The CPU32+ has issued a channel B reset break-change interrupt command. Refer to **8.4.1.3 Command Register (CR)** for more information on the reset break-change interrupt command.

RxRDYB—Channel B Receiver Ready or FIFO Full

The function of this bit is programmed by MR1B bit 6.

1 = If programmed as receiver ready, a character has been received in channel B and is waiting in the receiver buffer FIFO. If programmed as FIFO full, a character has been transferred from the receiver shift register to the FIFO, and the transfer has caused the channel B FIFO to become full (all three positions are occupied).
0 = If programmed as receiver ready, the CPU32+ has read the RB. After this read, if more characters are still in the FIFO, the bit is set again after the FIFO is 'popped'. If programmed as FIFO full, the CPU32+ has read the RB. If a character is waiting in the receiver shift register because the FIFO is full, the bit will be set again when the waiting character is loaded into the FIFO.

TxRDYB—Channel B Transmitter Ready

This bit is the duplication of the TxRDY bit in SRB.

1 = The transmitter holding register is empty and ready to be loaded with a character. This bit is set when the character is transferred to the transmitter shift register. This bit is also set when the transmitter is first enabled. Characters loaded into the transmitter holding register while the transmitter is disabled are not transmitted.
0 = The transmitter holding register was loaded by the CPU32+, or the transmitter is disabled.

XTAL_RDY—Serial Clock Running

This bit is always read as a zero when the X1 clock is running. This bit cannot be enabled to generate an interrupt.

1 = This bit is set at reset.
0 = This bit is cleared after the baud rate generator is stable. The CSR should not be accessed until this bit is zero.

DBA—Delta Break A

1 = The channel A receiver has detected the beginning or end of a received break.
0 = The CPU32+ has issued a channel A reset break-change interrupt command. Refer to **8.4.1.3 Command Register (CR)** for more information on the reset break-change interrupt command.

RxRDYA—Channel A Receiver Ready or FIFO Full

The function of this bit is programmed by MR1A bit 6.

1 = If programmed as receiver ready, a character has been received in channel A and is waiting in the receiver buffer FIFO. If programmed as FIFO full, a character has been transferred from the receiver shift register to the FIFO, and the transfer has caused the channel A FIFO to become full (all three positions are occupied).

0 = If programmed as receiver ready, the CPU32+ has read the receiver buffer. After this read, if more characters are still in the FIFO, the bit is set again after the FIFO is 'popped'. If programmed as FIFO full, the CPU32+ has read the receiver buffer. If a character is waiting in the receiver shift register because the FIFO is full, the bit will be set again when the waiting character is loaded into the FIFO.

TxRDYA—Channel A Transmitter Ready

This bit is the duplication of the TxRDY bit in SRA.

1 = The transmitter holding register is empty and ready to be loaded with a character. This bit is set when the character is transferred to the transmitter shift register. This bit is also set when the transmitter is first enabled. Characters loaded into the transmitter holding register while the transmitter is disabled are not transmitted.

0 = The transmitter holding register was loaded by the CPU32+, or the transmitter is disabled.

8.4.1.9 INTERRUPT VECTOR REGISTER (IVR). The IVR contains the 8-bit vector number of the interrupt. When the serial module is enabled (i.e., the STP bit in the MCR is cleared), this register can be read or written to at any time while in supervisor mode.

IVR $705

7	6	5	4	3	2	1	0
IVR7	IVR6	IVR5	IVR4	IVR3	IVR2	IVR1	IVR0
RESET:							
0	0	0	0	1	1	1	1

Read /Write Supervisor Only

IVR7–IVR0—Interrupt Vector Bits

Each module that generates interrupts has an interrupt vector field. This 8-bit number indicates the offset from the base of the vector table where the address of the exception handler for the specified interrupt is located. The IVR is reset to $0F, which indicates an uninitialized interrupt condition. See **Section 5 CPU030** for more information.

8.4.1.10 MODULE CONFIGURATION REGISTER (MCR). The MCR controls the serial module configuration. This register can be either read or written when the module is enabled and is in the supervisor state. The MCR is not affected by a CPU32+ RESET instruction. Only the MCR can be accessed when the module is disabled (i.e., the STP bit in the MCR is set).

MCR $700

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
STP	FRZ1	FRZ0	ICCS	0	0	0	0	SUPV	0	0	0	IARB			
RESET:															
0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0

Read/Write Supervisor Only

STP—Stop Mode Bit

1 = The serial module will be disabled. Setting the STP bit stops all clocks within the serial module (including the crystal or external clock and SCLK), except for the clock from the IMB. The clock from the IMB remains active to allow CPU32+ access to the MCR. The clock stops on the low phase of the clock and remains stopped until the STP bit is cleared by the CPU32+ or a hardware reset. Accesses to serial module registers while in stop mode produce a bus error. The serial module should be disabled in a known state prior to setting the STP bit; otherwise, unpredictable results may occur. The STP bit should be set prior to executing the LPSTOP instruction to reduce overall power consumption.

0 = The serial module is enabled and will operate in normal mode. When STP = 0, make sure the external crystal is stable (XTAL_RDY bit (bit 3) of the ISR is zero) before continuing.

NOTE

The serial module should be disabled (i.e., the STP bit in the MCR is set) before executing the LPSTOP instruction to obtain the lowest power consumption. The X1/X2 oscillator will continue to run during LPSTOP if STP = 0.

FRZ1–FRZ0—Freeze

These bits determine the action taken when the FREEZE signal is asserted on the IMB when the CPU32+ has entered background debug mode. Table 8-6 lists the action taken for each combination of bits.

Table 8-6. FRZx Control Bits

FRZ1	FRZ0	Action
0	0	Ignore FREEZE
0	1	Reserved (FREEZE Ignored)
1	0	Freeze on Character Boundary
1	1	Freeze on Character Boundary

If FREEZE is asserted, channel A and channel B freeze independently of each other. The transmitter and receiver freeze at character boundaries. The transmitter does not freeze in the send break mode. Communications can be lost if the channel is not programmed to support flow control. See **Section 5 CPU030** for more information on FREEZE.

ICCS—Input Capture Clock Select

1 = Selects SCLK as the clear-to-send input capture clock for both channels. Clear-to-send operation is enabled by setting bit 4 in MR2. The data is captured on the $\overline{\text{CTSx}}$ pins on the rising edge of the clock.

0 = The crystal clock is the clear-to-send input capture clock for both channels.

Bits 11–8, 6–4—Reserved by Motorola

SUPV—Supervisor/User

The value of this bit has no affect on registers permanently defined as supervisor only.

1 = The serial module registers, which are defined as supervisor or user, reside in supervisor data space and are only accessible from supervisor programs.

0 = The serial module registers, which are defined as supervisor or user, reside in user data space and are accessible from either supervisor or user programs.

IARB3–IARB0—Interrupt Arbitration Bits

Each module that generates interrupts has an IARB field. These bits are used to arbitrate for the bus in the case that two or more modules simultaneously generate an interrupt at the same priority level. No two modules can share the same IARB value. The reset value of the IARB field is $0, which prevents this module from arbitrating during the interrupt acknowledge cycle. The system software should initialize the IARB field to a value from $F (highest priority) to $1 (lowest priority).

8.4.1.11 MODE REGISTER 1 (MR1). MR1 controls some of the serial module configuration. This register can be read or written at any time when the serial module is enabled (i.e., the STP bit in the MCR is cleared).

MR1A, MR1B $710, $718

7	6	5	4	3	2	1	0
RxRTS	R/F	ERR	PM1	PM0	PT	B/C1	B/C0
RESET:							
0	0	0	0	0	0	0	0

Read/Write Supervisor/User

RxRTS—Receiver Request-to-Send Control

1 = Upon receipt of a valid start bit, $\overline{\text{RTSx}}$ is negated if the channel's FIFO is full. $\overline{\text{RTSx}}$ is reasserted when the FIFO has an empty position available.
0 = $\overline{\text{RTSx}}$ is asserted by setting bit 1 or 0 in the OP and negated by clearing bit 1 or 0 in the OP.

This feature can be used for flow control to prevent overrun in the receiver by using the $\overline{\text{RTSx}}$ output to control the $\overline{\text{CTSx}}$ input of the transmitting device. If both the receiver and transmitter are programmed for $\overline{\text{RTS}}$ control, $\overline{\text{RTS}}$ control will be disabled for both since this configuration is incorrect. See **8.4.1.12 Mode Register 2** for information on programming the transmitter $\overline{\text{RTSx}}$ control.

R/F—Receiver-Ready Select

1 = Bit 5 for channel B and bit 1 for channel A in the ISR reflect the channel FIFO full status. These ISR bits are set when the receiver FIFO is full and are cleared when a position is available in the FIFO.
0 = Bit 5 for channel B and bit 1 for channel A in the ISR reflect the channel receiver-ready status. These ISR bits are set when a character has been received and are cleared when the CPU32+ reads the receive buffer.

ERR—Error Mode

This bit controls the meaning of the three FIFO status bits (RB, FE, and PE) in the SR for the channel.

1 = Block mode—The values in the channel SR are the accumulation (i.e., the logical OR) of the status for all characters coming to the top of the FIFO since the last reset error status command for the channel was issued. Refer to **8.4.1.3 Command Register (CR)** for more information on serial module commands.
0 = Character mode—The values in the channel SR reflect the status of the character at the top of the FIFO.

NOTE

ERR = 0 must be used to get the correct A/D flag information when in multidrop mode.

PM1–PM0—Parity Mode

These bits encode the type of parity used for the channel (see Table 8-2). The parity bit is added to the transmitted character, and the receiver performs a parity check on incoming data. These bits can alternatively select multidrop mode for the channel.

PT—Parity Type

This bit selects the parity type if parity is programmed by the parity mode bits, and if multidrop mode is selected, it configures the transmitter for data character transmission or address character transmission. Table 8-7 lists the parity mode and type or the multidrop mode for each combination of the parity mode and the parity type bits.

Table 8-7. PMx and PT Control Bits

PM1	PM0	Parity Mode	PT	Parity Type
0	0	With Parity	0	Even Parity
0	0	With Parity	1	Odd Parity
0	1	Force Parity	0	Low Parity
0	1	Force Parity	1	High Parity
1	0	No Parity	X	No Parity
1	1	Multidrop Mode	0	Data Character
1	1	Multidrop Mode	1	Address Character

B/C1–B/C0—Bits per Character

These bits select the number of data bits per character to be transmitted. The character length listed in Table 8-8 does not include start, parity, or stop bits.

Table 8-8. B/Cx Control Bits

B/C1	B/C0	Bits/Character
0	0	Five Bits
0	1	Six Bits
1	0	Seven Bits
1	1	Eight Bits

8.4.1.12 MODE REGISTER 2 (MR2). MR2 controls some of the serial module configuration. This register can be read or written at any time the serial module is enabled (i.e., the STP bit in the MCR is cleared).

MR2A, MR2B $720, $721

7	6	5	4	3	2	1	0
CM1	CM0	TxRTS	TxCTS	SB3	SB2	SB1	SB0
RESET:							
0	0	0	0	0	0	0	0

Read/Write Supervisor/User

CM1, CM0—Channel Mode

These bits select a channel mode as listed in Table 8-9. See **8.3.3 Looping Modes** for more information on the individual modes.

Table 8-9. CMx Control Bits

CM1	CM0	Mode
0	0	Normal
0	1	Automatic Echo
1	0	Local Loopback
1	1	Remote Loopback

TxRTS—Transmitter Ready-to-Send

This bit controls the negation of the $\overline{\text{RTSA}}$ or $\overline{\text{RTSB}}$ signals. The output is normally asserted by setting OP0 or OP1 in the OPCR and negated by clearing OP0 or OP1 in the OPCR (see **8.4.1.14 Output Port Control Register (OPCR)**).

1 = In applications where the transmitter is disabled after transmission is complete, setting this bit causes the particular OP bit to be cleared automatically one bit time after the characters, if any, in the channel transmit shift register and the transmitter holding register are completely transmitted, including the programmed number of stop bits. This feature is used to automatically terminate transmission of a message. If both the receiver and the transmitter in the same channel are programmed for $\overline{\text{RTSx}}$ control, $\overline{\text{RTSx}}$ control is disabled for both since this is an incorrect configuration.

0 = Clearing this bit has no effect on the transmitter $\overline{\text{RTSx}}$.

TxCTS—Transmitter Clear-to-Send

1 = Enables clear-to-send operation. The transmitter checks the state of the $\overline{\text{CTSx}}$ input each time it is ready to send a character. If $\overline{\text{CTSx}}$ is asserted, the character is transmitted. If $\overline{\text{CTSx}}$ is negated, the channel TxDx remains in the high state, and the transmission is delayed until $\overline{\text{CTSx}}$ is asserted. Changes in $\overline{\text{CTSx}}$ while a character is being transmitted do not affect transmission of that character. If both TxCTS and TxRTS are enabled, TxCTS controls the operation of the transmitter.

0 = The $\overline{\text{CTSx}}$ has no effect on the transmitter.

SB3–SB0—Stop-Bit Length Control

These bits select the length of the stop bit appended to the transmitted character as listed in Table 8-10. Stop-bit lengths of nine-sixteenth to two bits, in increments of one-sixteenth bit, are programmable for character lengths of six, seven, and eight bits. For a character length of five bits, one and one-sixteenth to two bits are programmable in increments of one-sixteenth bit. In all cases, the receiver only checks for a high condition at the center of the first stop-bit position—i.e., one bit time after the last data bit or after the parity bit, if parity is enabled.

If an external 1× clock is used for the transmitter, MR2 bit 3 = 0 selects one stop bit, and MR2 bit 3 = 1 selects two stop bits for transmission.

Table 8-10. SBx Control Bits

SB3	SB2	SB1	SB0	Length 6-8 Bits	Length 5 Bits
0	0	0	0	0.563	1.063
0	0	0	1	0.625	1.125
0	0	1	0	0.688	1.188
0	0	1	1	0.750	1.250
0	1	0	0	0.813	1.313
0	1	0	1	0.875	1.375
0	1	1	0	0.938	1.438
0	1	1	1	1.000	1.500
1	0	0	0	1.563	1.563
1	0	0	1	1.625	1.625
1	0	1	0	1.688	1.688
1	0	1	1	1.750	1.750
1	1	0	0	1.813	1.813
1	1	0	1	1.875	1.875
1	1	1	0	1.938	1.938
1	1	1	1	2.000	2.000

8.4.1.13 OUTPUT PORT DATA REGISTER (OP). The bits in the OP register are set by performing a bit set command (writing to offset $71E) and are cleared by performing a bit reset command (writing to offset $71F). This register can only be written when the serial module is enabled (i.e., the STP bit in the MCR is cleared).

Bit Set

OP $71E

7	6	5	4	3	2	1	0
$\overline{OP7}$	$\overline{OP6}$	$\overline{OP5}$	$\overline{OP4}$	$\overline{OP3}$	$\overline{OP2}$	$\overline{OP1}$	$\overline{OP0}$
RESET:							
0	0	0	0	0	0	0	0

Write Only Supervisor/User

NOTE

OP bits 7, 5, 3, and 2 are not pinned out on the MC68349; thus, changing these bits has no effect.

$\overline{OP6}$, $\overline{OP4}$, $\overline{OP1}$, $\overline{OP0}$ —Output Port Parallel Outputs

The state of these register bits is the complement of the output signal level (e.g., a reset clears these bits (logic 0), while the output signals are asserted (logic 1))

1 = These bits can be set by writing a one to the bit position(s) at this address.
0 = These bits are not affected by writing a zero to this address.

Bit Reset

OP $71F

7	6	5	4	3	2	1	0
$\overline{OP7}$	$\overline{OP6}$	$\overline{OP5}$	$\overline{OP4}$	$\overline{OP3}$	$\overline{OP2}$	$\overline{OP1}$	$\overline{OP0}$
RESET:							
0	0	0	0	0	0	0	0

Write Only Supervisor/User

NOTE

OP bits 7, 5, 3, and 2 are not pinned out on the MC68349; thus, changing these bits has no effect.

$\overline{OP6}$, $\overline{OP4}$, $\overline{OP1}$, $\overline{OP0}$ —Output Port Parallel Outputs

The state of these register bits is the complement of the output signal level (e.g., a reset clears these bits (logic 0), while the output signals are asserted (logic 1))

1 = These bits can be cleared by writing a one to the bit position(s) at this address.
0 = These bits are not affected by writing a zero to this address.

8.4.1.14 OUTPUT PORT CONTROL REGISTER (OPCR). The OPCR individually configures four bits of the 8-bit parallel OP for general-purpose use or as an auxiliary function serving the communication channels. This register can only be written when the serial module is enabled (i.e., the STP bit in the MCR is cleared).

OPCR $71D

7	6	5	4	3	2	1	0
OP7 / $\overline{\text{TxRDYB}}$	OP6 / $\overline{\text{TxRDYA}}$	OP5 / RxRDYB	OP4 / RxRDYA	OP3	OP2	OP1 / $\overline{\text{RTSB}}$	OP0 / $\overline{\text{RTSA}}$
RESET:							
0	0	0	0	0	0	0	0

Write Only Supervisor/User

NOTE

OP bits 7, 5, 3, and 2 are not pinned out on the MC68349; thus, changing these bits has no effect.

OP6—Output Port 6/$\overline{\text{TxRDYA}}$

1 = The OP6/$\overline{\text{TxRDYA}}$ pin functions as the transmitter-ready signal for channel A. The signal reflects the complement of the value of bit 2 of the SRA; thus, $\overline{\text{TxRDYA}}$ is a logic zero when the transmitter is ready.

0 = The OP6/$\overline{\text{TxRDYA}}$ pin functions as a dedicated output. The signal reflects the complement of the value of bit 6 of the OP.

OP4—Output Port 4/$\overline{\text{RxRDYA}}$

1 = The OP4/$\overline{\text{RxRDYA}}$ pin functions as the FIFO-full or receiver-ready signal for channel A (depending on the value of bit 6 of MR1A). The signal reflects the complement of the value of ISR bit 1; thus, $\overline{\text{RxRDYA}}$ is a logic zero when the receiver is ready.

0 = The OP4/$\overline{\text{RxRDYA}}$ pin functions as a dedicated output. The signal reflects the complement of the value of bit 4 of the OP.

OP1—Output Port 1/$\overline{\text{RTSB}}$

1 = The OP1/$\overline{\text{RTSB}}$ pin functions as the ready-to-send signal for channel B. The signal is asserted and negated according to the configuration programmed by RxRTS bit 7 in the MR1B for the receiver and TxRTS bit 5 in the MR2B for the transmitter.

0 = The OP1/$\overline{\text{RTSB}}$ pin functions as a dedicated output. The signal reflects the complement of the value of bit 1 of the OP.

OP0—Output Port 0/$\overline{\text{RTSA}}$

1 = The OP0/$\overline{\text{RTSA}}$ pin functions as the ready-to-send signal for channel A. The signal is asserted and negated according to the configuration programmed by RxRTS bit 7 in the MR1A for the receiver and TxRTS bit 5 in the MR2A for the transmitter.

0 = The OP0/$\overline{\text{RTSA}}$ pin functions as a dedicated output. The signal reflects the complement of the value of bit 0 of the OP.

8.4.1.15 RECEIVER BUFFER (RB). The RB contains three receiver holding registers and a serial shift register. The channel's RxDx pin is connected to the serial shift register. The holding registers act as a FIFO. The CPU32+ reads from the top of the stack while the receiver shifts and updates from the bottom of the stack when the shift register has been filled (see Figure 8-4). This register can only be read when the serial module is enabled (i.e., the STP bit in the MCR is cleared).

RBA, RBB $713, $71B

7	6	5	4	3	2	1	0
RB7	RB6	RB5	RB4	RB3	RB2	RB1	RB0
RESET:							
0	0	0	0	0	0	0	0

Read Only Supervisor/User

RB7–RB0—These bits contain the character in the RB.

8.4.1.16 STATUS REGISTER (SR). The SR indicates the status of the characters in the FIFO and the status of the channel transmitter and receiver. This register can only be read when the serial module is enabled (i.e., the STP bit in the MCR is cleared).

SRA, SRB $711, $719

7	6	5	4	3	2	1	0
RB	FE	PE	OE	TxEMP	TxRDY	FFULL	RxRDY
RESET:							
0	0	0	0	0	0	0	0

Read Only Supervisor/User

RB—Received Break

1 = An all-zero character of the programmed length has been received without a stop bit. The RB bit is only valid when the RxRDY bit is set. Only a single FIFO position is occupied when a break is received. Further entries to the FIFO are inhibited until the channel RxDx returns to the high state for at least one-half bit time, which is equal to two successive edges of the internal or external 1× clock or 16 successive edges of the external 16× clock.
The received break circuit detects breaks that originate in the middle of a received character. However, if a break begins in the middle of a character, it must persist until the end of the next detected character time.

0 = No break has been received.

FE—Framing Error

1 = A stop bit was not detected when the corresponding data character in the FIFO was received. The stop-bit check is made in the middle of the first stop-bit position. The bit is valid only when the RxRDY bit is set.

0 = No framing error has occurred.

PE—Parity Error

1 = When the with parity or force parity mode is programmed in the MR1, the corresponding character in the FIFO was received with incorrect parity. When the multidrop mode is programmed, this bit stores the received A/D bit. This bit is valid only when the RxRDY bit is set.

0 = No parity error has occurred.

OE—Overrun Error

1 = One or more characters in the received data stream have been lost. This bit is set upon receipt of a new character when the FIFO is full and a character is already in the shift register waiting for an empty FIFO position. When this occurs, the character in the receiver shift register and its break detect, framing error status, and parity error, if any, are lost. This bit is cleared by the reset error status command in the CR.

0 = No overrun has occurred.

TxEMP—Transmitter Empty

1 = The channel transmitter has underrun (both the transmitter holding register and transmitter shift registers are empty). This bit is set after transmission of the last stop bit of a character if there are no characters in the transmitter holding register awaiting transmission.

0 = The transmitter buffer is not empty. The transmitter holding register is loaded by the CPU32+, or the transmitter is disabled. The transmitter is enabled/disabled by programming the TCx bits in the CR.

TxRDY—Transmitter Ready

This bit is duplicated in the ISR; bit 0 for channel A and bit 4 for channel B.

1 = The transmitter holding register is empty and ready to be loaded with a character. This bit is set when the character is transferred to the transmitter shift register. This bit is also set when the transmitter is first enabled. Characters loaded into the transmitter holding register while the transmitter is disabled are not transmitted and are lost.

0 = The transmitter holding register was loaded by the CPU32+, or the transmitter is disabled.

FFULL—FIFO Full

1 = A character was transferred from the receiver shift register to the receiver FIFO and the transfer caused the FIFO to become full (all three FIFO holding register positions are occupied).

0 = The CPU32+ has read the receiver buffer and one or more FIFO positions are available. Note that if there is a character in the receiver shift register because the FIFO is full, this character will be moved into the FIFO when a position is available, and the FIFO will remain full.

RxRDY—Receiver Ready

1 = A character has been received and is waiting in the FIFO to be read by the CPU32+. This bit is set when a character is transferred from the receiver shift register to the FIFO.

0 = The CPU32+ has read the receiver buffer, and no characters remain in the FIFO after this read.

8.4.1.17 TRANSMITTER BUFFER (TB). The TB consists of two registers, the transmitter holding register and the transmitter shift register (see Figure 8-4). The holding register accepts characters from the bus master if the TxRDY bit in the channel's SR is set. A write to the TB clears the TxRDY bit, inhibiting any more characters until the shift register is ready to accept more data. When the shift register is empty, it checks to see if the holding register has a valid character to be sent (TxRDY bit cleared). If there is a valid character, the shift register loads the character and reasserts the TxRDY bit in the channel's SR. Writes to the TB when the channel's SR TxRDY bit is clear and when the transmitter is disabled have no effect on the TB. This register can only be written when the serial module is enabled (i.e., the STP bit in the MCR is cleared).

TBA, TBB $713, $71B

7	6	5	4	3	2	1	0
TB7	TB6	TB5	TB4	TB3	TB2	TB1	TB0
RESET:							
0	0	0	0	0	0	0	0

Write Only Supervisor/User

TB7–TB0—These bits contain the character in the TB.

8.4.2 Programming

The basic interface software flowchart required for operation of the serial module is shown in Figure 8-10. The routines are divided into three categories:

- Serial Module Initialization
- I/O Driver
- Interrupt Handling

8.4.2.1 SERIAL MODULE INITIALIZATION. The serial module initialization routines consist of SINIT and CHCHK. SINIT is called at system initialization time to check channel A and channel B operation. Before SINIT is called, the calling routine allocates two words on the system stack. Upon return to the calling routine, SINIT passes information on the system stack to reflect the status of the channels. If SINIT finds no errors in either channel A or channel B, the respective receivers and transmitters are enabled. The CHCHK routine performs the actual channel checks as called from the SINIT routine. When called, SINIT places the specified channel in the local loopback mode and checks for the following errors:

- Transmitter Never Ready
- Receiver Never Ready
- Parity Error
- Incorrect Character Received

8.4.2.2 I/O DRIVER EXAMPLE. The I/O driver routines consist of INCH, OUTCH, and POUTCH. INCH is the terminal input character routine and gets a character from the channel A receiver and places it in the lower byte of register D0. OUTCH is used to send the character in the lower byte of register D0 to the channel A transmitter. POUTCH sends the character in the lower byte of D0 to the channel B transmitter.

