MC68332

USER'S MANUAL

MOTOROLA

MC68332

USER'S MANUAL

PREFACE

The MC68332 microcontroller unit (MCU) is an integral module of Motorola's M68300 Family of 32-bit MCUs. The *MC68332 User's Manual* describes the capabilities, operation, and functions of the MC68332 MCU.

This user's manual is organized as follows:

For additional information pertaining to the CPU32 processor used in the MC68332, refer to the *CPU32 Central Processor Unit Reference Manual*, Motorola document number CPU32RM/AD. For information pertaining to the timer system used in the MC68332, refer to the *TPU Time Processor Unit Reference Manual* , Motorola document number TPURM/AD.

TABLE OF CONTENTS

Paragraph Number	Title	Page Number

Section 1
Device Overview

Section 2
Signal Descriptions

TABLE OF CONTENTS (Continued)

Section 3
Bus Operation

TABLE OF CONTENTS (Continued)

TABLE OF CONTENTS (Continued)

Paragraph Number	Title	Page Number

Section 4
System Integration Module — SIM

TABLE OF CONTENTS (Continued)

Section 5
Queued Serial Module — QSM

TABLE OF CONTENTS (Continued)

Section 6
Standby Ram (with TPU Emulation)

TABLE OF CONTENTS (Continued)

Section 7
CPU32 Overview

Section 8
TPU Overview

TABLE OF CONTENTS (Continued)

TABLE OF CONTENTS (Continued)

Paragraph Number	Title	Page Number

TABLE OF CONTENTS (Continued)

Paragraph Number	Title	Page Number

Index

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

SECTION 1
DEVICE OVERVIEW

The MC68332 is a 32-bit integrated microcontroller, combining high-performance data manipulation capabilities with powerful peripheral subsystems. The MC68332 is the first member of the M68300 Family of modular embedded controllers featuring fully static, high-speed complementary metal-oxide semiconductor (CMOS) technology. Based on the powerful MC68020, the CPU32 instruction processing module provides enhanced system performance and utilizes the extensive software base for the Motorola M68000 Family. The following block diagram (Figure 1-1) shows the major components of the MC68332.
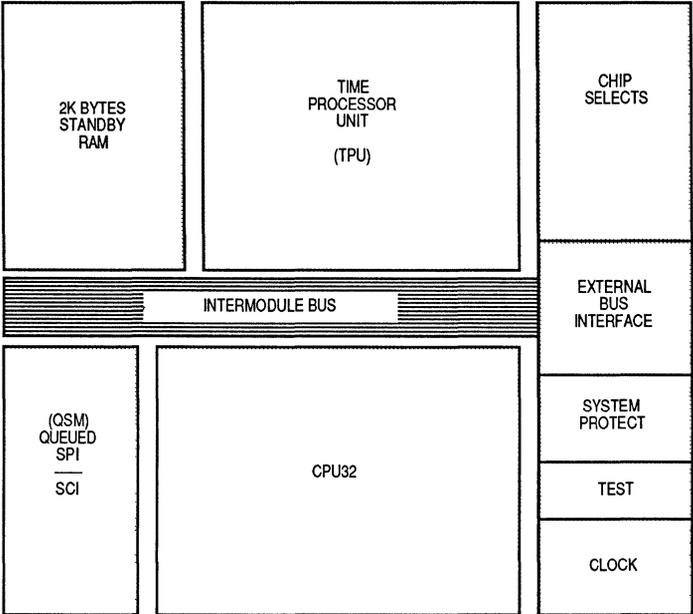
Figure 1-1. Block Diagram of MC68332

1

The MC68332 contains intelligent peripheral modules such as the time processor unit (TPU), which provides 16 microcoded channels for performing time-related activities from simple input capture or output compare to complicated motor control or pulse width modulation. High-speed serial communications are provided by the queued serial module (QSM) with synchronous and asynchronous protocols available. Two kilobytes of fully static standby RAM allow fast two-cycle access for system and data stacks and variable storage with provision for battery backup. Twelve chip selects enhance system integration for fast external memory or peripheral access. These modules are connected on-chip via the intermodule bus (IMB).