8.4.2.3 INTERRUPT HANDLING. The interrupt handling routine consists of SIRQ, which is executed after the serial module generates an interrupt caused by a channel A change-in-break (beginning of a break). SIRQ then clears the interrupt source, waits for the next change-in-break interrupt (end of break), clears the interrupt source again, then returns from exception processing to the system monitor.

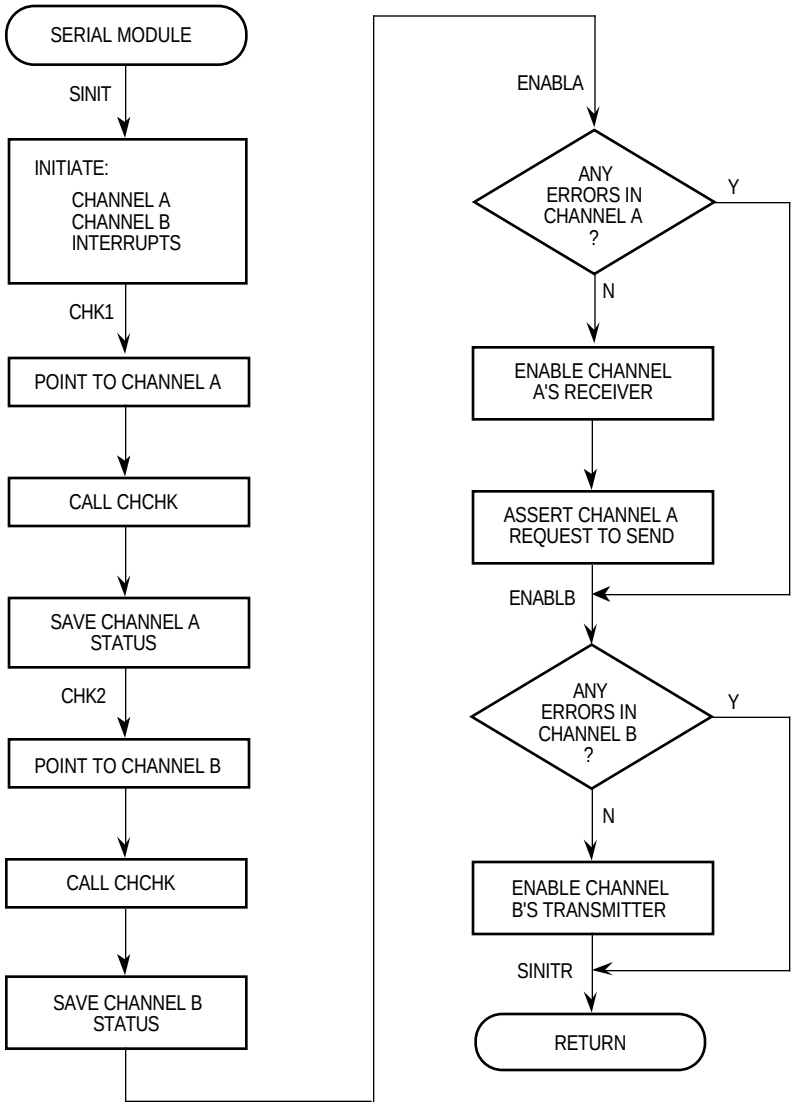


Figure 8-10. Serial Module Programming Flowchart (1 of 5)

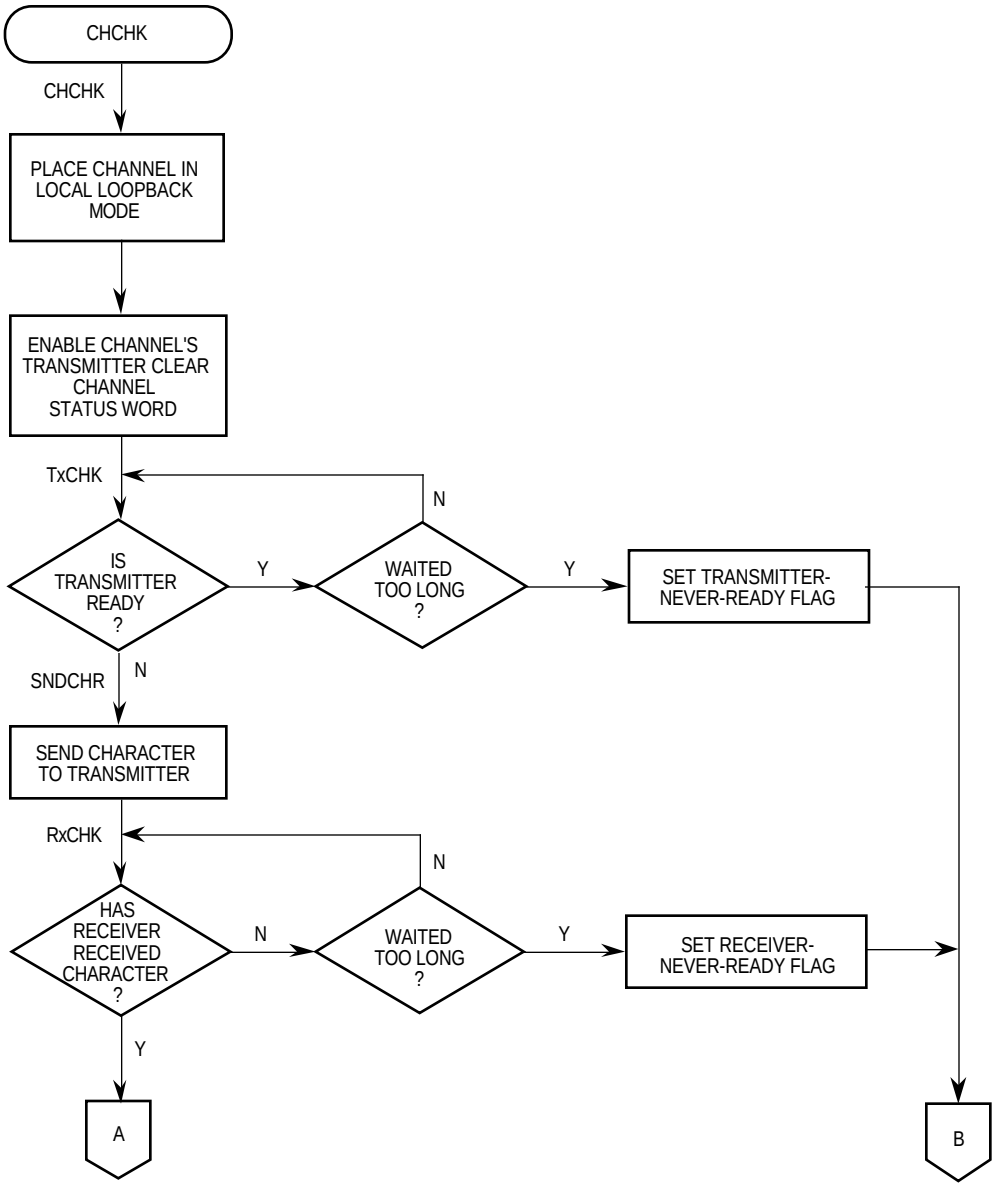


Figure 8-10. Serial Module Programming Flowchart (2 of 5)

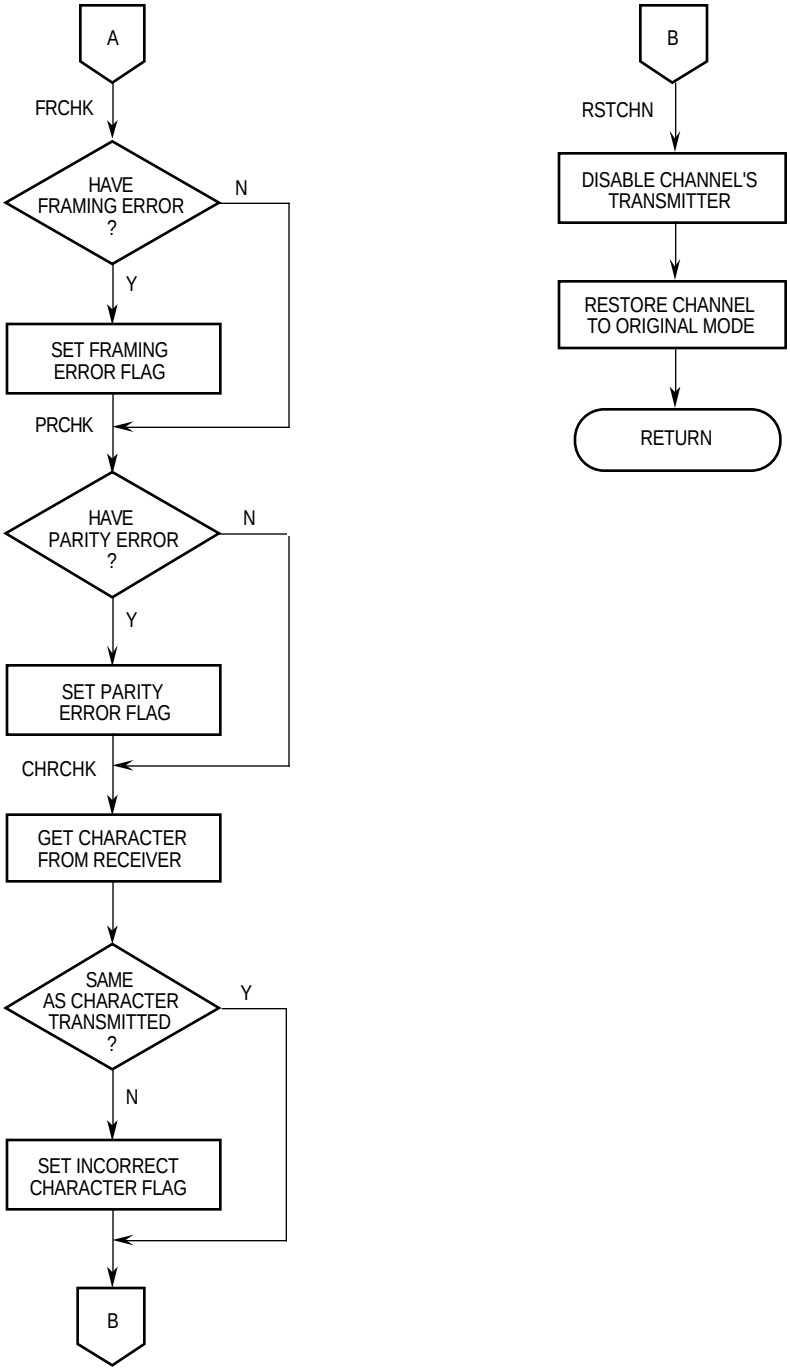


Figure 8-10. Serial Module Programming Flowchart (3 of 5)

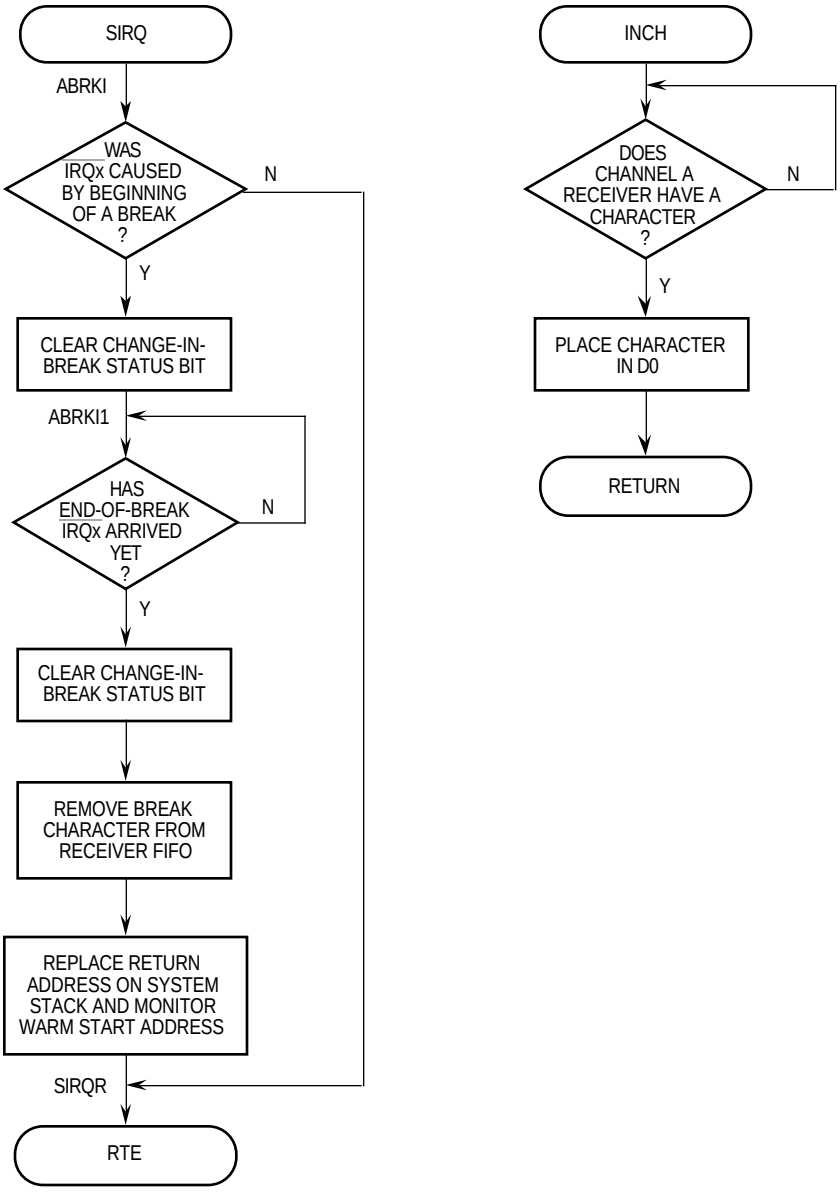


Figure 8-10. Serial Module Programming Flowchart (4 of 5)

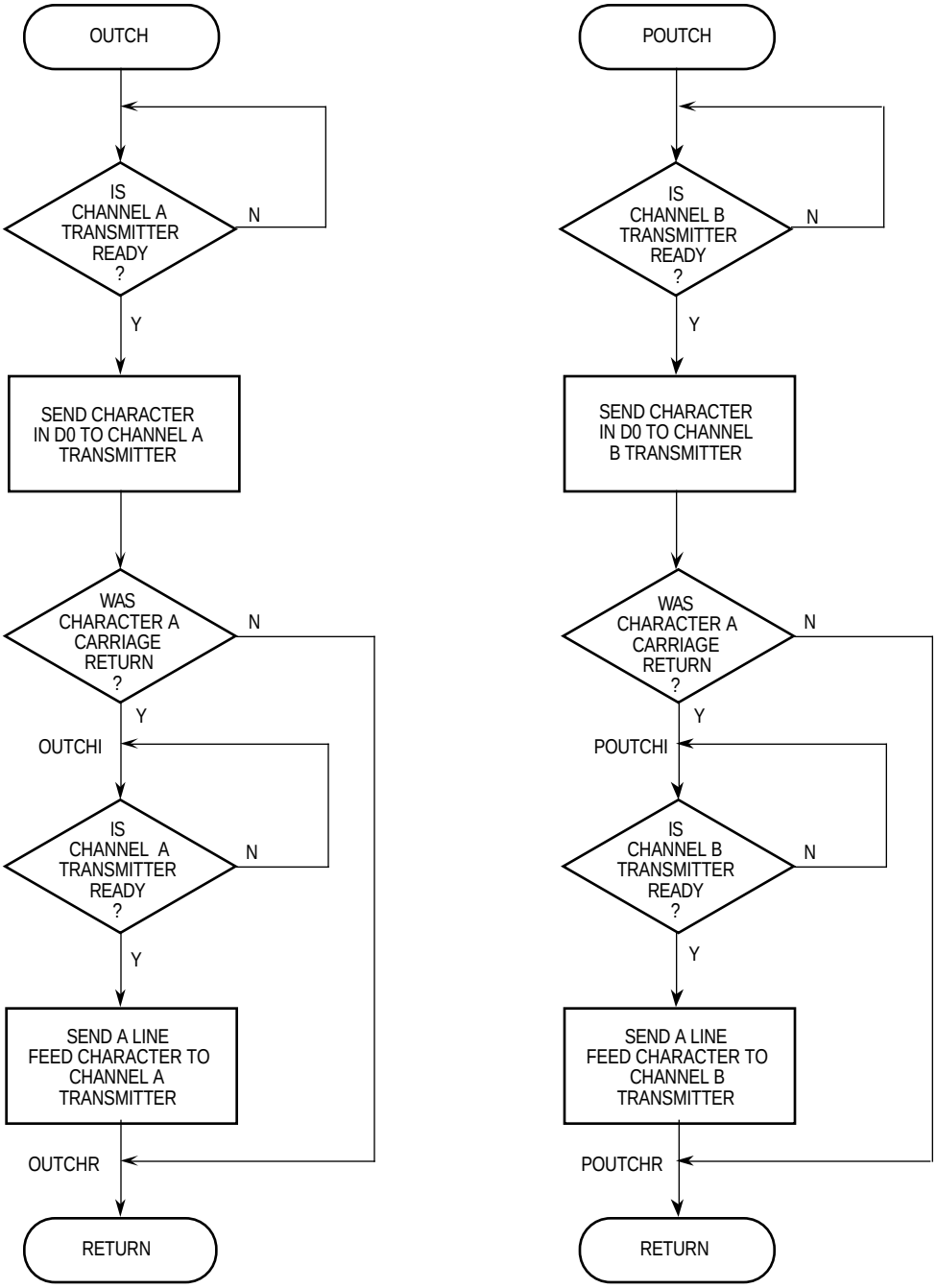


Figure 8-10. Serial Module Programming Flowchart (5 of 5)

8.5 SERIAL MODULE INITIALIZATION SEQUENCE

The following paragraphs discuss a suggested method for initializing the serial module.

8.5.1 Serial Module Configuration

If the serial capability of the MC68349 is being used, the following steps are required to properly initialize the serial module.

NOTE

The serial module registers can only be accessed by byte operations.

Command Register (CR)

- Reset the receiver and transmitter for each channel.

The following steps program both channels:

Module Configuration Register (MCR)

- Initialize the stop bit (STP) for normal operation.
- Select whether to respond to or ignore FREEZE (FRZx bits).
- Select the input capture clock (ICCS bit).
- Select the access privilege for the supervisor/user registers (SUPV bit).
- Select the interrupt arbitration level for the serial module (IARBx bits).

Interrupt Vector Register (IVR)

- Program the vector number for a serial module interrupt.

Interrupt Level Register (ILR)

- Program the interrupt priority level for a serial module interrupt.

Interrupt Enable Register (IER)

- Enable the desired interrupt sources.

Auxiliary Control Register (ACR)

- Select baud rate set (BRG bit).
- Initialize the input enable control (IEC bits).

Output Port Control Register (OPCR)

- Select the function of the output port pins.

Interrupt Status Register (ISR)

- The XTAL_RDY bit should be polled until it is cleared to ensure that an unstable crystal input is not applied to the baud rate generator.

The following steps are channel specific:

Clock Select Register (CSR)

- Select the receiver and transmitter clock.

Mode Register 1 (MR1)

- If desired, program operation of receiver ready-to-send (RxRTS bit).
- Select receiver-ready or FIFO-full notification (R/F bit).
- Select character or block error mode (ERR bit).
- Select parity mode and type (PM and PT bits).
- Select number of bits per character (B/Cx bits).

Mode Register 2 (MR2)

- Select the mode of channel operation (CMx bits).
- If desired, program operation of transmitter ready-to-send (TxRTS bit).
- If desired, program operation of clear-to-send (TxCTS bit).
- Select stop-bit length (SBx bits).

Command Register (CR)

- Enable the receiver and transmitter.

8.5.2 Serial Module Example Configuration Code

The following code is an example of a configuration sequence for the serial module.

```
*************************************************************************
* MC68349 basic serial module register initialization example code.
* This code is used to initialize the 68349's internal serial module registers,
* providing basic functions for operation.
* It sets up serial channel A for communication with a 9600 baud terminal.
* Note: All serial module registers must be accessed as bytes.
*************************************************************************
*************************************************************************
* equates
*************************************************************************
MBAR  EQU   $0003FF00   Address of SIM49 Module Base Address Reg.
MODBASE     EQU   $FFFFF000   SIM49 MBAR address value

*************************************************************
* Serial module equates
SERIAL      EQU   $700              Offset from MBAR for serial module regs
MCRH  EQU   $0                serial MCR high byte
MCRL  EQU   $1                serial MCR low byte

* Serial register offsets from serial base address
MR1A      EQU     $10         Mode register 1 A
MR2A      EQU     $20         Mode register 2 A
SRA       EQU     $11         Status register A
CSRA      EQU     $11         Clock select reg A
CRA       EQU     $12         Command reg A

ACR       EQU     $14         Auxiliary control reg
OPCR      EQU     $1D         Output port control reg
OP_BS     EQU     $1E         Output port bit set (write 1 to set)
OP_BR     EQU     $1F         Output port bit reset (write 1 to clear)


*************************************************************************
*************************************************************************
* Initialize Serial channel A
*************************************************************************
      LEA   MODBASE+SERIAL,A0 Pointer to serial channel A

* Module configuration register:
* Enable serial module for normal operation, ignore FREEZE, select the
* crystal clock. Supervisor/user serial registers unrestricted.
* Interrupt arbitration at priority $02.
      MOVE.B      #$00,MCRH(A0)
      MOVE.B      #$02,MCRL(A0)

* WAIT FOR TRANSMITTER EMPTY (OR TIMEOUT)
      MOVE.W      #$2000,D0               init loop counter
XBMTWAIT EQU      *
      BTST        #3,SRA(A0)              TX empty in status reg?
      NOP
      DBNE  D0,XBMTWAIT       loop until set or timeout
```

```
* NEGATE RTSA SIGNAL OUTPUT
      MOVE.B      #0,OPCR(A0)       make OP0-7 general purpose
      MOVE.B      #$01,OP_BR(A0)          clear RTSA/OP0 output

* RESET RECEIVER/TRANSMITTER
      MOVE.B      #$20,CRA(A0)            Issue reset receiver command
      MOVE.B      #$30,CRA(A0)            Issue reset transmitter command

* SET BAUD RATE SET 2
      MOVE.B      #$80,ACR(A0)

* MODE REGISTER 1
      MOVE.B      #$93,MR1A(A0)           8 bits, no parity, auto RTS control

* MODE REGISTER 2
      MOVE.B      #$07,MR2A(A0)           Normal, 1 stop bit

* SET UP BAUD RATE FOR PORT IN CLOCK SELECT REGISTER
      MOVE.B      #$BB,CSRA(A0)           Set 9600 baud for RX and TX

* SET RTSA ACTIVE
      MOVE.B      #$01,OP_BS(A0)          set RTSA/OP0 output

* ENABLE PORT
      MOVE.B      #$45,CRA(A0)            Reset error status, enable RX & TX

*************************************************************************
      END
*************************************************************************
```

SECTION 9
IEEE 1149.1 TEST ACCESS PORT

The MC68349 includes dedicated user-accessible test logic that is fully compatible with the *IEEE 1149.1 Standard Test Access Port and Boundary Scan Architecture*. Problems associated with testing high-density circuit boards have led to development of this proposed standard under the sponsorship of the Test Technology Committee of IEEE and the Joint Test Action Group (JTAG). The MC68349 implementation supports circuit-board test strategies based on this standard.

The test logic includes a test access port (TAP) consisting of four dedicated signal pins, a 16-state controller, an instruction register, and two test data registers. A boundary scan register links all device signal pins into a single shift register. The test logic, implemented using static logic design, is independent of the device system logic. The MC68349 implementation provides the following capabilities:

a. Perform boundary scan operations to test circuit-board electrical continuity
b. Sample the MC68349 system pins during operation and transparently shift out the result in the boundary scan register
c. Bypass the MC68349 for a given circuit-board test by effectively reducing the boundary scan register to a single bit
d. Disable the output drive to pins during circuit-board testing

NOTE

Certain precautions must be observed to ensure that the IEEE 1149.1 test logic does not interfere with nontest operation. See **9.6 Non-IEEE 1149.1 Operation** for details.

9.1 OVERVIEW

NOTE

This description is not intended to be used without the supporting IEEE 1149.1 document.

This discussion includes those items required by the standard and provides additional information specific to the MC68349 implementation. For internal details and applications of the standard, refer to the IEEE 1149.1 document.

An overview of the MC68349 implementation of IEEE 1149.1 is shown in Figure 9-1. The MC68349 implementation includes a 16-state controller, a 3-bit instruction register, and two test registers (a 1-bit bypass register and a 141-bit boundary scan register). This implementation includes a dedicated TAP consisting of the following signals:

TCK — a test clock input to synchronize the test logic
TMS — a test mode select input (with an internal pullup resistor) that is sampled on the rising edge of TCK to sequence the TAP controller's state machine
TDI — a test data input (with an internal pullup resistor) that is sampled on the rising edge of TCK.
TDO — a three-state test data output that is actively driven in the shift-IR and shift-DR controller states. TDO changes on the falling edge of TCK.

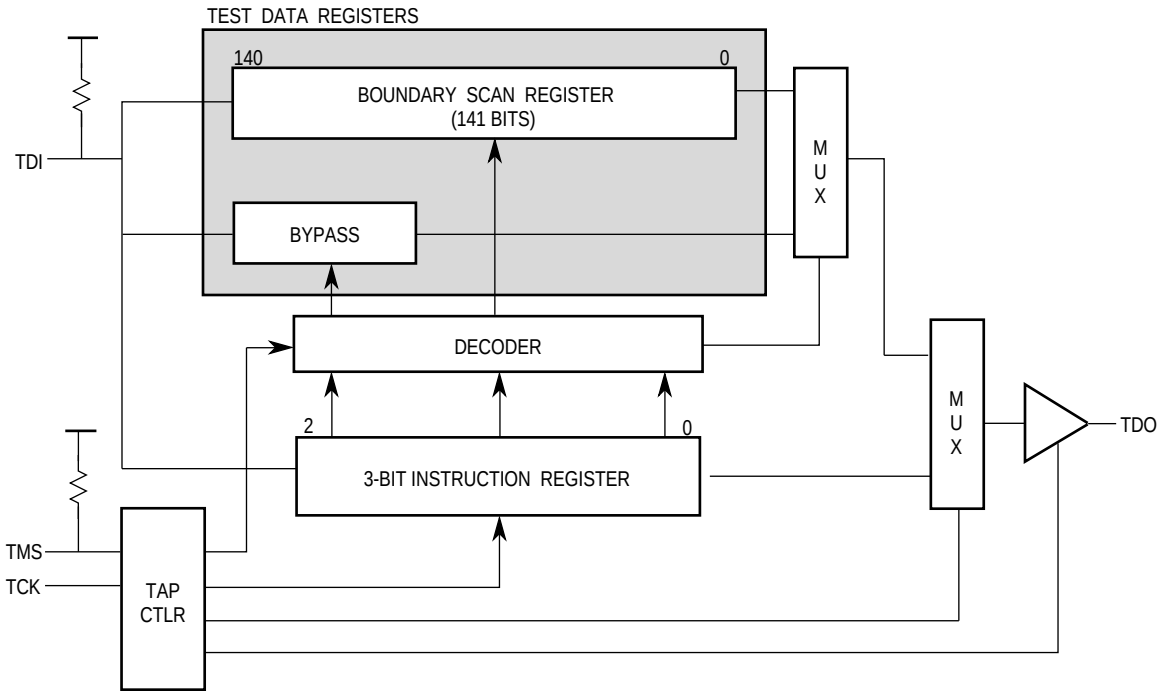


Figure 9-1. Test Access Port Block Diagram

9.2 TAP CONTROLLER

The TAP controller is responsible for interpreting the sequence of logical values on the TMS signal. It is a synchronous state machine that controls the operation of the JTAG logic. The state machine is shown in Figure 9-2; the value shown adjacent to each arc represents the value of the TMS signal sampled on the rising edge of the TCK signal. For a description of the TAP controller states, please refer to the IEEE 1149.1 document.

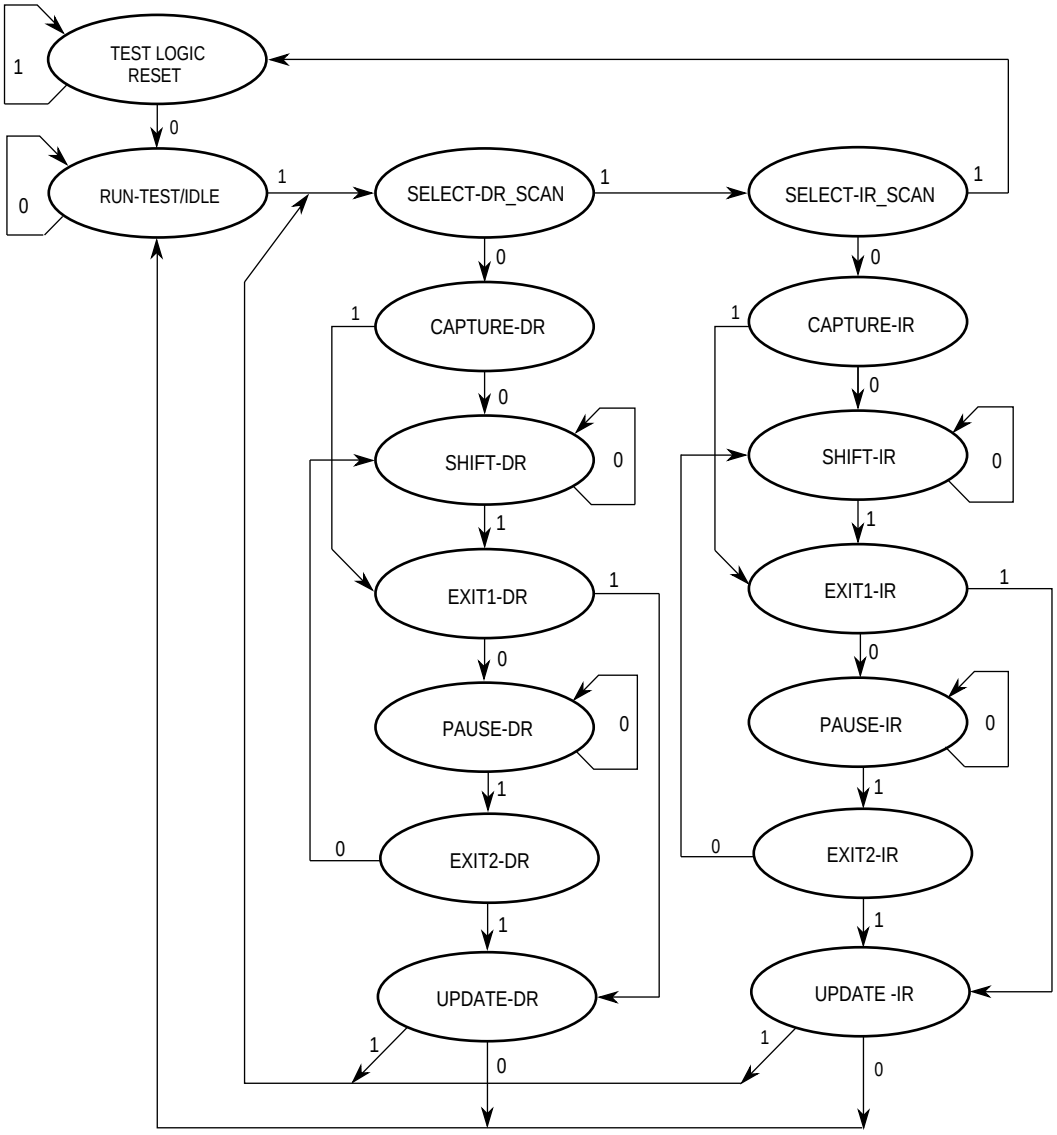


Figure 9-2. TAP Controller State Machine

9.3 BOUNDARY SCAN REGISTER

The MC68349 IEEE 1149.1 implementation has a 141-bit boundary scan register. This register contains bits for all device signal and clock pins and associated control signals. The XTAL, X2, and XFC pins are associated with analog signals and are not included in the boundary scan register.

All MC68349 bidirectional pins, except the open-drain I/O pins ($\overline{\text{DONE1}}$, $\overline{\text{DONE2}}$, $\overline{\text{HALT}}$, and $\overline{\text{RESET}}$), have a single register bit for pin data and an associated control bit in the boundary scan register. All open drain I/O pins have two register bits, input and output, for pin data and no associated control bit. To ensure proper operation, the open-drain pins require external pullups. Twenty-one control bits in the boundary scan register define the output enable signal for associated groups of bidirectional and three-state pins. The control bits and their bit positions are listed in Table 9-1.

Table 9-1. Boundary Scan Control Bits

Name	Bit Number	Name	Bit Number	Name	Bit Number
irq7.ctl	49	cs0.ctl	63	ab27.ctl	96
irq6.ctl	51	ab.ctl	86	ab28.ctl	98
irq5.ctl	53	berr.ctl	87	ab29.ctl	100
cs3.ctl	55	db.ctl	88	ab30.ctl	102
cs2.ctl	57	ab24.ctl	90	ab31.ctl	104
cs1.ctl	59	ab25.ctl	92	modck.ctl	132
irq3.ctl	61	ab26.ctl	94	ifetch.ctl	135

Boundary scan bit definitions are shown in Table 9-2. The first column in Table 9-2 defines the bit's ordinal position in the boundary scan register. The shift register bit nearest TDO (i.e., first to be shifted out) is defined as bit 0; the last bit to be shifted out is 140.

The second column references one of the five MC68349 cell types depicted in Figures 9-3–9-7, which describe the cell structure for each type.

The third column lists the pin name for all pin-related bits or defines the name of bidirectional control register bits. The active level of the control bits (i.e., output driver on) is defined by the last digit of the cell type listed for each control bit. For example, the active-high level for irq7.ctl (bit 49) is logic zero since the cell type is IO.Ctl0. The active level for ab.ctl (bit 86) is logic one, since the cell type is IO.Ctl1. IO.Ctl0 (see Figure 9-6) differs from IO.Ctl1 (see Figure 9-5) by an inverter in the output enable path.

The fourth column lists the pin type: I/O indicates a bidirectional pin and OD-I/O denotes an open-drain bidirectional pin. An open-drain output pin has two states: off (high impedance) and logic zero.

The last column indicates the associated boundary scan register control bit for bidirectional, three-state, and open-drain output pins.

Bidirectional pins include a single scan bit for data (IO.Cell) as depicted in Figure 9-7. These bits are controlled by one of the two bits shown in Figures 9-5 and 9-6. The value of the control bit determines whether the bidirectional pin is an input or an output. One or more bidirectional data bits can be serially connected to a control bit as shown in Figure 9-8. Note that, when sampling the bidirectional data bits, the bit data can be interpreted only after examining the IO control bit to determine pin direction.

Table 9-2. Boundary Scan Bit Definitions

Bit Num	Cell Type	Pin/Cell Name	Pin Type	Output CTL Cell
0	IO.Cell	FC3	I/O	ab.ctl
1	IO.Cell	FC2	I/O	ab.ctl
2	IO.Cell	FC1	I/O	ab.ctl
3	IO.Cell	FC0	I/O	ab.ctl
4	IO.Cell	A23	I/O	ab.ctl
5	IO.Cell	A22	I/O	ab.ctl
6	IO.Cell	A21	I/O	ab.ctl
7	IO.Cell	A20	I/O	ab.ctl
8	IO.Cell	A19	I/O	ab.ctl
9	IO.Cell	A18	I/O	ab.ctl
10	IO.Cell	A17	I/O	ab.ctl
11	IO.Cell	A16	I/O	ab.ctl
12	IO.Cell	A15	I/O	ab.ctl
13	IO.Cell	A14	I/O	ab.ctl
14	IO.Cell	A13	I/O	ab.ctl
15	IO.Cell	A12	I/O	ab.ctl
16	IO.Cell	A11	I/O	ab.ctl
17	IO.Cell	A10	I/O	ab.ctl
18	IO.Cell	A9	I/O	ab.ctl
19	IO.Cell	A8	I/O	ab.ctl
20	IO.Cell	A7	I/O	ab.ctl
21	IO.Cell	A6	I/O	ab.ctl
22	IO.Cell	A5	I/O	ab.ctl
23	IO.Cell	A4	I/O	ab.ctl
24	IO.Cell	A3	I/O	ab.ctl
25	IO.Cell	A2	I/O	ab.ctl
26	IO.Cell	A1	I/O	ab.ctl
27	IO.Cell	A0	I/O	ab.ctl
28	I.Pin	RxDA	Input	—
29	O.Latch	TxDA	Output	—
30	O.Latch	$\overline{\text{RTSA}}$	Output	—
31	I.Pin	$\overline{\text{CTSA}}$	Input	—
32	O.Latch	$\overline{\text{RxRDYA}}$	Output	—
33	O.Latch	$\overline{\text{TxRDYA}}$	Output	—
34	I.Pin	RxDB	Input	—
35	O.Latch	TxDB	Output	—
36	O.Latch	$\overline{\text{RTSB}}$	Output	—
37	I.Pin	$\overline{\text{CTSB}}$	Input	—
38	I.Pin	SCLK	Input	—
39	I.Pin	X1	Input	—
40	I.Pin	$\overline{\text{DREQ1}}$	Input	—
41	O.Latch	$\overline{\text{DACK1}}$	Output	—
42	O.Latch	$\overline{\text{DONE1}}$	OD-I/O	—
43	I.Pin	$\overline{\text{DONE1}}$	OD-I/O	—
44	I.Pin	$\overline{\text{DREQ2}}$	Input	—
45	O.Latch	$\overline{\text{DACK2}}$	Output	—
46	O.Latch	$\overline{\text{DONE2}}$	OD-I/O	—
47	I.Pin	$\overline{\text{DONE2}}$	OD-I/O	—
48	IO.Cell	$\overline{\text{IRQ7}}$	I/O	irq7.ctl
49	IO.Ctl0	irq7.ctl	—	—
50	IO.Cell	$\overline{\text{IRQ6}}$	I/O	irq6.ctl
51	IO.Ctl0	irq6.ctl	—	—
52	IO.Cell	$\overline{\text{IRQ5}}$	I/O	irq5.ctl
53	IO.Ctl0	irq5.ctl	—	—
54	IO.Cell	$\overline{\text{CS3}}$	I/O	cs3.ctl
55	IO.Ctl0	cs3.ctl	—	—
56	IO.Cell	$\overline{\text{CS2}}$	I/O	cs2.ctl
57	IO.Ctl0	cs2.ctl	—	—
58	IO.Cell	$\overline{\text{CS1}}$	I/O	cs1.ctl
59	IO.Ctl0	cs1.ctl	—	—
60	IO.Cell	$\overline{\text{IRQ3}}$	I/O	irq3.ctl
61	IO.Ctl0	irq3.ctl	—	—
62	IO.Cell	$\overline{\text{CS0}}$	I/O	cs0.ctl
63	IO.Ctl0	cs0.ctl	—	—
64	IO.Cell	D10	I/O	db.ctl
65	IO.Cell	D11	I/O	db.ctl
66	IO.Cell	D12	I/O	db.ctl
67	IO.Cell	D13	I/O	db.ctl
68	IO.Cell	D14	I/O	db.ctl
69	IO.Cell	D15	I/O	db.ctl
70	IO.Cell	D16	I/O	db.ctl
71	IO.Cell	D17	I/O	db.ctl

Table 9-2. Boundary Scan Bit Definitions (Continued)

Bit Num	Cell Type	Pin/Cell Name	Pin Type	Output CTL Cell
72	IO.Cell	D18	I/O	db.ctl
73	IO.Cell	D19	I/O	db.ctl
74	IO.Cell	D20	I/O	db.ctl
75	IO.Cell	D21	I/O	db.ctl
76	IO.Cell	D22	I/O	db.ctl
77	IO.Cell	D23	I/O	db.ctl
78	IO.Cell	D24	I/O	db.ctl
79	IO.Cell	D25	I/O	db.ctl
80	IO.Cell	D26	I/O	db.ctl
81	IO.Cell	D27	I/O	db.ctl
82	IO.Cell	D28	I/O	db.ctl
83	IO.Cell	D29	I/O	db.ctl
84	IO.Cell	D30	I/O	db.ctl
85	IO.Cell	D31	I/O	db.ctl
86	IO.Ctl1	ab.ctl	—	—
87	IO.Ctl0	berr.ctl	—	—
88	IO.Ctl1	db.ctl	—	—
89	IO.Cell	A24	I/O	ab24.ctl
90	IO.Ctl0	ab24.ctl	—	—
91	IO.Cell	A25	I/O	ab25.ctl
92	IO.Ctl0	ab25.ctl	—	—
93	IO.Cell	A26	I/O	ab26.ctl
94	IO.Ctl0	ab26.ctl	—	—
95	IO.Cell	A27	I/O	ab27.ctl
96	IO.Ctl0	ab27.ctl	—	—
97	IO.Cell	A28	I/O	ab28.ctl
98	IO.Ctl0	ab28.ctl	—	—
99	IO.Cell	A29	I/O	ab29.ctl
100	IO.Ctl0	ab29.ctl	—	—
101	IO.Cell	A30	I/O	ab30.ctl
102	IO.Ctl0	ab30.ctl	—	—
103	IO.Cell	A31	I/O	ab31.ctl
104	IO.Ctl0	ab31.ctl	—	—
105	IO.Cell	D9	I/O	db.ctl
106	IO.Cell	D8	I/O	db.ctl
107	IO.Cell	D7	I/O	db.ctl
108	IO.Cell	D6	I/O	db.ctl
109	IO.Cell	D5	I/O	db.ctl
110	IO.Cell	D4	I/O	db.ctl
111	IO.Cell	D3	I/O	db.ctl
112	IO.Cell	D2	I/O	db.ctl
113	IO.Cell	D1	I/O	db.ctl
114	IO.Cell	D0	I/O	db.ctl
115	IO.Cell	$\overline{\text{RMC}}$	I/O	ab.ctl
116	IO.Cell	R/$\overline{\text{W}}$	I/O	ab.ctl
117	IO.Cell	SIZ1	I/O	ab.ctl
118	IO.Cell	SIZ0	I/O	ab.ctl
119	IO.Cell	$\overline{\text{DS}}$	I/O	ab.ctl
120	IO.Cell	$\overline{\text{AS}}$	I/O	ab.ctl
121	I.Pin	$\overline{\text{BGACK}}$	Input	—
122	O.Latch	$\overline{\text{BG}}$	Output	—
123	I.Pin	$\overline{\text{BR}}$	Input	—
124	IO.Cell	$\overline{\text{BERR}}$	I/O	berr.ctl
125	O.Latch	$\overline{\text{HALT}}$	OD-I/O	—
126	I.Pin	$\overline{\text{HALT}}$	OD-I/O	—
127	O.Latch	$\overline{\text{RESET}}$	OD-I/O	—
128	I.Pin	$\overline{\text{RESET}}$	OD-I/O	—
129	O.Latch	CLKOUT	Output	—
130	I.Pin	EXTAL	Input	—
131	IO.Cell	MODCK	I/O	modck.ctl
132	IO.Ctl0	modck.ctl	—	—
133	O.Latch	$\overline{\text{IPIPE0}}$	Output	—
134	IO.Cell	$\overline{\text{IFETCH}}$	I/O	ifetch.ctl
135	IO.Ctl0	ifetch.ctl	—	—
136	I.Pin	$\overline{\text{BKPT}}$	Input	—
137	O.Latch	FREEZE	Output	—
138	IO.Cell	$\overline{\text{DSACK1}}$	I/O	berr.ctl
139	O.Latch	$\overline{\text{IPIPE1}}$	Output	—
140	IO.Cell	$\overline{\text{DSACK0}}$	I/O	berr.ctl
—	—	—	—	—

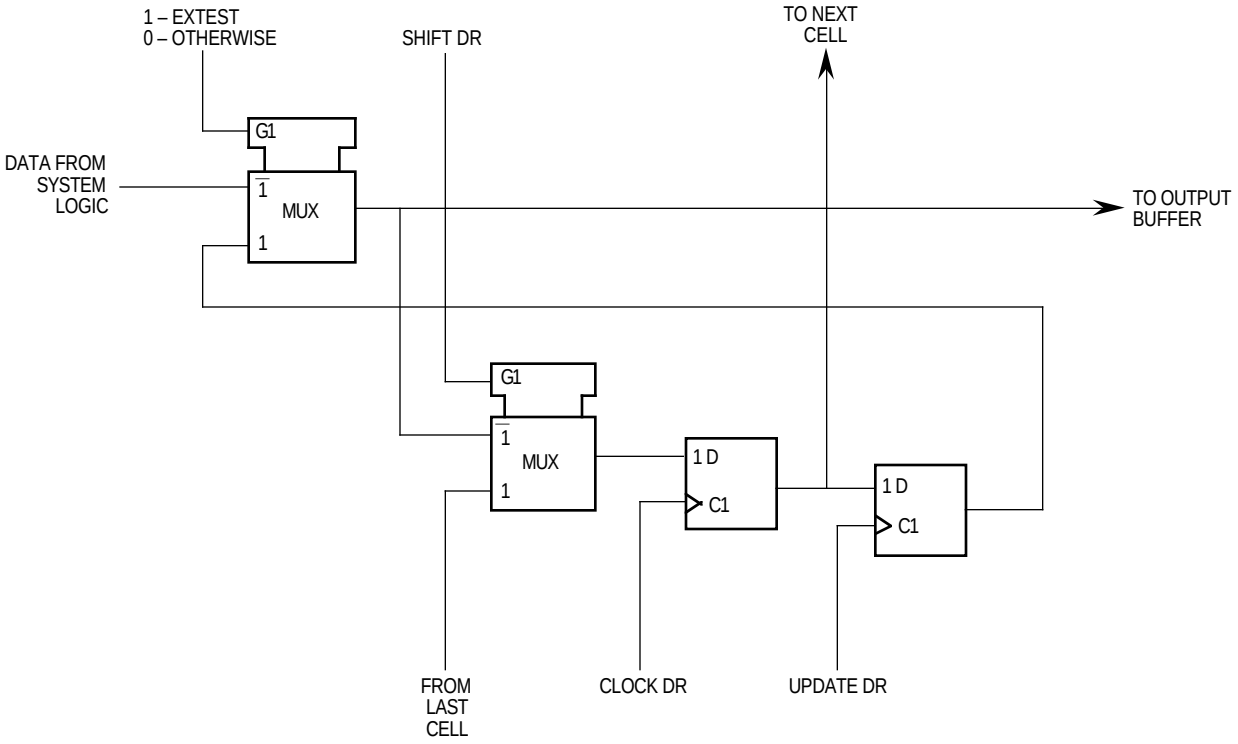


Figure 9-3. Output Latch Cell (O.Latch)

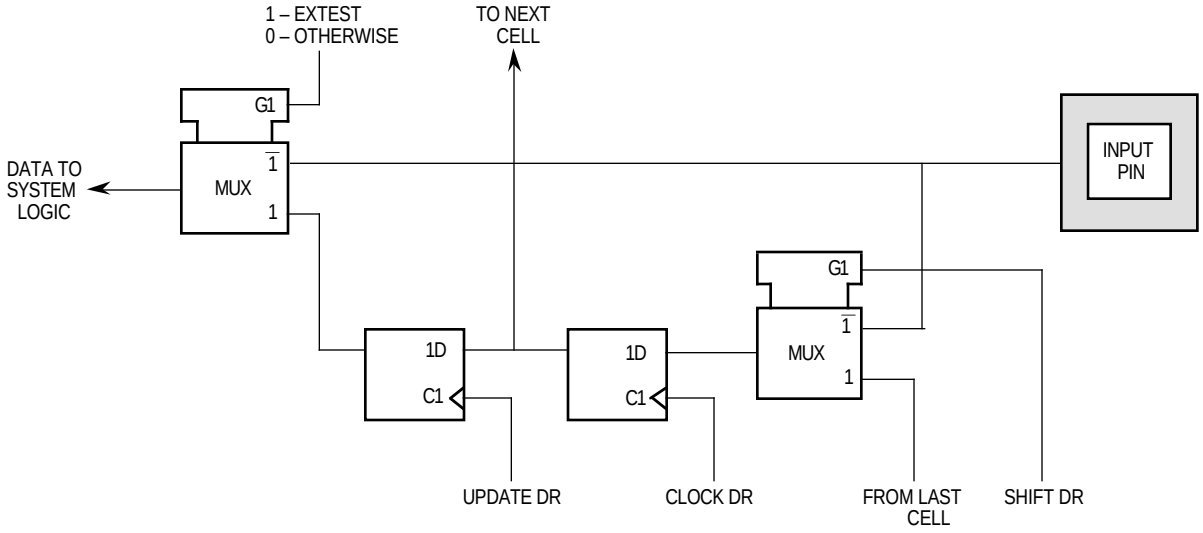


Figure 9-4. Input Pin Cell (I.Pin)