The major features of the MC68332 are as follows:
- Low-Power Operation: 600 mW Maximum; 500 µW in Standby Mode
- Frequency: 16.78 MHz at 5-V Supply, Software Programmable
- Technology: 1-Micron High-Density Complementary Metal-Oxide Semiconductor (HCMOS), Static Design
- Transistor Count: 422,000
- Package: 132-Pin Plastic Quad Flat Pack (PQFP)
- Modular Architecture in a Single Chip
- CPU: 32-Bit M68000 Family (Upward Object-Code Compatible from the MC68010)
- New Instructions for Controller Applications
- Intelligent 16-Bit Timer:
  — 16 Independent, Programmable Channels and Pins
  — Any Channel Can Perform Any Time Function (Input Capture, Output Compare, Pulse Width Modulation (PWM), etc.)
  — Two Timer Count Registers with 2-Bit Programmable Prescalers
  — Selectable Channel Priority Levels
- Two Serial I/O Subsystems:
  — Enhanced M68HC11-Type Serial Communications Interface (SCI) Universal Asynchronous Receiver Transmitter (UART) with Parity
  — Enhanced M68HC11-Type Serial Peripheral Interface with I/O Queue (QSPI)
- On-Chip Memory: 2K Bytes Standby RAM
- On-Chip, Programmable, Chip-Select Logic
  Up to 12 Signals for Memory and Peripheral I/O Select
- System Failure Protection:
  — M68HC11-Type Computer Operating Properly (COP) Watchdog Timer
  — M68HC11-Type Periodic Interrupt Timer
  — M68000 Family Spurious Interrupt, HALT, and Bus Timeout Monitors
- Up to 32 Discrete I/O Pins

1.1 Central Processor Unit

The CPU32 is upward compatible with the M68000 Family that excels at processing calculation-intensive algorithms and supporting high-level languages. All of the MC68010 and most of the MC68020 enhancements, such as virtual memory support, loop mode operation, instruction pipeline, and 32-bit mathematical operations, are supported. Powerful addressing modes provide compatibility with existing software programs and increase the efficiency of high-level language compilers. New instructions, such as table lookup and interpolate and low power stop, support the specific requirements of controller applications.

1.2 Intelligent Peripherals

To improve total system throughput, the MC68332 features intelligent, stand alone subsystems. These subsystems include the microcoded TPU, the QSM, the system integration module (SIM), and the 2K-byte standby RAM module. These modules work together with the CPU32 to reduce part count, size, and cost of system implementation.

1.2.1 Time Processor Unit (TPU)

The TPU provides optimum performance in controlling time-related activity. The TPU drastically reduces the need for CPU intervention with its dedicated execution unit, tri-level prioritized scheduler, data storage RAM, dual-time bases, and microcode ROM. The TPU controls 16 independent, orthogonal channels, each with associated I/O pin and capable of performing any time function. Each channel also contains a dedicated event register, allowing both match and input capture functions. The timing algorithms presently available in microcoded ROM include the following:

- Discrete Input/Output

- Period Measurement with Additional Transition Detection

- Period Measurement with Missing Transition Detection

- Position-Synchronized Pulse Generator

- Stepper Motor Control

- Output Match

- Period/Pulse Width Accumulator

Each channel can be synchronized to either of two 16-bit free-running counters with a prescaler. One counter is based on the system clock and provides resolution to 500 ns. The second counter, based on an external reference, provides resolution to 250 ns. Channels may also be linked together, allowing the user to reference operations on one channel to the occurrence of a specified action on another channel to provide intertask control.