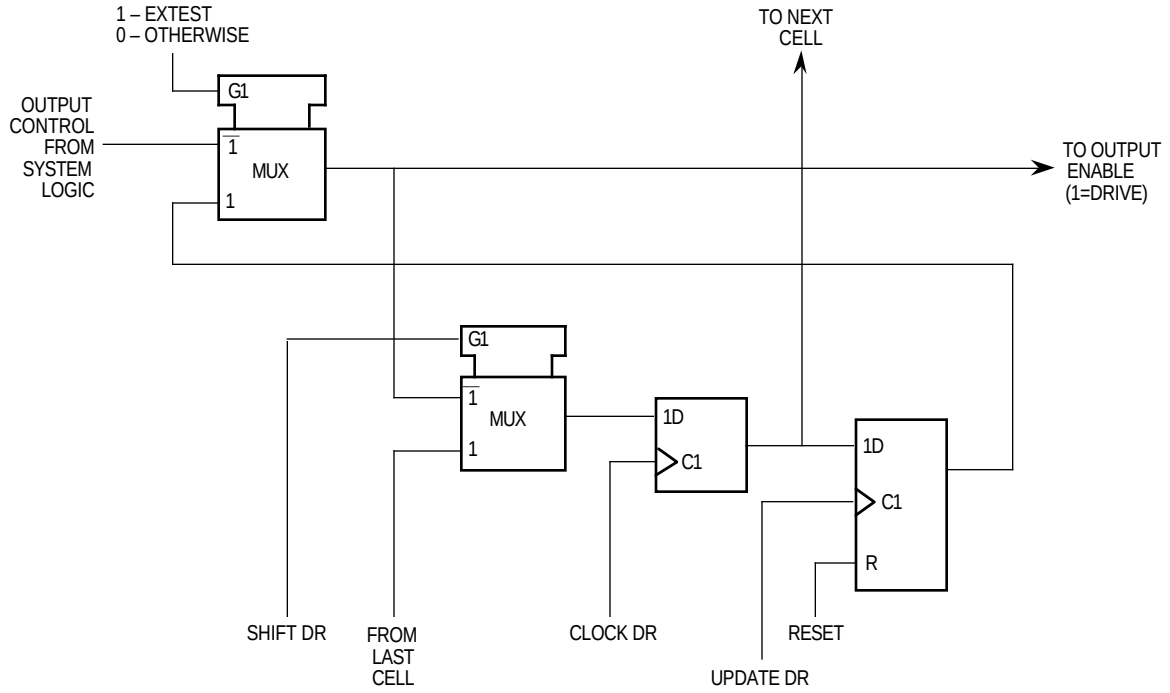


Figure 9-5. Active-High Output Control Cell (IO.Ctl1)

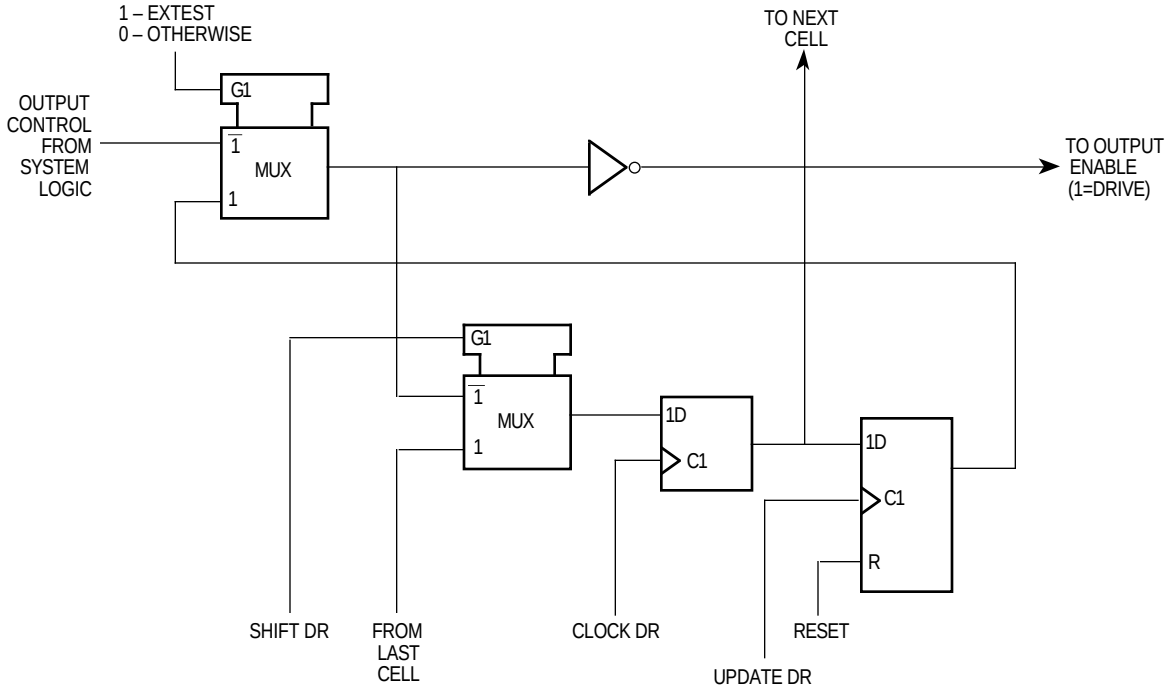


Figure 9-6. Active-Low Output Control Cell (IO.Ctl0)

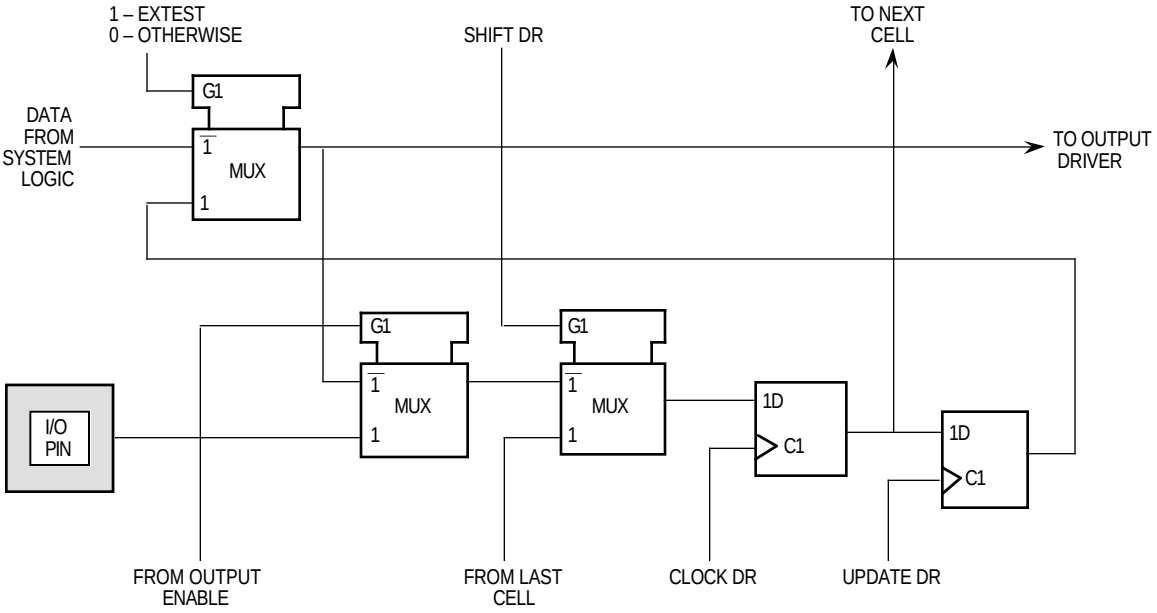
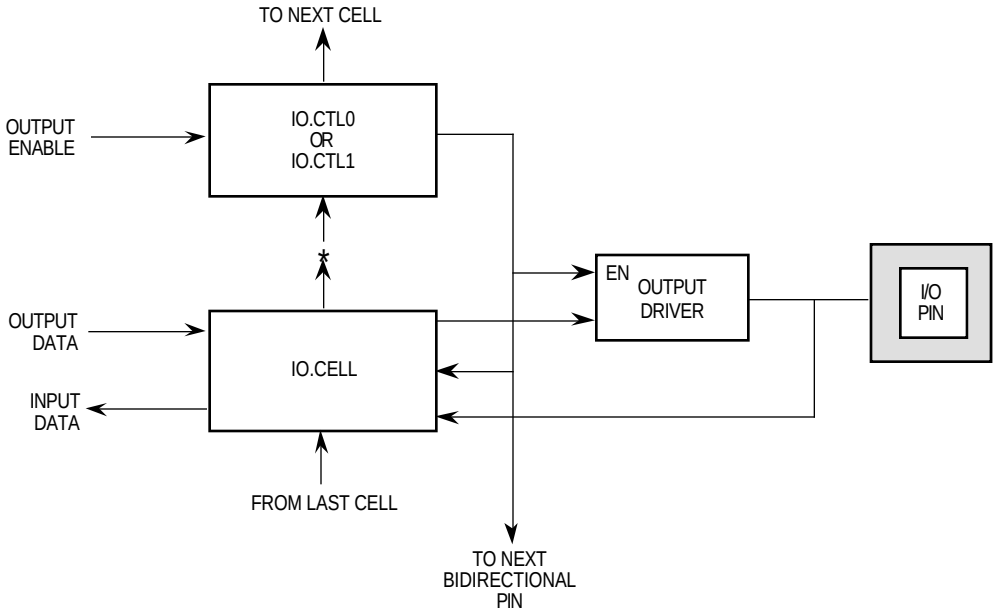


Figure 9-7. Bidirectional Data Cell (IO.Cell)



NOTE: More than one IO.Cell could be serially connected and controlled by a single IO.Ctlx cell.

Figure 9-8. General Arrangement for Bidirectional Pins

9.4 INSTRUCTION REGISTER

The MC68349 IEEE 1149.1 implementation includes the three mandatory public instructions (EXTEST, SAMPLE/PRELOAD, and BYPASS), but does not support any of the optional public instructions defined by IEEE 1149.1. One additional public instruction (HI-Z) provides the capability for disabling all device output drivers. The MC68349 includes a 3-bit instruction register without parity, consisting of a shift register with three parallel outputs. Data is transferred from the shift register to the parallel outputs during the update-IR controller state. The three bits are used to decode the four unique instructions listed in Table 9-3.

The parallel output of the instruction register is reset to all ones in the test-logic-reset controller state. Note that this preset state is equivalent to the BYPASS instruction.

Table 9-3. Instructions

Code			Instruction
B2	B1	B0	
0	0	0	EXTEST
0	0	1	SAMPLE/PRELOAD
X	1	X	BYPASS
1	0	0	HI-Z
1	0	1	BYPASS

During the capture-IR controller state, the parallel inputs to the instruction shift register are loaded with the standard 2-bit binary value (01) into the two least significant bits and the loss-of-crystal (LOC) status signal into bit 2. The parallel outputs, however, remain unchanged by this action since an update-IR signal is required to modify them.

The LOC status bit of the instruction register indicates whether an internal clock is detected when operating with a crystal clock source. The LOC bit is clear when a clock is detected and set when it is not. The LOC bit is always clear when an external clock is used. The LOC bit can be used to detect faulty connectivity when a crystal is used to clock the device.

9.4.1 EXTEST (000)

The external test (EXTEST) instruction selects the 141-bit boundary scan register. EXTEST asserts internal reset for the MC68349 system logic to force a predictable benign internal state while performing external boundary scan operations.

By using the TAP, the register is capable of a) scanning user-defined values into the output buffers, b) capturing values presented to input pins, c) controlling the direction of bidirectional pins, and d) controlling the output drive of three-state output pins. For more details on the function and uses of EXTEST, please refer to the IEEE 1149.1 document.

9.4.2 SAMPLE/PRELOAD (001)

The SAMPLE/PRELOAD instruction selects the 141-bit boundary scan register and provides two separate functions. First, it provides a means to obtain a snapshot of system data and control signals. The snapshot occurs on the rising edge of TCK in the capture-DR controller state. The data can be observed by shifting it transparently through the boundary scan register.

NOTE

Since there is no internal synchronization between the IEEE 1149.1 clock (TCK) and the system clock (CLKOUT), the user must provide some form of external synchronization to achieve meaningful results.

The second function of SAMPLE/PRELOAD is to initialize the boundary scan register output bits prior to selection of EXTEST. This initialization ensures that known data will appear on the outputs when entering the EXTEST instruction.

9.4.3 BYPASS (X1X, 101)

The BYPASS instruction selects the single-bit bypass register as shown in Figure 9-9. This creates a shift-register path from TDI to the bypass register and, finally, to TDO, circumventing the 141-bit boundary scan register. This instruction is used to enhance test efficiency when a component other than the MC68349 becomes the device under test.

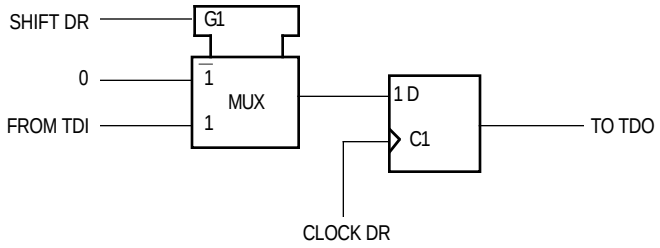


Figure 9-9. Bypass Register

When the bypass register is selected by the current instruction, the shift-register stage is set to a logic zero on the rising edge of TCK in the capture-DR controller state. Therefore, the first bit to be shifted out after selecting the bypass register will always be a logic zero.

9.4.4 HI-Z (100)

The HI-Z instruction is not included in the IEEE 1149.1 standard. It is provided as a manufacturer's optional public instruction to prevent having to backdrive the output pins during circuit-board testing. When HI-Z is invoked, all output drivers, including the two-state drivers, are turned off (i.e., high impedance). The instruction selects the bypass register.

9.5 MC68349 RESTRICTIONS

The control afforded by the output enable signals using the boundary scan register and the EXTEST instruction requires a compatible circuit-board test environment to avoid device-destructive configurations. The user must avoid situations in which the MC68349 output drivers are enabled into actively driven networks. Overdriving the TDO driver when it is active is not recommended.

The MC68349 includes on-chip circuitry to detect the initial application of power to the device.Power-on reset (POR), the output of this circuitry, is used to reset both the system and IEEE 1149.1 logic. The purpose for applying POR to the IEEE 1149.1 circuitry is to avoid the possibility of bus contention during power-on. The time required to complete device power-on is power-supply dependent. However, the IEEE 1149.1 TAP controller remains in the test-logic-reset state while POR is asserted. The TAP controller does not respond to user commands until POR is negated.

The MC68349 features a low-power stop mode that uses a CPU instruction called LPSTOP. The interaction of the IEEE 1149.1 interface with LPSTOP mode is as follows:

1. Leaving the TAP controller test-logic-reset state negates the ability to achieve minimal power consumption, but does not otherwise affect device functionality.
2. The TCK input is not blocked in LPSTOP mode. To consume minimal power, the TCK input should be externally connected to V_{CC} or ground.
3. The TMS and TDI pins include on-chip pullup resistors. In LPSTOP mode, these two pins should remain either unconnected or connected to V_{CC} to achieve minimal power consumption.

9.6 NON-IEEE 1149.1 OPERATION

In non-IEEE 1149.1 operation, there are two constraints. First, the TCK input does not include an internal pullup resistor and should be pulled up externally to preclude mid-level inputs. The second constraint is to ensure that the IEEE 1149.1 test logic is kept transparent to the system logic by forcing the TAP controller into the test-logic-reset state, using either of two methods. During power-on, POR forces the TAP controller into this state. Alternatively, sampling TMS as a logic one for five consecutive TCK rising edges also forces the TAP controller into this state. If TMS either remains unconnected or is connected to V_{CC}, then the TAP controller cannot leave the test-logic-reset state, regardless of the state of TCK.

SECTION 10
APPLICATIONS

This section provides guidelines for using the MC68349. Minimum system-configuration requirements and memory interface information are discussed.

10.1 MINIMUM SYSTEM CONFIGURATION

One of the powerful features of the MC68349 is the small number of external components needed to create an entire system. The information contained in the following paragraphs details a simple high-performance MC68349 system (see Figure 10-1). This system configuration features the following hardware:

- Processor Clock Circuitry
- Reset Circuitry
- Read-Only Memory (ROM) Interface
- Serial Interface

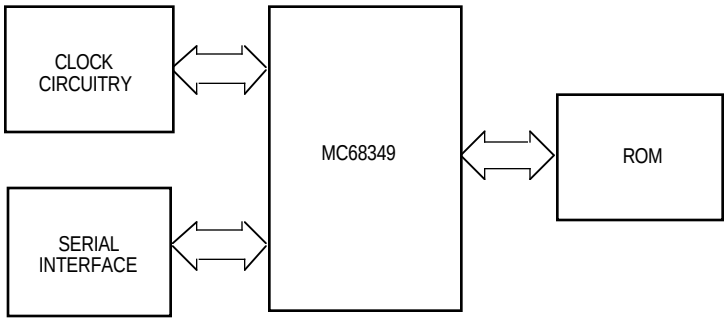


Figure 10-1. Minimum System Configuration Block Diagram

10.1.1 Processor Clock Circuitry

The MC68349 has an on-chip clock synthesizer that can operate from an on-chip phase-locked loop (PLL) and a voltage-controlled oscillator (VCO). The clock synthesizer uses an external crystal connected between the EXTAL and XTAL pins as a reference frequency source. Figure 10-2 shows a typical circuit using an inexpensive 32.768-kHz watch crystal. The 20-M resistor connected between the EXTAL and XTAL pins provides biasing for a faster oscillator startup time. The crystal manufacturer's documentation should be consulted for specific recommendations on external component values.

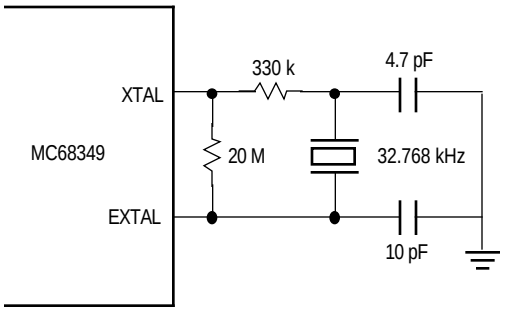


Figure 10-2. Sample Crystal Circuit

The circuit shown in Figure 10-3 is the typical circuit recommended by Statek Corporation, for 32.768 kHz crystal, part number CX-IV. It is recommended to start with these values, but parameter values may need to be adjusted to compensate for variables in layout.

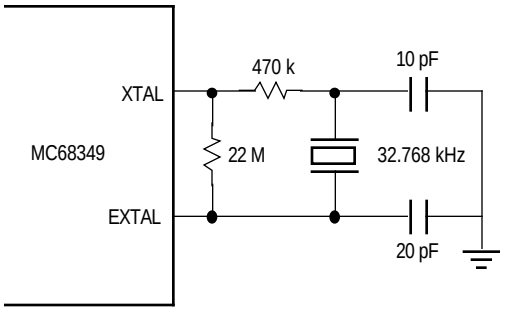
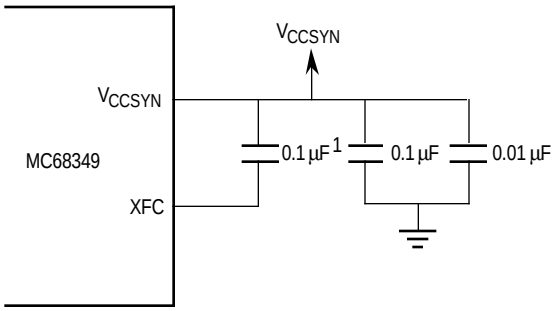


Figure 10-3. Statek Corporation Crystal Circuit

A separate power pin (V_{CCSYN}) is used to allow the clock circuits to operate with the rest of the device powered down and to provide increased noise immunity for the clock circuits. The source for V_{CCSYN} should be a quiet power supply, and external bypass capacitors (see Figure 10-4) should be placed as close as possible to the V_{CCSYN} pin to ensure a stable operating frequency.

Additionally, the PLL requires that an external low-leakage filter capacitor, typically in the range of 0.01 to 0.1 μF, be connected between the XFC and V_{CCSYN} pins. The XFC capacitor should provide 50-MΩ insulation but should not be electrolytic. For external clock mode without PLL, the XFC pin can be left open. Smaller values of the external filter capacitor provide a faster response time for the PLL, and larger values provide greater frequency stability. Figure 10-4 depicts examples of both an external filter capacitor and bypass capacitors for V_{CCSYN}.



NOTE 1: Must be a low leakage capacitor.

Figure 10-4. XFC and V_{CCSYN} Capacitor Connections

10.1.2 Reset Circuitry

A power-on reset (POR) is generated by the SIM49 module when it detects a positive going V_{CC} transition—the V_{CC} threshold is typically in the range 2.0–2.7V, and varies depending on processing and environmental variables. Hysterisis is included in the reset circuit to prevent reassertion for a monotonically increasing V_{CC} voltage; however, excessively long V_{CC} rise times (>100 ms) may allow the reset logic to release $\overline{\text{RESET}}$ before V_{CC} has stabilized. The reset thresholds provided in the SIM49 should not be relied upon to monitor V_{CC}, since internal logic may fail at voltages between spec V_{CCmin} and the reset trigger threshold. An external low voltage monitor circuit, such as the MC34064, should be used instead.

When the MC68349 is used in crystal clock mode, the simplest external reset logic consists of simply a 1K pullup resistor from $\overline{\text{RESET}}$ to V_{CC}. This solution relies on a monotonically increasing V_{CC} that has a rise time on the order of 100ms or less - the actual allowable rise time is dependent on the startup time of the 32.768kHz oscillator circuit. As noted above, this does not provide rigorous V_{CC} monitoring, and may be susceptible to sags or glitches in the V_{CC} supply voltage.

In external clock mode, either with or without the PLL, the POR time delay of 328 * T_{clkin} does not provide adequate time for V_{CC} to stabilize before allowing reset to negate. Applications using these two clocking modes should include an external reset circuit which generates an appropriate delay for the power source being used, as well as a voltage monitor if needed.

10.1.3 SRAM Interface

Although the MC68349 contains internal static random access memory (SRAM), an optional external SRAM interface may be created using the programmable chip selects. External circuitry to decode address information and circuitry to return data and size acknowledge ($\overline{\text{DSACKx}}$) is not required. However, external logic is required to provide write enables for the high and low data bytes.

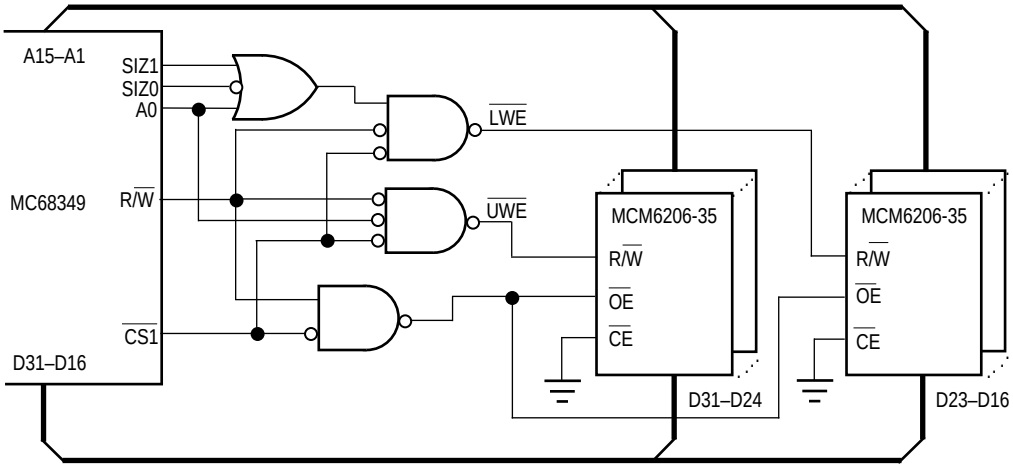


Figure 10-5. SRAM Interface

The SRAM interface shown in Figure 10-5 is a two-clock interface at 16.78-MHz operating frequency. The MCM6206C-35 memories provide an access time of 15 ns when the chip enable ($\overline{E}$) input is low. If buffers are required to reduce signal loading or if slower and less expensive memories are desired, a three-clock cycle can be used. In the circuit shown in Figure 10-5, additional memories can be used provided the MC68349 specification for load capacitance on the chip-select ($\overline{CSx}$) signal is not exceeded. (Address buffers may be needed, however.)

10.1.4 ROM Interface

Using the programmable chip selects creates a very straightforward ROM interface. As shown in Figure 10-6, no external circuitry is needed. Care must be used, however, not to overload the address bus. Address buffers may be required to ensure that the total system input capacitance on the address signals does not exceed the C_L specification. The pullup and pulldown resistors on D30 and D31, respectively, configure the processor to boot from a 16-bit ROM. An 8- or 32-bit boot ROM can be implemented in a similar manner (refer to **Section 4 System Integration Module** for D31 and D30 setup information).

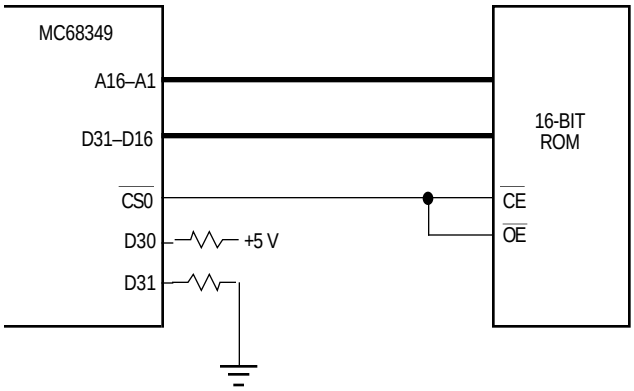


Figure 10-6. ROM Interface

10.1.5 Serial Interface

The necessary circuitry to create an RS-232 interface with the MC68349 includes an external crystal and an RS-232 receiver/driver (see Figure 10-7). The resistor and capacitor values shown are typical; the crystal manufacturer's documentation should be consulted for specific recommendations on external component values. The circuit shown does not include modem support (ready-to-send (RTS) and clear-to-send (CTS) are not shown); however, these signals can be connected to the receiver/driver and to the connector in a similar manner as the connections for TxD and RxD.

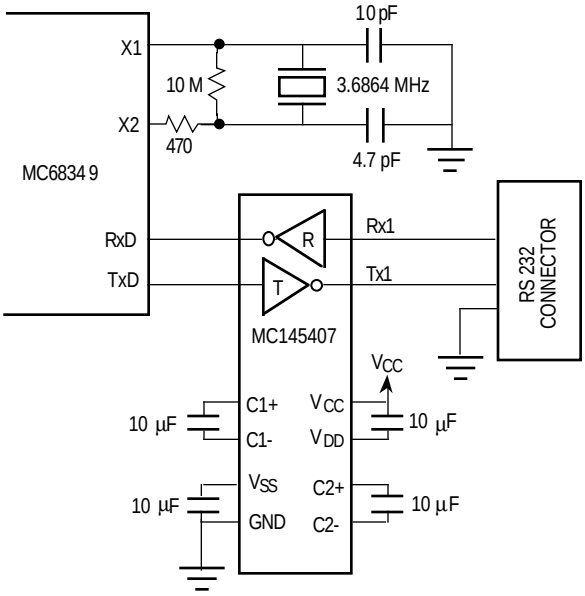


Figure 10-7. Serial Interface

10.2 MEMORY INTERFACE INFORMATION

The following paragraphs contain information on performing access time calculations, calculating frequency-adjusted outputs, and interfacing an 8-bit device to 16-bit memory using the DMA channel single-address mode.

10.2.1 Access Time Calculations

The two time paths that are critical in an MC68349 application using the $\overline{CSx}$ signals are shown in Figure 10-8. The first path is the time from address valid to when data must be available to the processor; the second path is the time from $\overline{CSx}$ asserted to when data must be available to the processor.

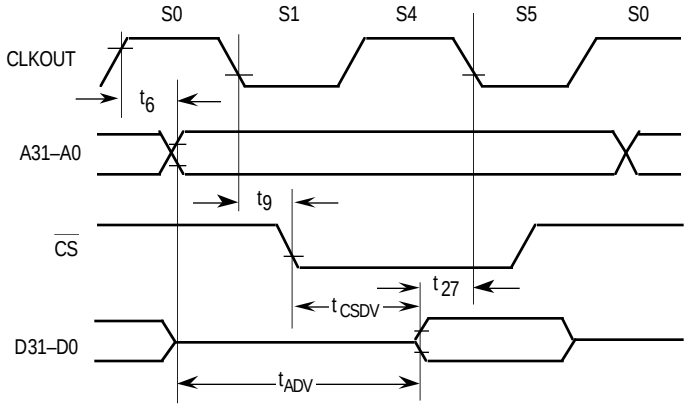


Figure 10-8. Access Time Computation Diagram

As shown in the diagram, an equation for the address access time, t_{ADV}, can be developed as follows:

$$t_{ADV} = t_{cyc}(N_c - 0.5) - t_{s9} - t_{s27}$$

where:

t_{cyc} = system CLKOUT period

N_c = number of clocks per bus cycle

t_{s6} = CLKOUT high to address valid

t_{s27} = data-in valid to CLKOUT low setup

An equation for the chip select access time, t_{CSDV}, can be developed as follows:

$$t_{CSDV} = t_{cyc}(N_c - 1) - t_{s9} - t_{s27}$$

where:

t_{cyc} = system clock period

N_c = number of clocks per access

t_{s9} = CLKOUT low to $\overline{CSx}$ asserted

t_{s27} = data-in valid to CLKOUT low setup

Using these equations, the memory access times at 16.78 MHz and 25.16 MHz are shown in Tables 10-1 and 10-2, respectively. See **Section 11 Electrical Characteristics** for more timing information.

Table 10-1. Memory Access Times at 16.78 MHz

Access Time (ns)	N = 2	N = 3	N = 4	N = 5	N = 6	N = 7	N = 8	N = 9
t_{ADV}	54	114	173	233	292	352	412	471
t_{CSDV}	24	84	143	203	263	322	382	441

Table 10-2. Memory Access Times at 25.16 MHz

Access Time (ns)	N = 2	N = 3	N = 4	N = 5	N = 6	N = 7	N = 8	N = 9
t_{ADV}	34	74	114	153	193	233	273	312
t_{CSDV}	14	54	94	133	173	213	253	292

The values can be used to determine how many clock cycles an access will take, given the access time of the memory devices and any delays through buffers or external logic that may be needed.

10.2.2 Calculating Frequency-Adjusted Output

The general relationship between CLKOUT and most input and output signals is shown in Figure 10-9. Most outputs transition off of a falling edge of CLKOUT, but the same principle applies to those outputs that transition off of a rising edge.

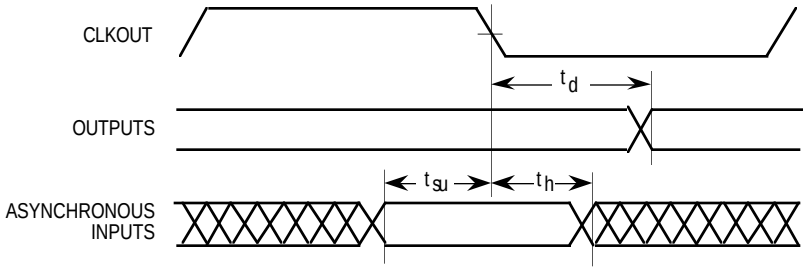


Figure 10-9. Signal Relationships to CLKOUT

For outputs that are referenced to a clock edge, the propagation delay (t_d) does not change as the frequency changes. For instance, specification 6 in the electrical characteristics, shown in **Section 11 Electrical Characteristics**, shows that address, function code, and size information is valid 3 to 30 ns after the rising edge of S0 for a 16.78-MHz device. This specification does not change even if the device frequency is less than 16.78 MHz. Additionally, the relationship between the asynchronous inputs and the clock edge, as shown in Figure 10-9, does not change as frequency changes.

A second type of specification indicates the minimum amount of time a signal will be asserted. This type of specification is illustrated in Figure 10-10.

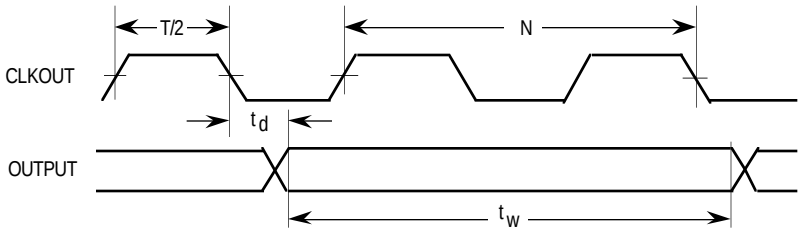


Figure 10-10. Signal Width Specifications

The method for calculating a frequency-adjusted t_W is as follows:

$$t_W' = t_W + N\,(T_f'/2 - T_f/2) + (T_f'/2 - t_d)$$

where:

t_W' = the frequency-adjusted signal width

t_W = the signal width at rated maximum frequency

N = the number of full one-half clock periods in t_W

$T_f'/2$ = one-half the new clock period

$T_f/2$ = one-half the clock period at full speed

t_d = the propagation time from the clock edge

The following calculation uses a 16.78-MHz part, specification 14, $\overline{AS}$ width asserted, at 12.5 MHz as an example:

t_W = 100 ns

N = 3

$T_f'/2$ = 80/2 = 40 ns

$T_f/2$ = 60/2 = 30 ns

t_d = 30 ns maximum

therefore:

$$t_W' = 100 + 3(40 - 30) + (40 - 30) = 140 \text{ ns}$$

The third type of specification used is a skew between two outputs (see Figure 10-11).

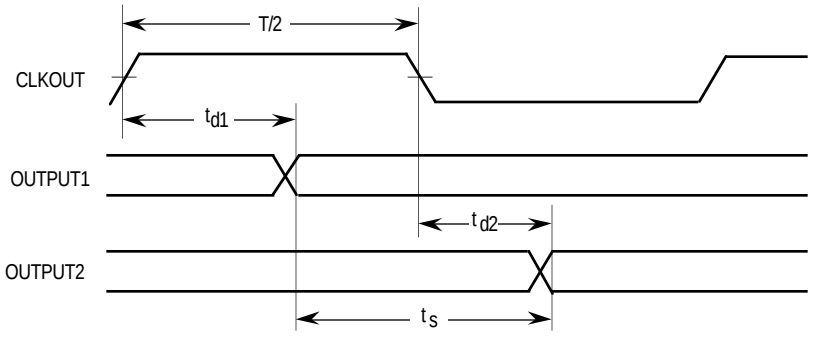


Figure 10-11. Skew between Two Outputs

The method for calculating a frequency-adjusted t_S is as follows:

$$t_S' = t_S + N\,(T_f'/2 - T_f/2) + (T_f'/2 - t_{d1})$$

where:

t_S' = the frequency-adjusted skew

t_S = the skew at full speed

N = the number of full one-half clock periods in t_S, if any

$T_f'/2$ = one-half the new clock period

$T_f/2$ = one-half the clock period at full speed

t_{d1} = the propagation time for the first output from the clock edge

The following calculation uses a 16.78-MHz port, specification 21, R/$\overline{W}$ high to $\overline{AS}$ asserted, at 8 MHz as an example:

t_S = 15 ns minimum

N = 0

$T_f'/2$ = 125/2 = 62.5 ns

$T_f/2$ = 60/2 = 30 ns

t_{d1} = 30 ns maximum

therefore:

$$t_S' = 15 + 0(62.5 - 30) + (62.5 - 30) = 47.5 \text{ ns minimum}$$

In this manner, new specifications for lower frequencies can be derived for an MC68349.

10.2.3 Interfacing an 8-Bit Device to 16-Bit Memory Using Single-Address DMA Mode

One of the requirements of single-address mode is that the source and destination must be the same port size. However, the MC68349 can perform direct memory accesses in single-address mode between an 8-bit device and 16-/32-bit memory or between a 16-bit device and 32-bit memory. Figure 10-12 shows the external logic required for interfacing an 8-bit port to 16-bit memory.

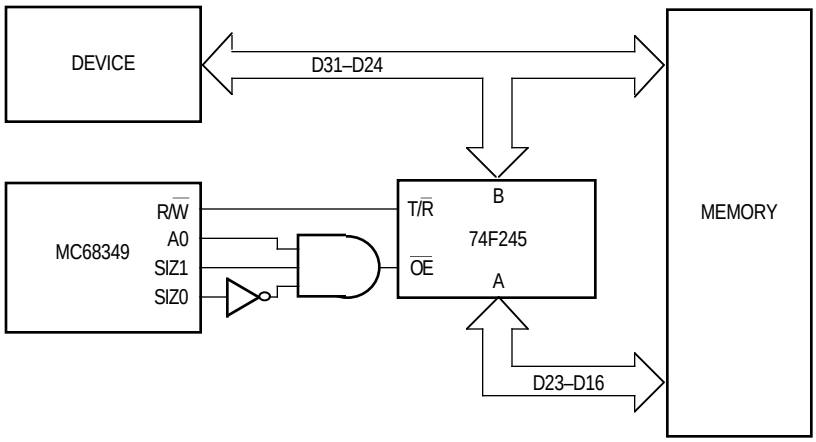


Figure 10-12. Circuitry for Interfacing 8-Bit Device to 16-Bit Memory in Single-Address DMA Mode

During even-byte accesses, the data is transferred directly on D31–D24. However, during odd-byte accesses, the data must be routed on D31–D24 for the 8-bit device and on D23–D16 for the 16-bit memory.

10.3 POWER CONSUMPTION CONSIDERATIONS

The MC68349 can be designed into low-power applications that involve high-performance processing capability (32-bits), high functional density, small size, portable capability, and battery operation.

The MC68349 fits into the following types of applications:

- Personal Intelligent Communicators
- Personal Digital Assistants
- "Palmtop" Computers
  - — Stylus Input
  - — Voice Input
  - — Image Input
- Transaction Tracking
  - — Car Rental
  - — Cargo
  - — Courier
  - — Handheld
- Bar Code Scanners
- Telephony
  - — Cordless Phones
  - — Cellular Phones
- Defense Industry
  - — Guidance Systems
  - — Tracking Systems
- Data Entry
- Instruments
- Handheld Games

The 3.3 V operation provides the following user advantages:

Advantage	Benefit
Lower Supply Voltage	Fewer Batteries
Fewer Batteries	Less Weight Smaller Size
Lower Current Drain	Extended Battery Life
Less Heat Generated	No Fan No Fan Noise
Less EMF Radiation	Easier FCC Certification Less Crosstalk Closer PCB Traces
High Functional Integration	All-In-One 3.3 V Part: Processor Peripherals Glue Logic

These advantages result in a much more portable system.

SECTION 11
ELECTRICAL CHARACTERISTICS

This section contains detailed information on power considerations, DC/AC electrical characteristics, and AC timing specifications of the MC68349 and MC68349V. Refer to **Section 12 Ordering Information and Mechanical Data** for specific part numbers corresponding to voltage and frequency availability.

11.1 MAXIMUM RATINGS

Rating	Symbol	Value	Unit
Supply Voltage [1, 2]	V_{CC}	–0.3 to +6.5	V
Input Voltage [1, 2]	V_{in}	–0.3 to +6.5	V
Operating Temperature Range	T_A	0 to 70	°C
Storage Temperature Range	T_{stg}	–55 to +150	°C

NOTES:

1. Permanent damage can occur if maximum ratings are exceeded. Exposure to voltages or currents in excess of recommended values affects device reliability. Device modules may not operate normally while being exposed to electrical extremes.
2. Although sections of the device contain circuitry to protect against damage from high static voltages or electrical fields, take normal precautions to avoid exposure to voltages higher than maximum-rated voltages.

This device contains protective circuitry against damage due to high static voltages or electrical fields; however, it is advised that normal precautions be taken to avoid application of any voltages higher than maximum-rated voltages to this high-impedance circuit. Reliability of operation is enhanced if unused inputs are tied to an appropriate logic voltage level (e.g., either GND or V_{CC}).

The following ratings define a range of conditions in which the device will operate without being damaged. However, sections of the device may not operate normally while being exposed to the electrical extremes.

11.2 THERMAL CHARACTERISTICS

Characteristic	Symbol	Value	Unit
Thermal Resistance—Junction to Case Plastic 160-Pin QFP	θ_{JC}	TBD	°C/W
Thermal Resistance—Junction to Ambient Plastic 160-Pin QFP	θ_{JA}	40*	°C/W

* Estimated

11.3 POWER CONSIDERATIONS

The average chip-junction temperature, T_J, in °C can be obtained from:

$$T_J = T_A + (P_D \bullet \theta_{JA}) \qquad (1)$$

where:

T_A	=	Ambient Temperature, °C
θ_{JA}	=	Package Thermal Resistance, Junction-to-Ambient, °C/W
P_D	=	$P_{INT} + P_{I/O}$
P_{INT}	=	$I_{CC} \times V_{CC}$, Watts—Chip Internal Power
$P_{I/O}$	=	Power Dissipation on Input and Output Pins—User Determined

For most applications, $P_{I/O} < P_{INT}$ and can be neglected.

An approximate relationship between P_D and T_J (if $P_{I/O}$ is neglected) is:

$$P_D = K \div (T_J + 273°C)$$

Solving Equations (1) and (2) for K gives:

$$K = P_D \bullet (T_A + 273°C) + \theta_{JA} \bullet P_D^2$$

where K is a constant pertaining to the particular part. K can be determined from equation (3) by measuring P_D (at thermal equilibrium) for a known T_A. Using this value of K, the values of P_D and T_J can be obtained by solving Equations (1) and (2) iteratively for any value of T_A.

11.4 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.

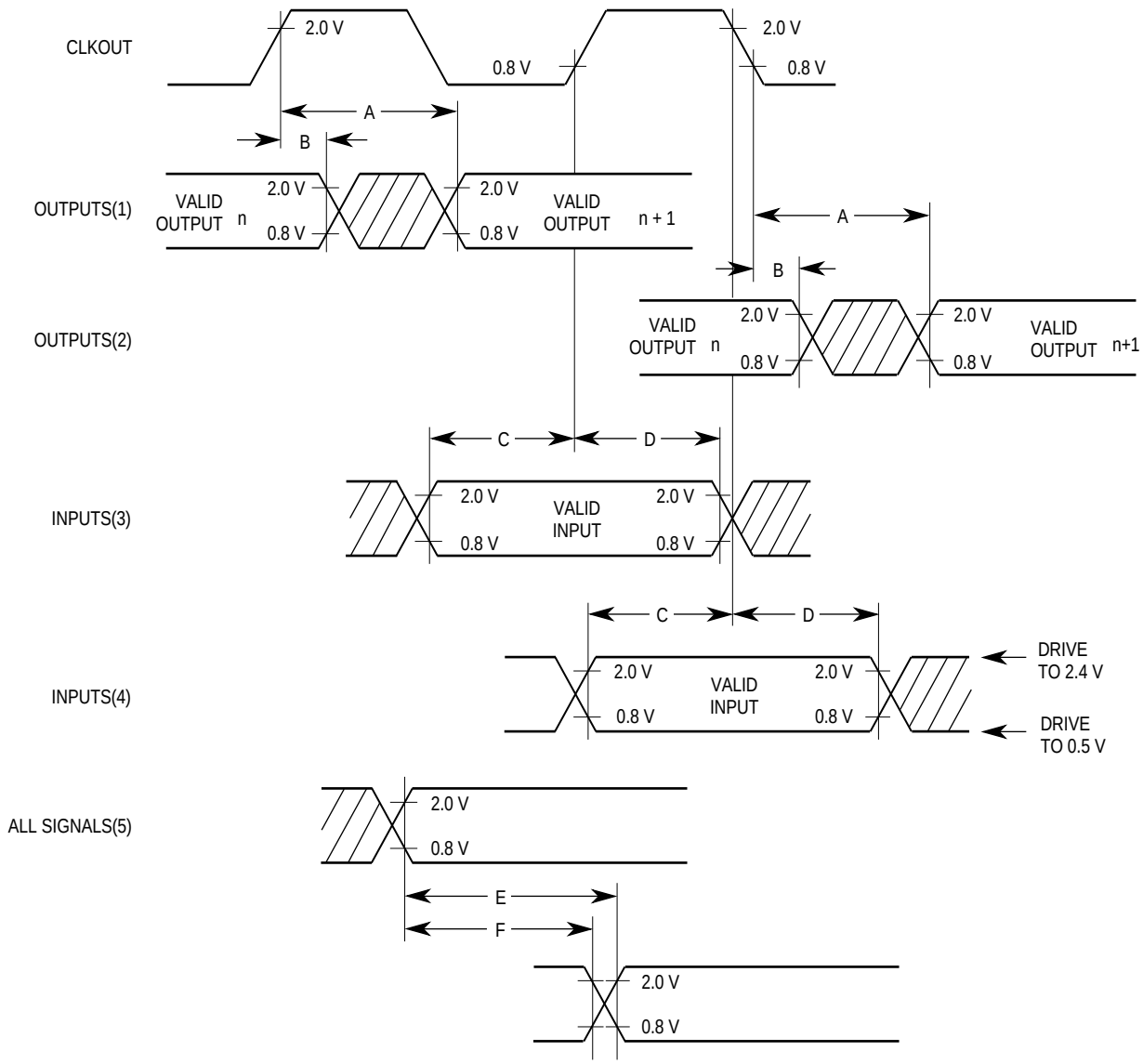
The measurement of the AC specifications is defined by the waveforms shown in Figure 11-1. To test the parameters guaranteed by Motorola, inputs must be driven to the voltage levels specified in the figure. Outputs are specified with minimum and/or maximum limits, as appropriate, and are measured as shown. Inputs are specified with minimum setup and hold times and are measured as shown. Finally, the measurement for signal-to-signal specifications are shown.

Note that the testing levels used to verify conformance to the AC specifications do not affect the guaranteed DC operation of the device as specified in the DC electrical characteristics.

The MC68349V low voltage parts can operate up to 16.78 MHz with a 3.3 V $\pm$ 0.3 V supply. Separate part numbers are used to distinguish the operation of the parts according to the supply voltage. Refer to **Section 12 Ordering Information and Mechanical Data** for the part numbering schemes. MC68349 is used throughout this section to refer to the 16.78- or 25.16-MHz parts at 5.0 V $\pm$ 5%. MC68349V is used throughout this section to refer to the 16.78-MHz parts at 3.3 V $\pm$ 0.3 V.

NOTE

The electrical specifications in this section for the MC68349 and MC68349V are preliminary.



NOTES:
1. This output timing is applicable to all parameters specified relative to the rising edge of the clock.
2. This output timing is applicable to all parameters specified relative to the falling edge of the clock.
3. This input timing is applicable to all parameters specified relative to the rising edge of the clock.
4. This input timing is applicable to all parameters specified relative to the falling edge of the clock.
5. This timing is applicable to all parameters specified relative to the assertion/negation of another signal.

LEGEND:
A. Maximum output delay specification.
B. Minimum output hold time.
C. Minimum input setup time specification.
D. Minimum input hold time specification.
E. Signal valid to signal valid specification (maximum or minimum).
F. Signal valid to signal invalid specification (maximum or minimum).

Figure 11-1. Drive Levels and Test Points for AC Specifications

11.5 DC ELECTRICAL SPECIFICATIONS (See notes (a), (b), and (c) corresponding to part operation, GND = 0 Vdc, TA = 0 to 70°C; see numbered notes)

Characteristic	Symbol	Min	Max	Unit
Input High Voltage (except clock)	V_{IH}	2.0	V_{CC}	V
Input Low Voltage	V_{IL}	GND	0.8	V
Clock Input High Voltage	V_{IHC}	0.7∗(V_{CC})	V_{CC}+0.3	V
Undershoot	—	—	–0.8	V
Input Leakage Current (All Input Only Pins) V_{in} = V_{CC} or GND	I_{in}	–2.5	2.5	µA
Hi-Z (Off-State) Leakage Current (All Noncrystal Outputs and I/O Pins) V_{in} = 0.5/2.4 V[1]	I_{OZ}	–20	20	µA
Signal Low Input Current TMS, TDI V_{IL} = 0.8 V	I_L	–0.015	0.2	mA
Signal High Input Current TMS, TDI V_{IH} = 2.0 V	I_H	–0.015	0.2	mA
Output High Voltage[1, 2] I_{OH} = –0.8 mA, V_{CC} = 4.75 V All Noncrystal Outputs except $\overline{HALT}$, $\overline{RESET}$, $\overline{DONE2}$, $\overline{DONE1}$	V_{OH}	2.4	—	V
Output Low Voltage[1] I_{OL} = 2.0 mA CLKOUT, FREEZE, $\overline{IPIPE0}$, $\overline{PIPE1}$, $\overline{IFETCH}$ I_{OL} = 3.2 mA A23–A0, D31–D0, FC3–FC0, SIZ1, SIZ0 I_{OL} = 5.3 mA All Other Output Only and Group 2 I/O Pins I_{OL} = 15.3 mA $\overline{HALT}$, $\overline{RESET}$	V_{OL}	—	 0.5 0.5 0.5 0.5	V
Total Supply Current at 5 V +5% @ 16.78 MHz RUN[3] LPSTOP (VCO Off)	 I_{CC} S_{ICC}	—	 TBD TBD	 mA µA
Power Dissipation at 5 V +5% @ 16.78 MHz[4]	P_D	—	TBD	mW
Total Supply Current at 3.3 V + 0.3 V @ 16.78 MHz RUN[5] LPSTOP (VCO Off)	 I_{CC} S_{ICC}	—	 TBD TBD	 mA µA
Power Dissipation at 3.3 V +0.3 V @ 16.78 MHz[4]	P_D	—	TBD	mW
Input Capacitance[6] All Input-Only Pins All I/O Pins	C_{in}	—	 10 20	pF
Load Capacitance[6]	C_L	—	100	pF

NOTES:

(a) The 16.78-MHz @ 3.3 V ±0.3 V specifications are preliminary and apply to the MC68349V.
(b) The 16.78-MHz @ 5.0 V ±5% specifications are preliminary and apply to the MC68349.
(c) The 25.16 MHz @ 5.0 V ±5% specifications are preliminary and apply to the MC68349.

1. Input-Only Pins: $\overline{BERR}$, $\overline{BG}$, $\overline{BKPT}$, $\overline{BR}$, $\overline{CTSB}$, $\overline{CTSA}$, $\overline{DREQ2}$, $\overline{DREQ1}$, $\overline{DSACK1}$, $\overline{DSACK0}$, EXTAL, RxDB, RxDA, SCLK, TCK, TDI, TMS
   Output-Only Pins: A23–A0, $\overline{AS}$, $\overline{BG}$, CLKOUT, $\overline{DACK2}$, $\overline{DACK1}$, $\overline{DS}$, FC3–FC0, FREEZE, $\overline{IFETCH}$, $\overline{IPIPE0}$, $\overline{IPIPE1}$, $\overline{RMC}$, $\overline{RTSB}$, $\overline{RTSA}$, R/$\overline{W}$, $\overline{RxRDYA}$, SIZ1, SIZ0, TDO, TxDB, TxDA, $\overline{TxRDYA}$
   Input/Output Pins:
   Group 1: D31–D0
   Group 2: A31–A24, $\overline{CS3}$–$\overline{CS0}$, $\overline{DONE2}$, $\overline{DONE1}$, $\overline{IRQ7}$, $\overline{IRQ5}$, $\overline{IRQ3}$, MODCK
   Group 3: $\overline{HALT}$, $\overline{RESET}$
2. V_{OH} specification for $\overline{HALT}$, $\overline{RESET}$, $\overline{DONE2}$, and $\overline{DONE1}$ is not applicable because they are open-drain pins.
3. Supply current measured with system clock frequency of 16.78 MHz @ 5.25 V.
4. Power dissipation measured with a system clock frequency of 16.78 MHz, all modules active.
5. Supply current measured with system clock frequency of 16.78 MHz @ 3.6 V.
6. Capacitance is periodically sampled rather than 100% tested.

11.6 AC ELECTRICAL SPECIFICATIONS CONTROL TIMING (See notes (a), (b), and (c) corresponding to part operation, GND = 0 Vdc, TA = 0 to 70°C; see numbered notes)

Num.	Characteristic	Symbol	3.3 V or 5.0 V 16.78 MHz Min	3.3 V or 5.0 V 16.78 MHz Max	5.0 V 25.16 MHz Min	5.0 V 25.16 MHz Max	Unit
	System Frequency[1]	f_{sys}	dc	16.78	dc	25.16	MHz
	Crystal Frequency	f_{XTAL}	25	50	25	50	kHz
	On-Chip VCO System Frequency	f_{sys}	0.13	16.78	0.13	25.16	MHz
	On-Chip VCO Frequency Range	f_{VCO}	0.1	33.5	0.1	50.3	MHz
	External Clock Operation	f_{sys}	0	16	0	25	MHz
	PLL Start-up Time[2]	t_{rc}	—	20	—	20	ms
	Limp Mode Clock Frequency[3] SYNCR X-bit = 0 SYNCR X-bit = 1	f_{limp}	 — —	 $f_{sys}/2$ f_{sys}	 — —	 $f_{sys}/2$ f_{sys}	kHz
	CLKOUT stability[4]	Δ_{CLK}	–1	+1	–1	+1	%
1[5]	CLKOUT Period in Crystal Mode	t_{cyc}	59.6	—	40	—	ns
1B[6]	External Clock Input Period	t_{EXTcyc}	62.5	—	40	—	ns
1C[7]	External Clock Input Period with PLL	t_{EXTcyc}	62.5	—	40	—	ns
2,3[8]	CLKOUT Pulse Width in Crystal Mode	t_{CW}	28	—	19	—	ns
2B, 3B[9]	CLKOUT Pulse Width in External Mode	t_{EXTCW}	28	—	18	—	ns
2C, 3C[10]	CLKOUT Pulse Width in External w/PLL Mode	t_{EXTCW}	31	—	20	—	ns
4,5	CLKOUT Rise and Fall Times	t_{Crf}	—	5	—	4	ns

NOTES:

(a) The 16.78-MHz @ 3.3 V ±0.3 V specifications are preliminary and apply to the MC68349V.
(b) The 16.78-MHz @ 5.0 V ±5% specifications are preliminary and apply to the MC68349.
(c) The 25.16 MHz @ 5.0 V ±5% specifications are preliminary and apply to the MC68349.

1. All internal registers retain data at 0 Hz.
2. Assumes that a stable V_{CCSYN} is applied, that an external filter capacitor with a value of 0.1 µF is attached to the XFC pin, and that the crystal oscillator is stable. Lock time is measured from power-up to $\overline{\text{RESET}}$ release. This specification also applies to the period required for PLL lock after changing the W and Y frequency control bits in the synthesizer control register (SYNCR) while the PLL is running, and to the period required for the clock to lock after exiting LPSTOP.
3. Determined by the initial control voltage applied to the on-chip VCO. The X-bit in the SYNCR controls a divide-by-two scalar on the system clock output.
4. CLKOUT stability is the average deviation from programmed frequency measured at maximum f_{sys}. Measurement is made with a stable external clock input applied using the PLL.
5. All crystal mode clock specifications are based on using a 32.768-kHz crystal for the input.
6. When using the external clock input mode (MODCK reset value = 0 V), the minimum allowable t_{EXTcyc} period will be reduced when the duty cycle of the signal applied to EXTAL exceeds 5% tolerance. The relationship between external clock input duty cycle and minimum t_{EXTcyc} is expressed:
   Minimum t_{EXTcyc} period = minimum t_{EXTCW} / (50% – external clock input duty cycle tolerance).
   Minimum external clock low and high times are based on a 45% duty cycle.
7. When using the external clock input mode with the PLL (MODCK reset value = 0 V), the external clock input duty cycle can be at minimum 20% to produce a CLKOUT with a 50% duty cycle.
8. For crystal mode operation, the minimum CLKOUT pulse width is based on a 47% duty cycle.

11.6 AC ELECTRICAL SPECIFICATIONS CONTROL TIMING (Continued)

9. For external clock mode operation, the minimum CLKOUT pulse width is based on a 45% duty cycle, with a 50% duty cycle input clock.
10. For external clock w/PLL mode operation, the minimum CLKOUT pulse width is based on a 50% duty cycle.
11. For external clock mode, there is a 10–40 ns skew between the input clock signal and the output CLKOUT signal from the MC68349. Clock skew is measured from the rising edges of the clock signals.
12. For external clock mode w/PLL, there is a max 5 ns skew between the input clock signal and the output CLKOUT signal from the MC68349. Clock skew is measured from the rising edges of the clock signals.

11.7 AC TIMING SPECIFICATIONS (See notes (a), (b), and (c) corresponding to part operation, GND = 0 Vdc, TA = 0 to 70°C; see numbered notes; see Figures 11-2–11-11)

Num.	Characteristic	Symbol	3.3 V or 5.0 V 16.78 MHz Min	3.3 V or 5.0 V 16.78 MHz Max	5.0 V 25.16 MHz Min	5.0 V 25.16 MHz Max	Unit
6	CLKOUT High to Address, FC, SIZx, $\overline{RMC}$ Valid	t_{CHAV}	0	30	0	20	ns
7	CLKOUT High to Address, Data, FC, SIZx, $\overline{RMC}$ High Impedance	t_{CHAZx}	0	60	0	40	ns
8	CLKOUT High to Address, FC, SIZx, $\overline{RMC}$ Invalid	t_{CHAZn}	0	—	0	—	ns
9[9]	CLKOUT Low to $\overline{AS}$, $\overline{DS}$, $\overline{CS}$, $\overline{IFETCH}$, $\overline{IPIPEx}$, $\overline{IACKx}$ Asserted	t_{CLSA}	3	30	3	20	ns
9A[2]	$\overline{AS}$ to $\overline{DS}$ or $\overline{CS}$ Asserted (Read)	t_{STSA}	–15	15	–6	6	ns
11	Address, FC, SIZx, $\overline{RMC}$ Valid to $\overline{AS}$, $\overline{CS}$ (and $\overline{DS}$ Read) Asserted	t_{AVSA}	15	—	10	—	ns
12	CLKOUT Low to $\overline{AS}$, $\overline{DS}$, $\overline{CS}$, $\overline{IFETCH}$, $\overline{IPIPEx}$, $\overline{IACKx}$ Negated	t_{CLSN}	3	30	3	20	ns
13	$\overline{AS}$, $\overline{DS}$, $\overline{CS}$, $\overline{IACKx}$ Negated to Address, FC, SIZ Invalid (Address Hold)	t_{SNAI}	15	—	10	—	ns
14	$\overline{AS}$, $\overline{CS}$ (and $\overline{DS}$ Read) Width Asserted	t_{SWA}	100	—	70	—	ns
14A	$\overline{DS}$ Width Asserted (Write)	t_{SWAW}	45	—	30	—	ns
14B	$\overline{AS}$, $\overline{CS}$, $\overline{IACKx}$ (and $\overline{DS}$ Read) Width Asserted (Fast Termination Cycle)	t_{SWDW}	40	—	30	—	ns
15[3]	$\overline{AS}$, $\overline{DS}$, $\overline{CS}$ Width Negated	t_{SN}	40	—	30	—	ns
16	CLKOUT High to $\overline{AS}$, $\overline{DS}$, R/$\overline{W}$ High Impedance	t_{CHSZ}	—	60	—	40	ns
17	$\overline{AS}$, $\overline{DS}$, $\overline{CS}$ Negated to R/$\overline{W}$ High	t_{SNRN}	15	—	10	—	ns
18	CLKOUT High to R/$\overline{W}$ High	t_{CHRH}	0	30	0	20	ns
20	CLKOUT High to R/$\overline{W}$ Low	t_{CHRL}	0	30	0	20	ns
21[9]	R/$\overline{W}$ High to $\overline{AS}$, $\overline{CS}$ Asserted	t_{RAAA}	15	—	10	—	ns
22	R/$\overline{W}$ Low to $\overline{DS}$ Asserted (Write)	t_{RASA}	70	—	47	—	ns
23	CLKOUT High to Data-Out Valid	t_{CHDO}	—	30	—	20	ns
24	Data-Out Valid to Negating Edge of $\overline{AS}$, $\overline{CS}$, (Fast Termination Write)	t_{DVASN}	15	—	10	—	ns
25	$\overline{DS}$, $\overline{CS}$, Negated to Data-Out Invalid (Data-Out Hold)	t_{SNDOI}	15	—	10	—	ns
26	Data-Out Valid to $\overline{DS}$ Asserted (Write)	t_{DVSA}	15	—	10	—	ns
27	Data-In Valid to CLKOUT Low (Data Setup)	t_{DICL}	5	—	5	—	ns
27A	Late $\overline{BERR}$, $\overline{HALT}$, $\overline{BKPT}$ Asserted to CLKOUT Low (Setup Time)	t_{BELCL}	20	—	10	—	ns
28	$\overline{AS}$, $\overline{DS}$ Negated to $\overline{DSACKx}$, $\overline{BERR}$, $\overline{HALT}$ Negated	t_{SNDN}	0	80	0	50	ns
29[4]	$\overline{DS}$, $\overline{CS}$ Negated to Data-In Invalid (Data-In Hold)	t_{SNDI}	0	—	0	—	ns
29A[4]	$\overline{DS}$, $\overline{CS}$ Negated to Data-In High Impedance	t_{SHDI}	—	60	—	40	ns

11.7 AC TIMING SPECIFICATIONS (Continued)

Num.	Characteristic	Symbol	3.3 V or 5.0 V 16.78 MHz Min	3.3 V or 5.0 V 16.78 MHz Max	5.0 V 25.16 MHz Min	5.0 V 25.16 MHz Max	Unit
30[4]	CLKOUT Low to Data-In Invalid (Fast Termination Hold)	t_{CLDI}	15	—	10	—	ns
30A[4]	CLKOUT Low to Data-In High Impedance	t_{CLDH}	—	90	—	60	ns
31[5]	$\overline{DSACKx}$ Asserted to Data-In Valid	t_{DADI}	—	50	—	32	ns
31A	$\overline{DSACKx}$ Asserted to $\overline{DSACKx}$ Valid (Skew)	t_{DADV}	—	30	—	20	ns
32	$\overline{HALT}$ and $\overline{RESET}$ Input Transition Time	t_{HRrf}	—	200	—	140	ns
33	CLKOUT Low to $\overline{BG}$ Asserted	t_{CLBA}	—	30	—	20	ns
34	CLKOUT Low to $\overline{BG}$ Negated	t_{CLBN}	—	30	—	20	ns
35[6]	$\overline{BR}$ Asserted to $\overline{BG}$ Asserted ($\overline{RMC}$ Not Asserted)	t_{BRAGA}	1	—	1	—	CLKOUT
37	$\overline{BGACK}$ Asserted to $\overline{BG}$ Negated	t_{GAGN}	1	2.5	1	2.5	CLKOUT
39	$\overline{BG}$ Width Negated	t_{GH}	2	—	2	—	CLKOUT
39A	$\overline{BG}$ Width Asserted	t_{GA}	1	—	1	—	CLKOUT
46	R/$\overline{W}$ Width Asserted (Write or Read)	t_{RWA}	150	—	100	—	ns
46A	R/$\overline{W}$ Width Asserted (Fast Termination Write or Read)	t_{RWAS}	90	—	60	—	ns
47A[8]	Asynchronous Input Setup Time	t_{AIST}	8, 5	—	5	—	ns
47B	Asynchronous Input Hold Time	t_{AIHT}	15	—	10	—	ns
48[5,7]	$\overline{DSACKx}$ Asserted to $\overline{BERR}$, $\overline{HALT}$ Asserted	t_{DABA}	—	30	—	20	ns
53	Data-Out Hold from CLKOUT High	t_{DOCH}	0	—	0	—	ns
54	CLKOUT High to Data-Out High Impedance	t_{CHDH}	—	30	—	20	ns
55	R/$\overline{W}$ Asserted to Data Bus Impedance Change	t_{RADC}	40	—	25	—	ns
56	$\overline{RESET}$ Pulse Width (Reset Instruction)	t_{HRPW}	512	—	512	—	CLKOUT
56A	$\overline{RESET}$ Pulse Width (Input from External Device)	t_{RPWI}	590	—	590	—	CLKOUT
57	$\overline{BERR}$ Negated to $\overline{HALT}$ Negated (Rerun)	t_{BNHN}	0	—	0	—	ns
70	CLKOUT Low to Data Bus Driven (Show Cycle)	t_{SCLDD}	0	30	0	20	ns
71	Data Setup Time to CLKOUT Low (Show Cycle)	t_{SCLDS}	15	—	10	—	ns
72	Data Hold from CLKOUT Low (Show Cycle)	t_{SCLDH}	10	—	6	—	ns
80	DSI Input Setup Time	t_{DSISU}	15	—	10	—	ns
81	DSI Input Hold Time	t_{DSIH}	10	—	6	—	ns
82	DSCLK Setup Time	t_{DSCSU}	15	—	10	—	ns
83	DSCLK Hold Time	t_{DSCH}	10	—	6	—	ns
84	DSO Delay Time	t_{DSOD}	—	t_{cyc} + 25	—	t_{cyc} + 16	ns

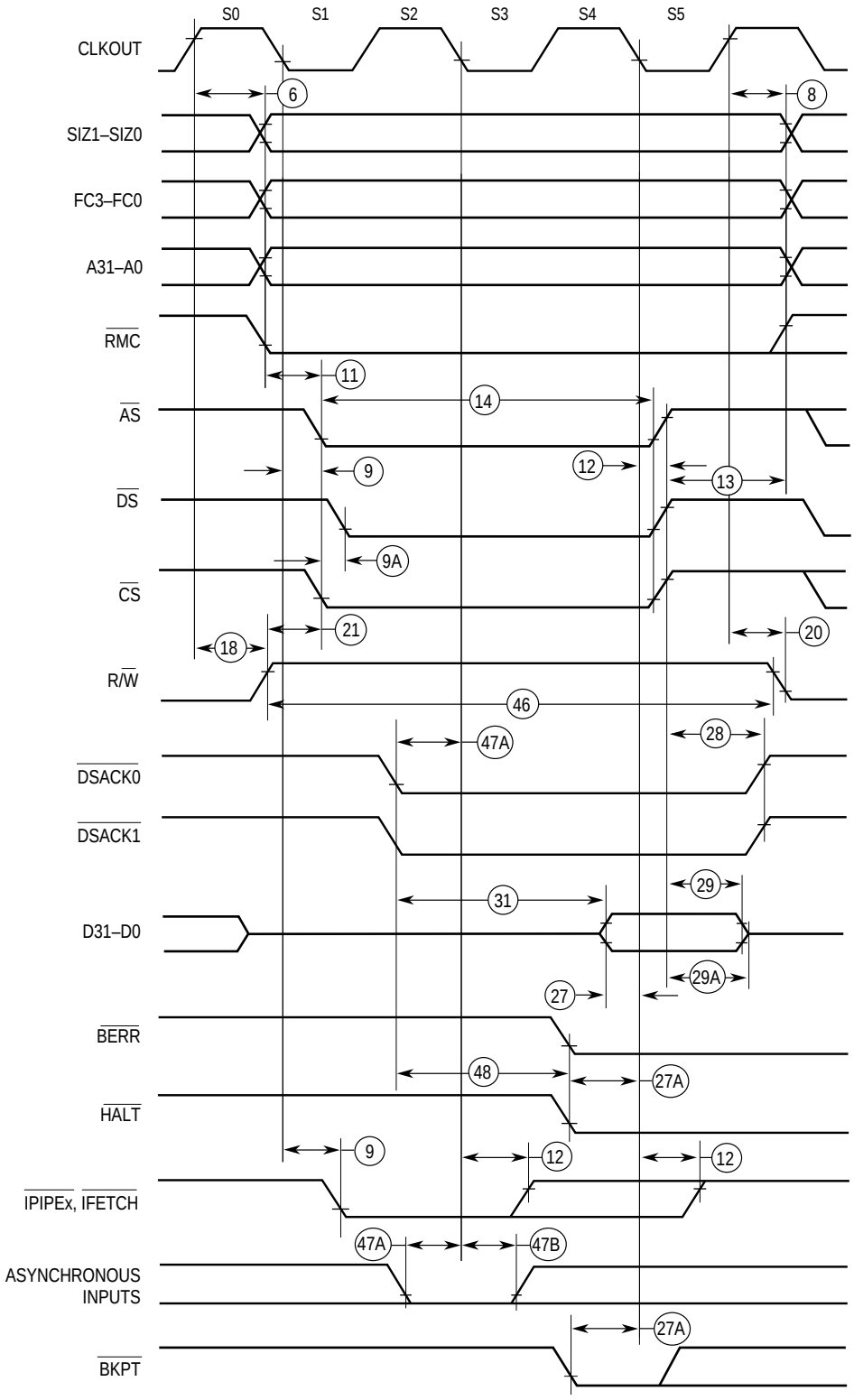
11.7 AC TIMING SPECIFICATIONS (Concluded)

Num.	Characteristic	Symbol	3.3 V or 5.0 V		5.0 V		Unit
			16.78 MHz		25.16 MHz		
			Min	Max	Min	Max	
85	DSCLK Cycle	t_{DSCCYC}	2	—	2	—	CLKOUT
86	CLKOUT High to FREEZE Asserted	t_{FRZA}	0	50	0	35	ns
87	CLKOUT High to FREEZE Negated	t_{FRZN}	0	50	0	35	ns
88	CLKOUT High to $\overline{\text{IFETCH}}$ High Impedance	t_{IFZ}	0	50	0	35	ns
89	CLKOUT High to $\overline{\text{IFETCH}}$ Valid	t_{IF}	0	50	0	35	ns

NOTES:

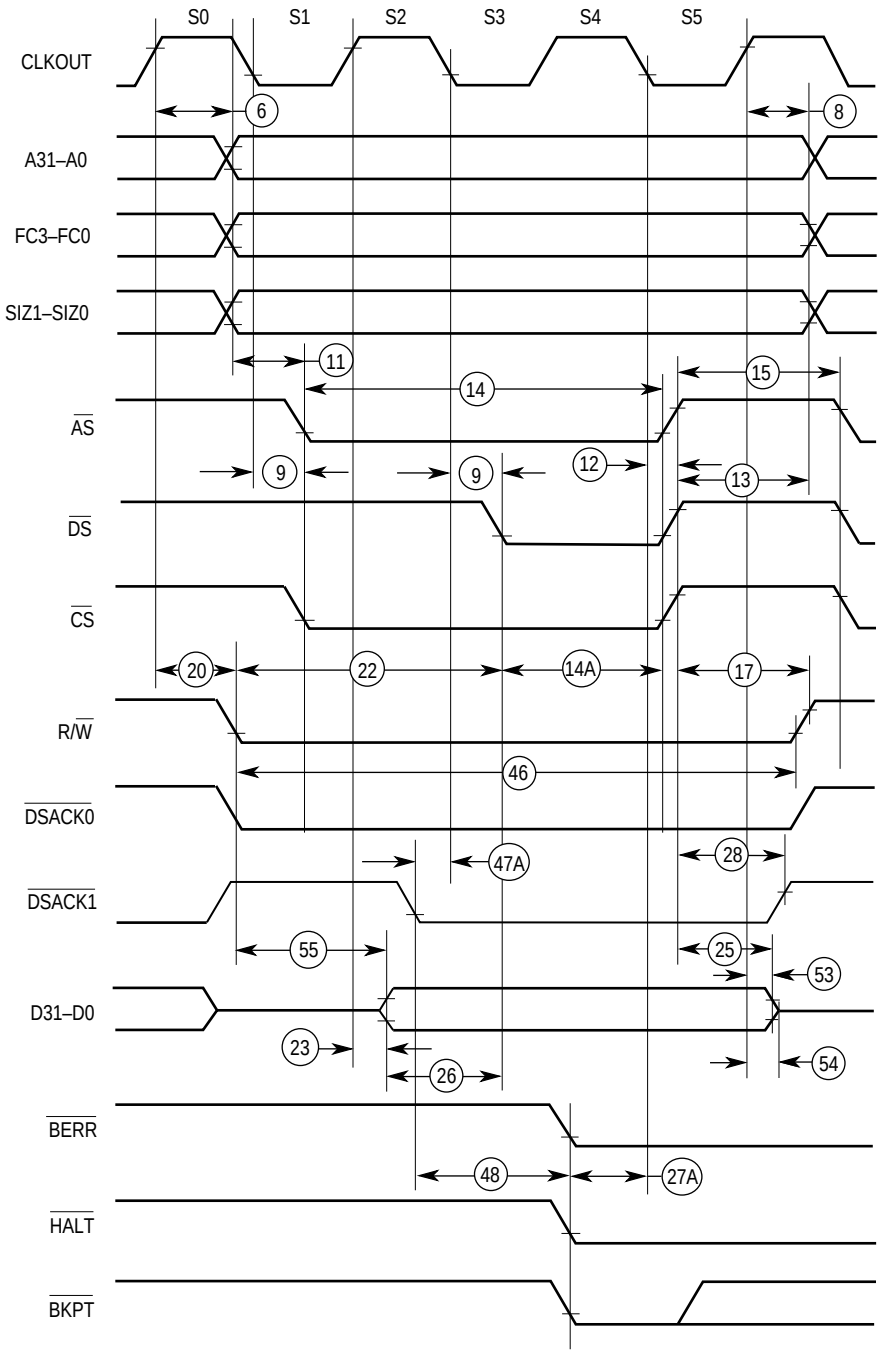
(a) The 16.78-MHz @ 3.3 V ±0.3 V specifications are preliminary and apply to the MC68349V.
(b) The 16.78-MHz @ 5.0 V ±5% specifications are preliminary and apply to the MC68349.
(c) The 25.16 MHz @ 5.0 V ±5% specifications are preliminary and apply to the MC68349.

1. All AC timing is shown with respect to 0.8 V and 2.0 V levels unless otherwise noted.
2. This number can be reduced to 5 ns if strobes have equal loads.
3. If multiple chip selects are used, the $\overline{\text{CS}}$ width negated (#15) applies to the time from the negation of a heavily loaded chip select to the assertion of a lightly loaded chip select.
4. These hold times are specified with respect to $\overline{\text{DS}}$ or $\overline{\text{CS}}$ on asynchronous reads and with respect to CLKOUT on fast termination reads. The user is free to use either hold time for fast termination reads.
5. If the asynchronous setup time (#47) requirements are satisfied, the $\overline{\text{DSACKx}}$ low to data setup time (#31) and $\overline{\text{DSACKx}}$ low to $\overline{\text{BERR}}$ low setup time (#48) can be ignored. The data must only satisfy the data-in to CLKOUT low setup time (#27) for the following clock cycle: $\overline{\text{BERR}}$ must only satisfy the late $\overline{\text{BERR}}$ low to CLKOUT low setup time (#27A) for the following clock cycle.
6. To ensure coherency during every operand transfer, $\overline{\text{BG}}$ will not be asserted in response to $\overline{\text{BR}}$ until after cycles of the current operand transfer are complete and $\overline{\text{RMC}}$ is negated.
7. In the absence of $\overline{\text{DSACKx}}$, $\overline{\text{BERR}}$ is an asynchronous input using the asynchronous setup time (#47).
8. Specification #47A for 16.78 MHz @ 3.3 V ±0.3V will be 8 ns.
9. During interrupt acknowledge cycles up to two wait states may be inserted by the processor between states S0 and S1.



NOTE: All timing is shown with respect to 0.8V and 2.0V levels.

Figure 11-2. Read Cycle Timing Diagram



NOTE: All timing is shown with respect to 0.8-V and 2.0-V levels.

Figure 11-3. Write Cycle Timing Diagram

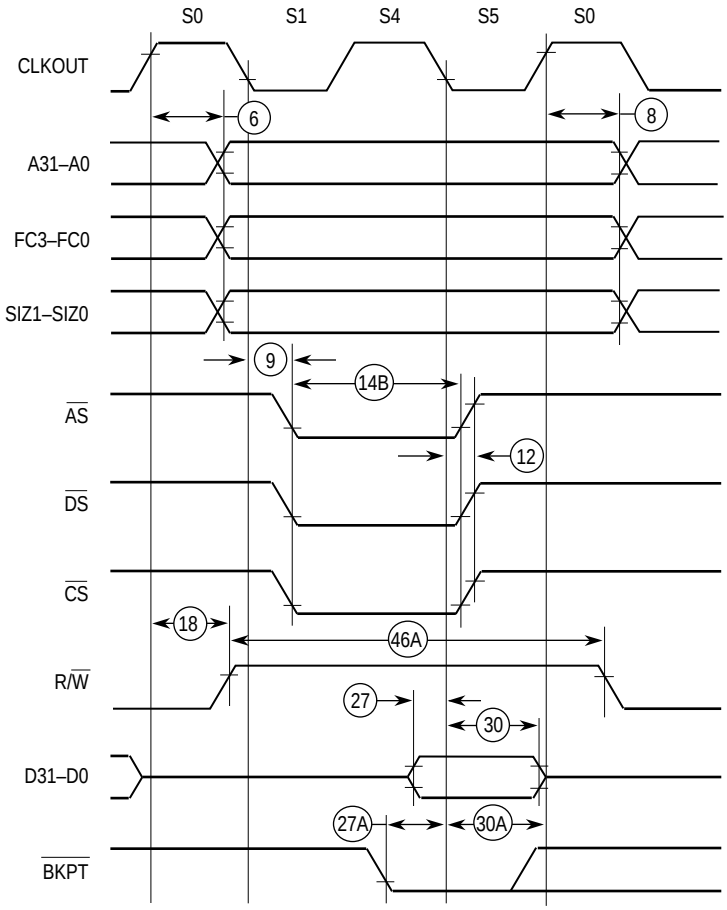


Figure 11-4. Fast Termination Read Cycle Timing Diagram

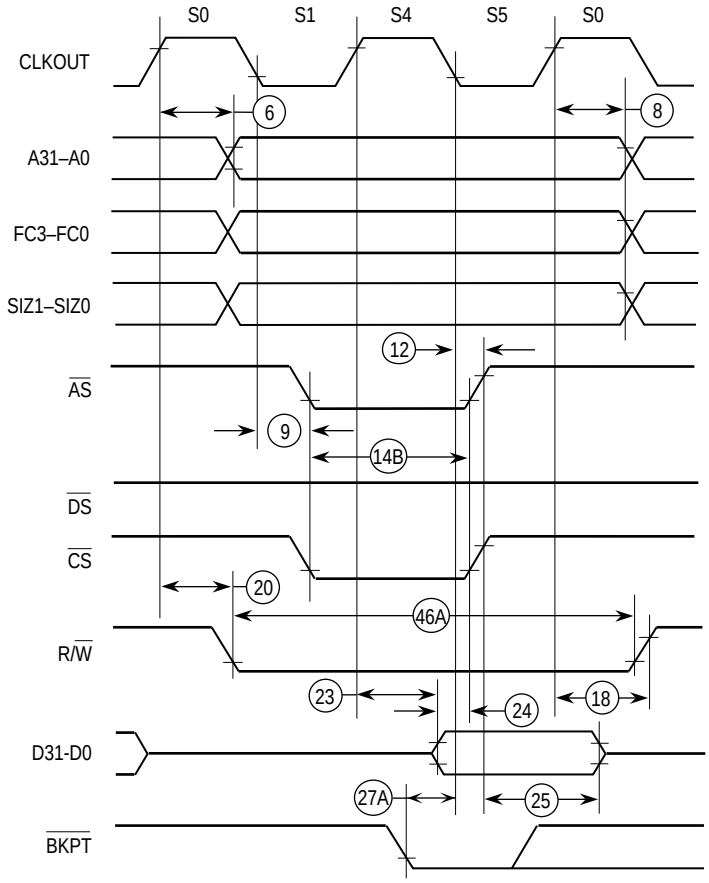


Figure 11-5. Fast Termination Write Cycle Timing Diagram

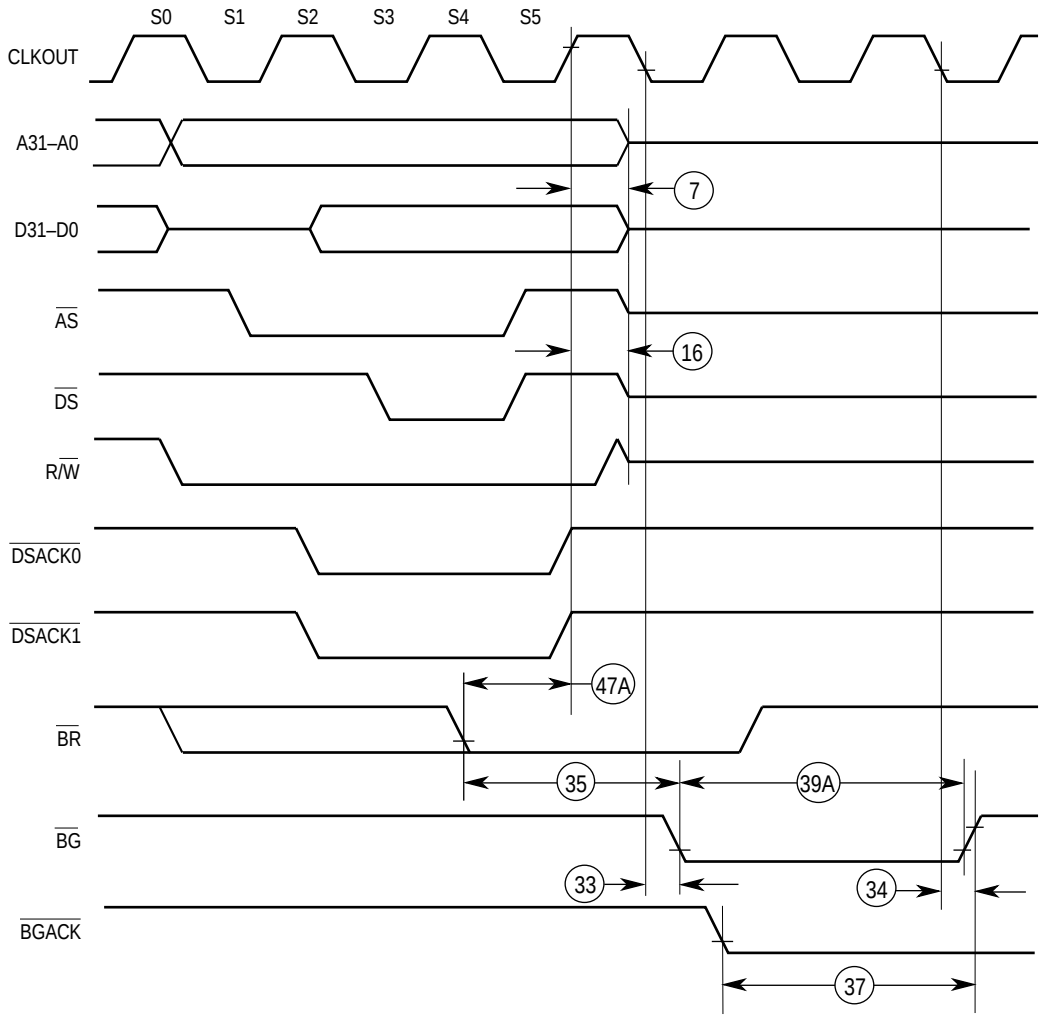


Figure 11-6. Bus Arbitration Timing—Active Bus Case

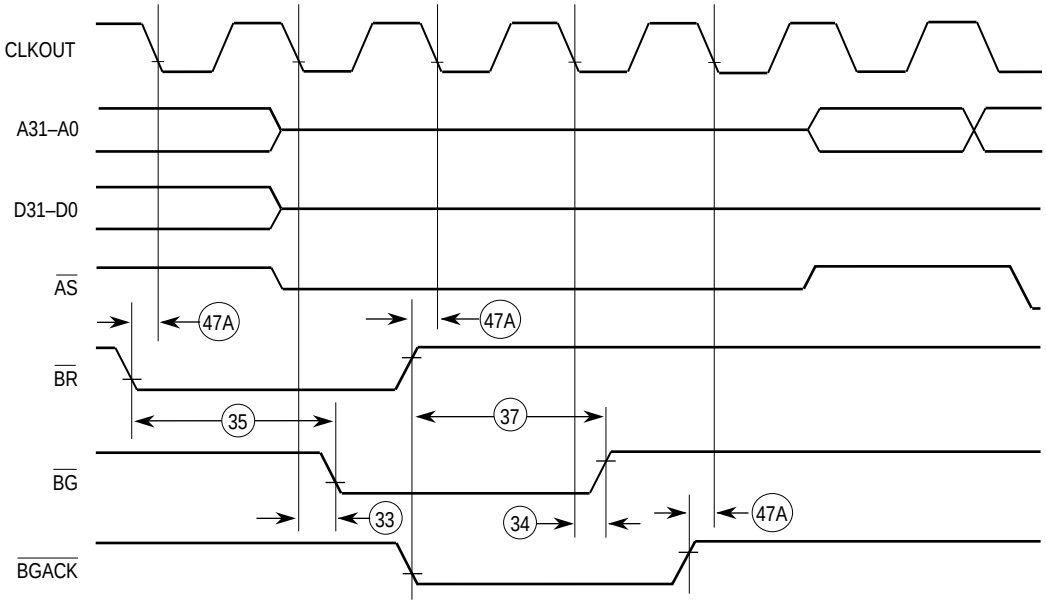


Figure 11-7. Bus Arbitration Timing—Idle Bus Case

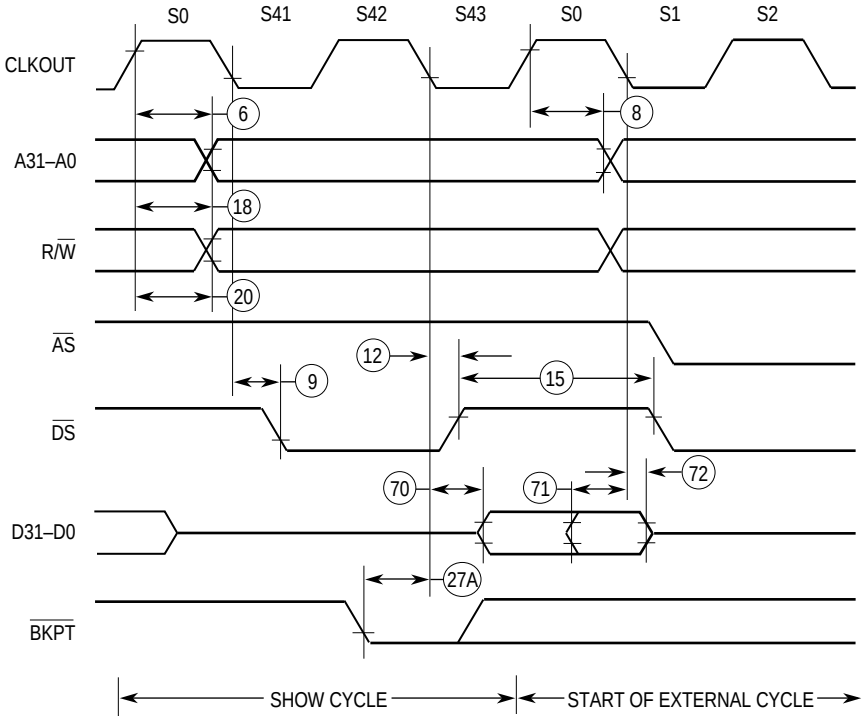
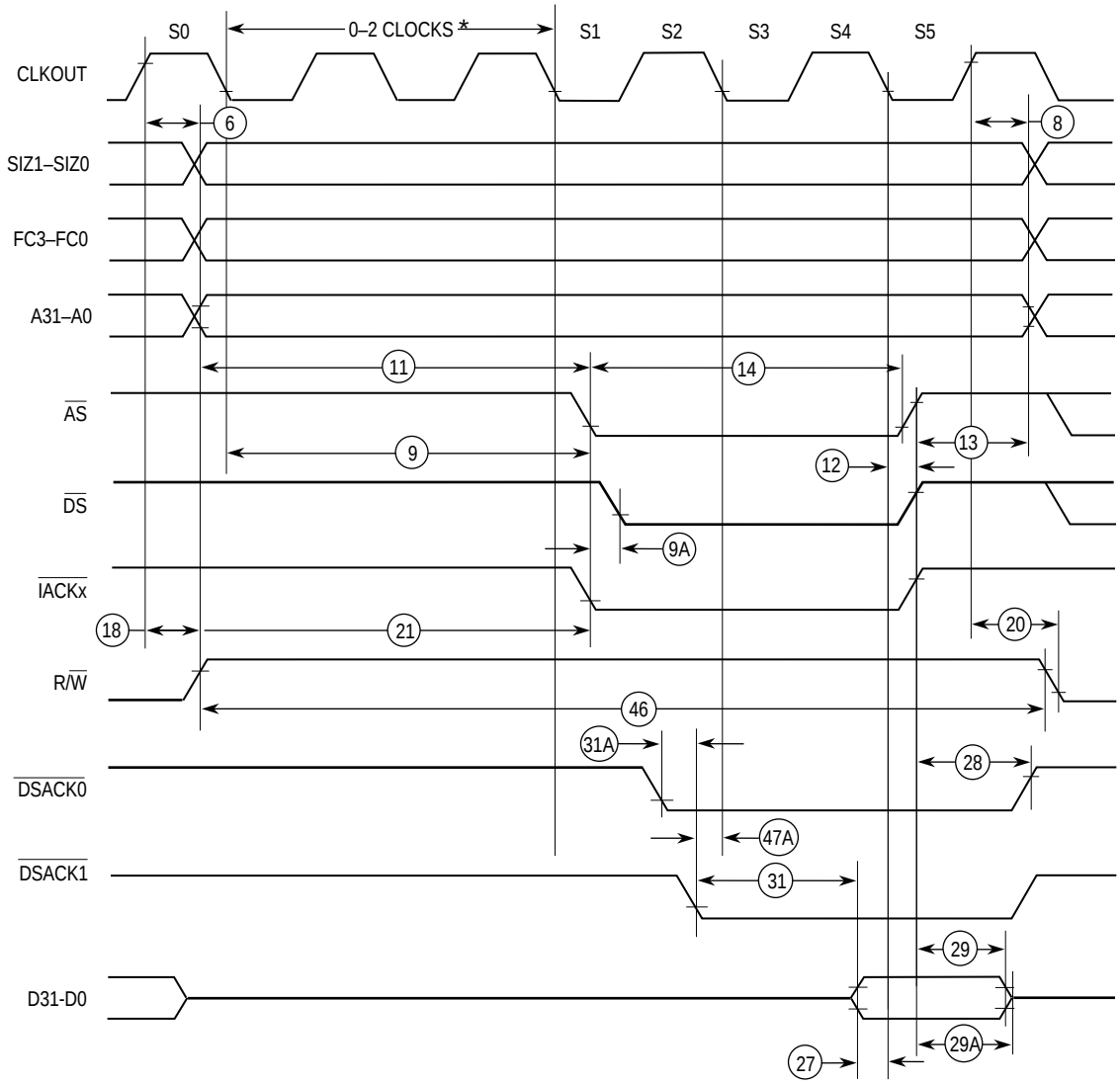


Figure 11-8. Show Cycle Timing Diagram



*Up to two wait states may be inserted by the processor between states S0 and S1.

Figure 11-9. $\overline{\text{IACK}}$ Cycle Timing Diagram

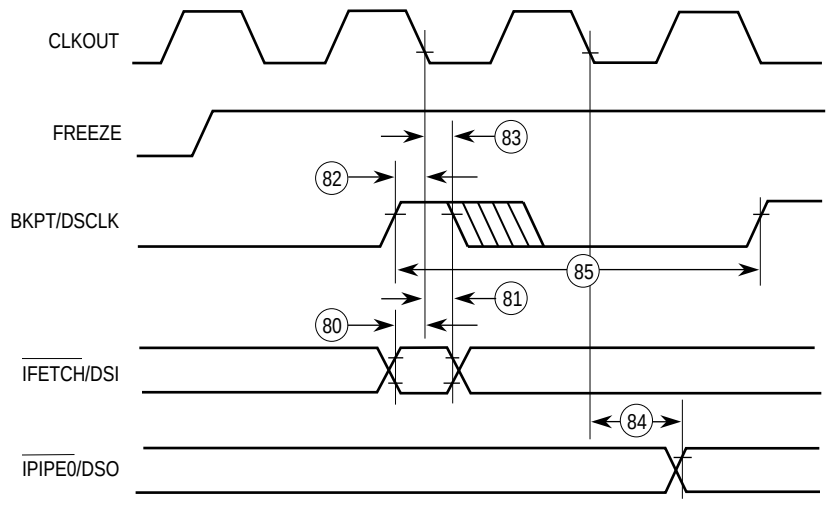


Figure 11-10. Background Debug Mode Serial Port Timing

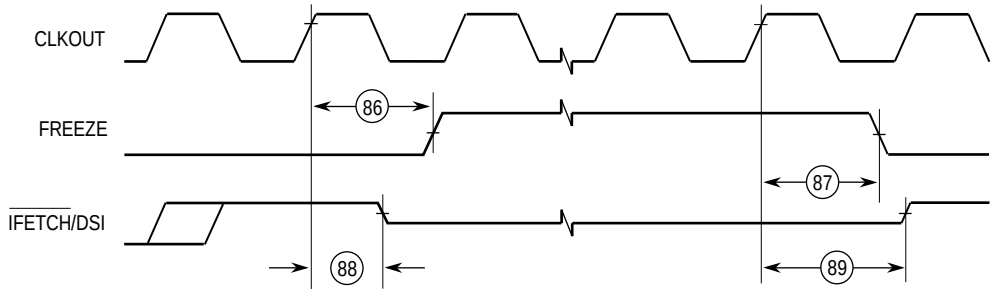


Figure 11-11. Background Debug Mode FREEZE Timing

11.8 DMA MODULE AC ELECTRICAL SPECIFICATIONS (See notes (a), (b), and (c) corresponding to part operation, GND = 0 Vdc, TA = 0 to 70°C; see Figure 11-12)

Num.	Characteristic	3.3 V or 5.0 V 16.78 MHz Min	3.3 V or 5.0 V 16.78 MHz Max	5.0 V 25.16 MHz Min	5.0 V 25.16 MHz Max	Unit
1	CLKOUT Low to $\overline{AS}$, $\overline{DACK}$, $\overline{DONE}$ Asserted	—	30	—	20	ns
2	CLKOUT Low to $\overline{AS}$, $\overline{DACK}$ Negated	—	30	—	20	ns
3	$\overline{DREQx}$ Asserted to $\overline{AS}$ Asserted (for DMA Bus Cycle)	$3t_{cyc} + t_{AIST} + t_{CLSA}$				ns
4[1]	Asynchronous Input Setup Time to CLKOUT Low	8, 5	—	5	—	ns
5	Asynchronous Input Hold Time from CLKOUT Low	15	—	10	—	ns
6	$\overline{AS}$ to $\overline{DACK}$ Assertion Skew	–15	15	–10	10	ns
7	$\overline{DACK}$ to $\overline{DONE}$ Assertion Skew	–15	15	–8	8	ns
8	$\overline{AS}$, $\overline{DACK}$, $\overline{DONE}$ Width Asserted	100	—	70	—	ns
8A	$\overline{AS}$, $\overline{DACK}$, $\overline{DONE}$ Width Asserted (Fast Termination Cycle)	40	—	28	—	ns

NOTES:
(a) The 16.78-MHz @ 3.3 V ±0.3 V specifications are preliminary and apply to the MC68349V.
(b) The 16.78-MHz @ 5.0 V ±5% specifications are preliminary and apply to the MC68349.
(c) The 25.16 MHz @ 5.0 V ±5% specifications are preliminary and apply to the MC68349.
1. Specification #4 for 16.78 MHz @ 3.3 V ±0.3 V will be 8 ns.

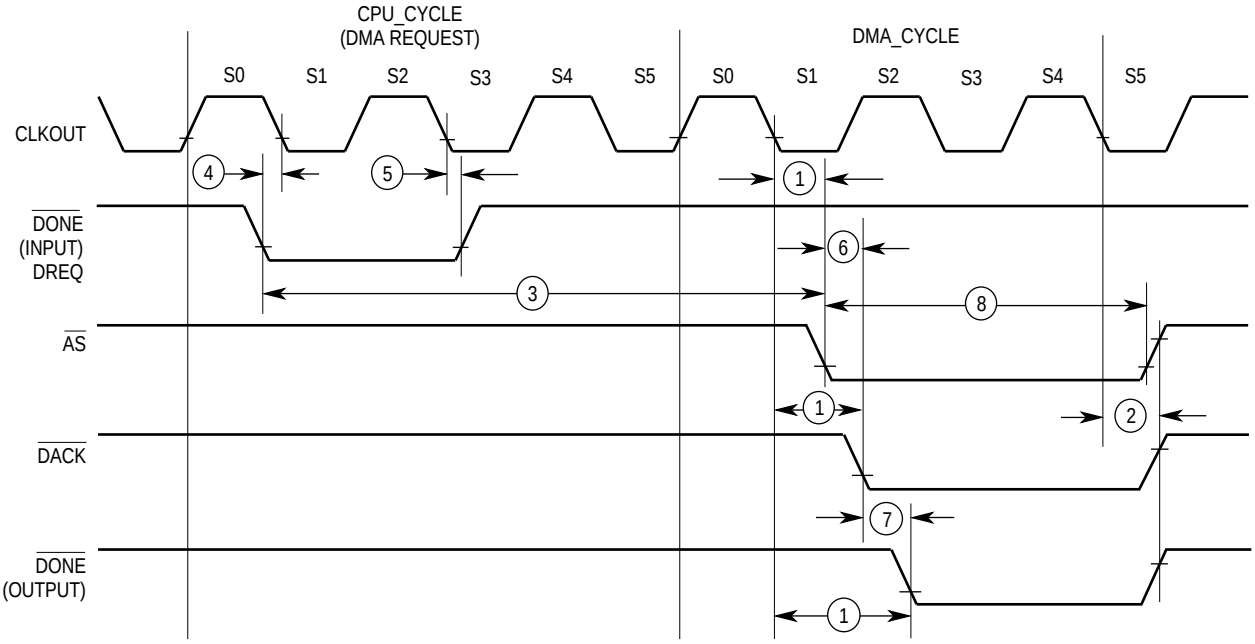


Figure 11-12. DMA Signal Timing Diagram

11.9 SERIAL MODULE ELECTRICAL SPECIFICATIONS (See notes (a), (b), and (c) corresponding to part operation, GND = 0 Vdc, TA = 0 to 70°C; see numbered notes; see Figures 11-13–11-16)

Num.	Characteristic	Symbol	3.3 V or 5.0 V 16.78 MHz Min	3.3 V or 5.0 V 16.78 MHz Max	5.0 V 25.16 MHz Min	5.0 V 25.16 MHz Max	Unit
1	CLKOUT Cycle Time	t_{cyc}	59.6	—	40	—	ns
2	Clock Rise or Fall Time	t_{rf}	—	10	—	5	ns
3[2]	Clock Input (X1 or SCLK) Synchronizer Setup Time	t_{CS}	8, 5	—	5	—	ns
4	Clock Input (X1 or SCLK) Synchronizer Hold Time	t_{CH}	15	—	8	—	ns
5	TxD Data Valid from CLKOUT High	t_{VLD}	$0.5\ t_{cyc}$				Max
6	X1 Cycle Time	t_{X1}	$2.25\ t_{cyc}$				Min
7	X1 High or Low Time	t_{X1HL}	$0.55\ t_{cyc} + 0.75(t_{CS} + t_{CH})$				Min
8	SCLK High or Low Time, Asynchronous (16x) Mode	t_{AHL}	$t_{cyc} + t_{CS} + t_{CH}$				Min
9[1]	SCLK High Time, Synchronous (1x) Mode	t_{SH}	$t_{cyc}(Gx) + t_{CS}(Gx) + t_{CH}(Gx)$				Min
10	SCLK Low Time, Synchronous (1x) Mode	t_{SL}	greater of $(\{1.5t_{cyc}(Tx) + t_{CS}(Tx) + t_{VLD}(Tx)\} + 0.5t_{cyc}(Rx) + t_{CS}(Rx) + t_{CH}(Rx)\})$ or t_{SH}				Min
11	TxD Data Valid from SCLK Low, Synchronous (1x) Mode	$t_{T \times D}$	$1.5t_{cyc}(Tx) + t_{CS}(Tx) + t_{VLD}(Tx)$				Max
12	RxD Setup Time to SCLK High, Synchronous (1x) Mode	$t_{R \times S}$	$0.5t_{cyc}(Rx) + t_{CS}(Rx) + t_{CH}(Rx)$				Min
13	RxD Hold Time from SCLK High, Synchronous (1x) Mode	$t_{R \times H}$	$0.5t_{cyc}(Rx) + t_{CS}(Rx) + t_{CH}(Rx)$				Min

NOTES:
- (a) The 16.78-MHz @ 3.3 V ±0.3 V specifications are preliminary and apply to the MC68349V.
- (b) The 16.78-MHz @ 5.0 V ±5% specifications are preliminary and apply to the MC68349.
- (c) The 25.16 MHz @ 5.0 V ±5% specifications are preliminary and apply to the MC68349.
1. Asynchronous operation numbers take into account a receiver and transmitter operating at different clock frequencies. (Rx) refers to receiver value. (Tx) refers to transmitter value. (Gx) refers to the value that is greater, either receiver or transmitter.
2. Specification #3 for 16.78 MHz @ 3.3 V ±0.3 V will be 8 ns.

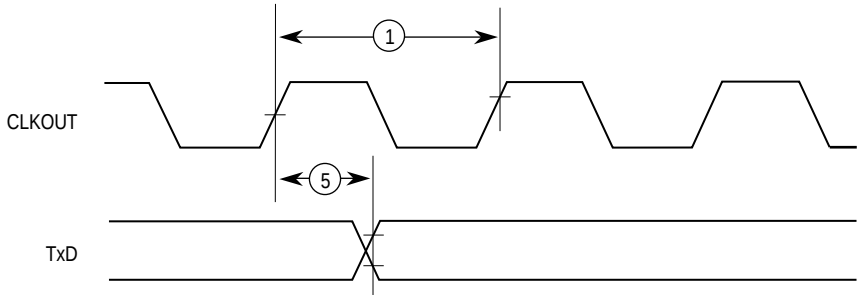


Figure 11-13. Serial Module General Timing Diagram

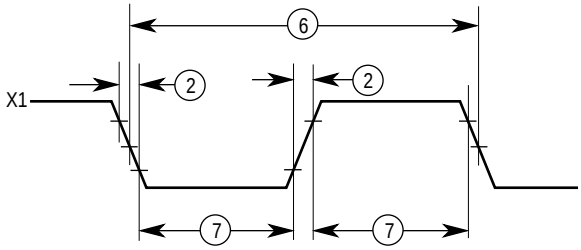


Figure 11-14. Serial Module Asynchronous Mode Timing (X1)

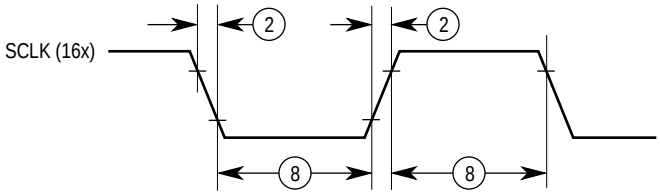


Figure 11-15. Serial Module Asynchronous Mode Timing (SCLK–16X)

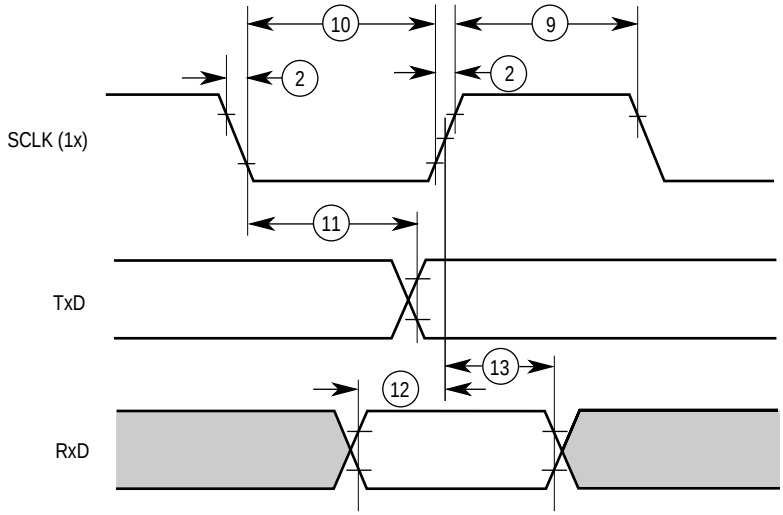


Figure 11-16. Serial Module Synchronous Mode Timing Diagram

11.10 IEEE 1149.1 ELECTRICAL SPECIFICATIONS (See notes (a), (b), and (c) corresponding to part operation, GND = 0 Vdc, TA = 0 to 70°C; see Figures 11-17–11-19)

Num.	Characteristic	3.3 V or 5.0 V 16.78 MHz Min	3.3 V or 5.0 V 16.78 MHz Max	5.0 V 25.16 MHz Min	5.0 V 25.16 MHz Max	Unit
	TCK Frequency of Operation	0	16.78	0	25	MHz
1	TCK Cycle Time in Crystal Mode	59.6	—	40	—	ns
2	TCK Clock Pulse Width Measured at 1.5 V	28	—	18	—	ns
3	TCK Rise and Fall Times	0	5	0	3	ns
6	Boundary Scan Input Data Setup Time	16	—	10	—	ns
7	Boundary Scan Input Data Hold Time	26	—	18	—	ns
8	TCK Low to Output Data Valid	0	40	0	26	ns
9	TCK Low to Output High Impedance	0	60	0	40	ns
10	TMS, TDI Data Setup Time	15	—	10	—	ns
11	TMS, TDI Data Hold Time	15	—	10	—	ns
12	TCK Low to TDO Data Valid	0	25	0	16	ns
13	TCK Low to TDO High Impedance	0	25	0	16	ns

NOTES:
(a) The 16.78-MHz @ 3.3 V ±0.3 V specifications are preliminary and apply to the MC68349V.
(b) The 16.78-MHz @ 5.0 V ±5% specifications are preliminary and apply to the MC68349.
(c) The 25.16 MHz @ 5.0 V ±5% specifications are preliminary and apply to the MC68349.

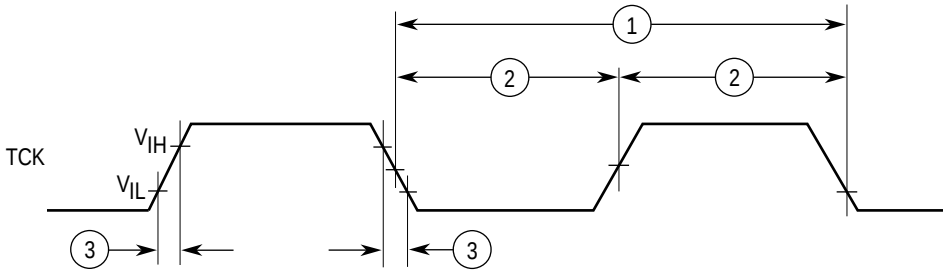


Figure 11-17. Test Clock Input Timing Diagram

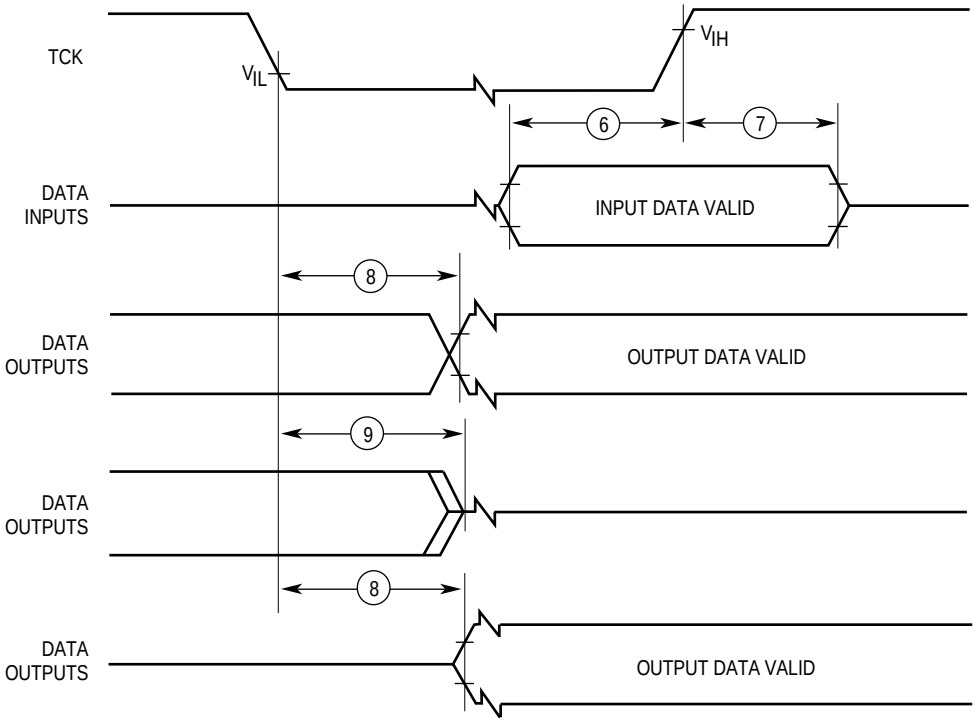


Figure 11-18. Boundary Scan Timing Diagram

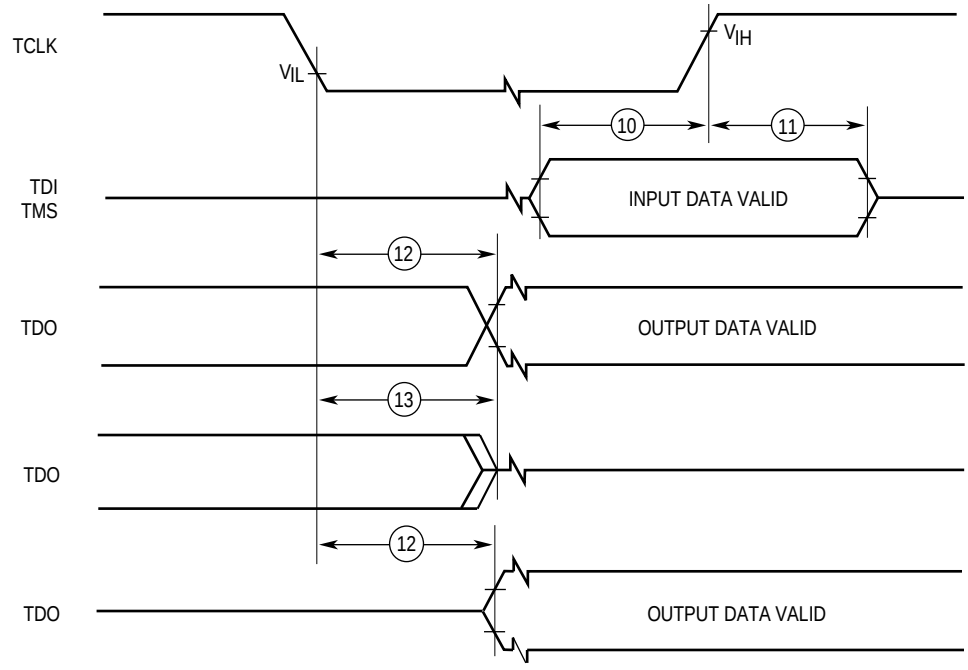


Figure 11-19. Test Access Port Timing Diagram

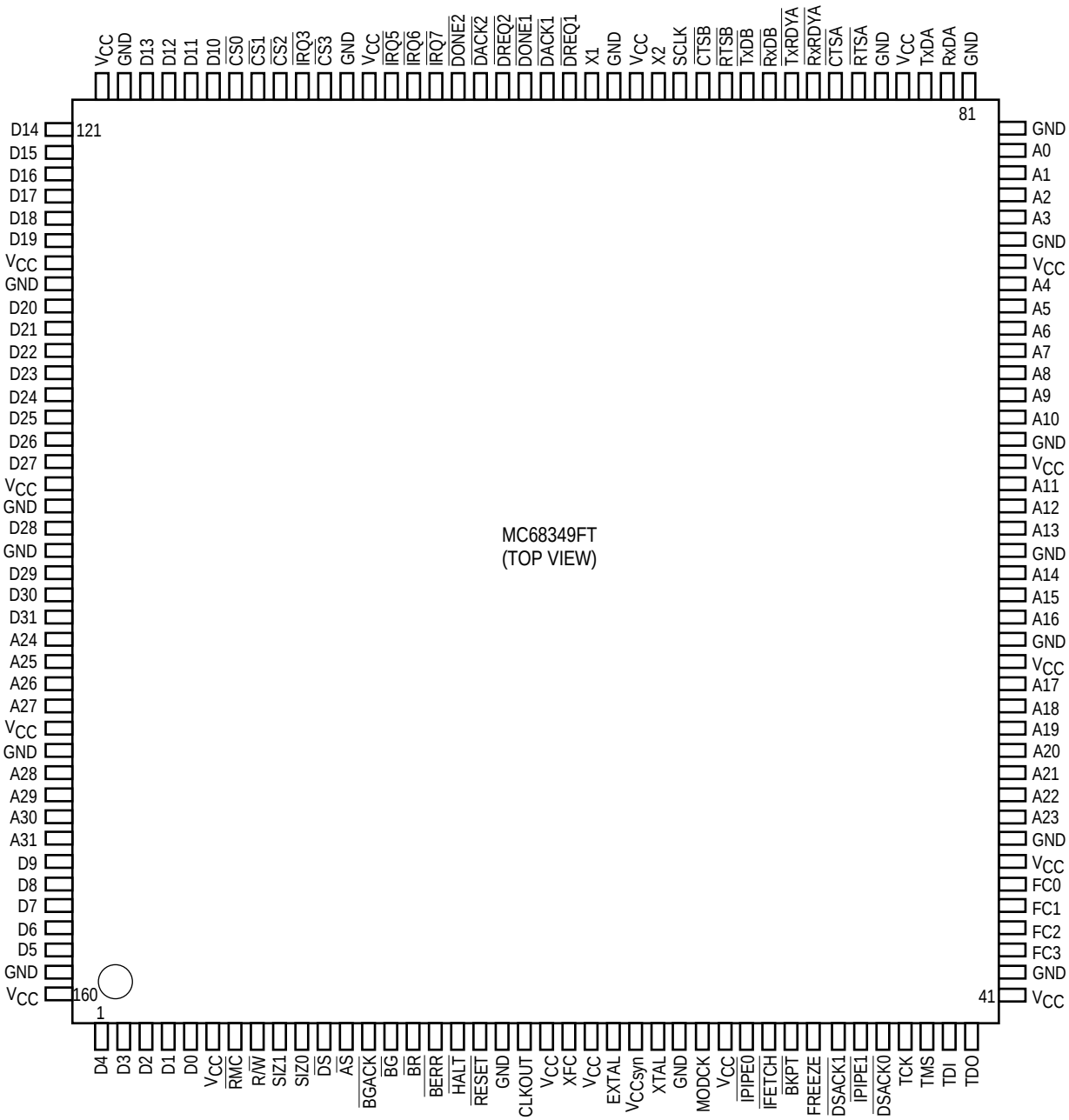
SECTION 12
ORDERING INFORMATION AND MECHANICAL DATA

This section contains ordering information, pin assignments, and package dimensions of the MC68349.

12.1 STANDARD MC68349 ORDERING INFORMATION

Supply Voltage	Package Type	Frequency (MHz)	Temperature	Order Number
5.0 V	Plastic Quad Flat Pack FT Suffix	0 – 16.78 0 – 25	0°C to +70°C 0°C to +70°C	MC68349FT16 MC68349FT25
3.3 V	Plastic Quad Flat Pack FT Suffix	0 – 16.78	0°C to +70°C	MC68349FT16V

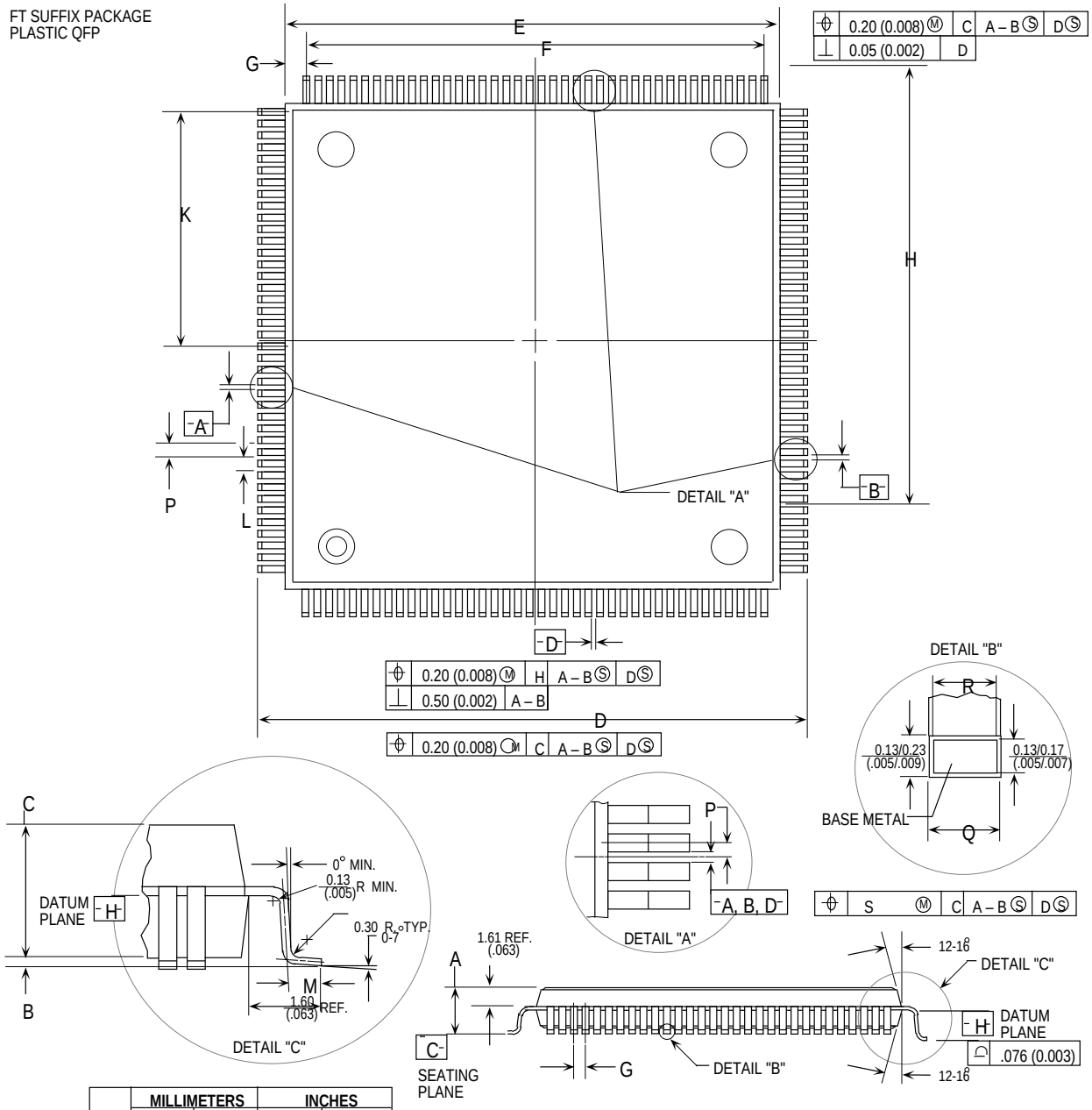
12.2 PIN ASSIGNMENT—160-LEAD PLASTIC QUAD FLAT PACK (FT SUFFIX)



The V_{CC} and GND pins are separated into groups to help electrically isolate the output drivers for different functions of the MC68349. These groups are shown in the following table for the FT suffix package.

Pin Group — FT Suffix	V_{CC}	GND
Address Bus, Function Codes	47, 56, 65, 74, 148	48, 57, 66, 75, 80, 149
Data Bus	120, 127, 137, 160	119, 128, 138, 159
$\overline{AS}$, $\overline{BG}$, CLKOUT, $\overline{DS}$, FREEZE, $\overline{HALT}$, $\overline{IFETCH}$, $\overline{IPIPE1}$, $\overline{IPIPE0}$, MODCK, $\overline{RESET}$, $\overline{RMC}$, R/$\overline{W}$, $\overline{SIZ}$x, TDO, Internal Logic	6, 21, 23, 41	19, 27, 42
$\overline{CS}$x, $\overline{DACK}$x, $\overline{DONE}$x, $\overline{IRQ}$x, $\overline{RTS}$x, RxRDYA , TxDx, $\overline{TxRDYA}$, Internal Logic	84, 96, 108	81, 85, 97, 109
Oscillator (VCCSYN)	25	—
Internal Only	29	140, 61

FT SUFFIX PACKAGE
PLASTIC QFP



	MILLIMETERS		INCHES	
DIM	MIN	MAX	MIN	MAX
A	—	4.07	—	.154
B	0.25	—	.010	—
C	3.17	3.67	.125	.144
D	30.95	31.45	1.218	1.238
E	27.90	28.10	1.098	1.106
F	25.35 REF.		.998 REF.	
G	1.33 REF.		.052 REF.	
H	30.95	31.45	1.218	1.238
J	27.90	28.10	1.098	1.106
K	25.35 REF.		.998 REF.	
L	1.33 REF		.052 REF.	
M	0.65	0.95	.026	.037
N	—	—	—	—
P	0.65 BSC.		.0256 BSC.	
Q	0.22	0.38	.009	.015
R	0.22	0.33	.009	.013
S	0.12		.005	

NOTES:
1. DIMENSIONING AND TOLERANCING PER ANSI Y14.5M, 1982.
2. CONTROLLING DIMENSION: MILLIMETER. INCHES ARE IN "( )".
3. DATUM PLANE **-H-** IS LOCATED AT BOTTOM OF LEAD AND IS COINCIDENT WITH THE LEAD WHERE THE LEAD EXITS THE PLASTIC BODY AT THE BOTTOM OF THE PARTING LINE.
4. DATUMS **-A-**, **-B-**, AND **-D-** TO BE DETERMINED AT DATUM PLANE **-H-**.
5. DIMENSIONS S AND V TO BE DETERMINED AT SEATING PLANE **-C-**.
6. DIMENSIONS A AND B DO NOT INCLUDE MOLD PROTRUSION. ALLOWABLE PROTRUSION IS 0.254mm(0.010) PER SIDE. DIMENSIONS A AND B DO INCLUDE MOLD MISMATCH AND ARE DETERMINED AT DATUM PLANE **-H-**.
7. DIMENSION D DOES NOT INCLUDE DAMBAR PROTRUSION. ALLOWABLE DAMBAR PROTRUSION SHALL BE 0.08 (0.003) TOTAL IN EXCESS OF THE D DIMENSION AT MAXIMUM MATERIAL CONDITION. DAMBAR CANNOT BE LOCATED ON THE LOWER RADIUS OR THE FOOT.
8. PACKAGE TOP DIMENSIONS ARE SMALLER THAN BOTTOM DIMENSIONS BY 0.20(.008) MILLIMETERS.

INDEX

— **T** —