1.2.2 Queued Serial Module (QSM)

The QSM contains two serial ports. The QSPI provides easy peripheral expansion or interprocessor communications via a full-duplex, synchronous, three-line bus: data in, data out, and a serial clock. Four programmable peripheral-select pins provide addressability for up to 16 peripheral devices. The QSPI is enhanced with the addition of a queue contained in a small RAM. This allows the QSPI to handle up to 16 serial transfers of 8–16 bits each or to transmit a stream of data up to 256 bits long without CPU intervention. A special wraparound mode allows the user to do continuous sampling of a serial peripheral, automatically updating the QSPI RAM for efficient interfacing to serial analog-to-digital (A/D) converters.

The SCI provides a standard nonreturn to zero (NRZ) (mark/space) format. Advanced error detection circuitry catches noise glitches to 1/16 of a bit time in duration. Word length is software selectable between 8 or 9 bits, and its modulus-type baud rate generator provides baud rates from 64–524 kbaud based on a 16.78-MHz system clock. The SCI features full- or half-duplex operation, with separate transmitter and receiver enable bits and double buffering of data. Optional parity generation and detection provide either even or odd parity check capability. Wakeup functions allow the CPU to run uninterrupted until either a true idle line is detected or a new address byte is received.

1.2.3 System Integration Module

The SIM includes an external interface and various functions that reduce the need for external glue logic. The SIM contains the external bus interface (EBI), 12 chip selects, system protection, test, and clock submodules.

1.2.3.1 External Bus Interface

Based on the MC68020 bus, the external bus provides 24 address lines and a 16-bit data bus. The data bus allows dynamic sizing between 8- and 16-bit data accesses. Read-modify-write cycles are provided for via the RMC signal. External bus arbitration is accomplished by a three-line handshaking interface.

1.2.3.2 Chip Selects

Twelve independent programmable chip selects provide fast, two-cycle external memory or peripheral access. Block size is programmable from a minimum of 2K bytes to 1M bytes in length. Accesses can be preselected for either 8- or 16-bit transfers. Up to 13 wait states can be programmed for insertion during the access. All bus interface signals are automatically handled by the chip-select logic.

1.2.3.3 System Protection Submodule

System protection is provided on the MC68332 by various monitors and timers, including the bus monitor, HALT monitor, spurious interrupt monitor, software watchdog timer, and the periodic interrupt timer. These system functions are integrated on the microcontroller to reduce board size and the cost required by external components.

1.2.3.4 Test Submodule

The test module consolidates the microcontroller test logic into a single block to facilitate production testing. Scan paths throughout the MC68332 provide signature analysis checks on internal logic.

1.2.3.5 System Clock

The system clock is generated by an on-chip phase-locked loop circuit to run the device up to 16.78 MHz from a 32.768-kHz crystal. The system speed can be changed dynamically, providing either high performance or low power consumption under software control. With its fully static CMOS design, it is possible to completely stop the system clock using a low power stop instruction, while still retaining the contents of the registers and on-board RAM.

1.3 Standby Ram Module

This module contains 2K bytes of fast static RAM powered by V_{DD} in normal operation. During power-down, the RAM contents are maintained by power on the standby voltage pin, V_{STBY}. The RAM is especially useful for the CPU as system stack space or variable storage, due to its fast two-cycle access time. Alternately, it can be used by the TPU as emulation RAM for new timer algorithms. Data can be read or written by byte, word, or long-word length. The RAM can be mapped to any 2K-byte boundary in the address map.

1.4 Module Memory Map

Figure 1-2 illustrates the memory map of the MC68332. The RAM array is positioned by the base address register in the RAM CTRL block. Reset forces the RAM array to be disabled. Unimplemented blocks are mapped externally.

```
$YFF000 ┌──────────────────┐        ┌──────────────────┐
        │                  │        │                  │
        │                  │        │    RAM ARRAY     │
        │                  │        │   (2.0K BYTES)   │
$YFF800 │                  │        │                  │
$YFFA00 ├──────────────────┤        │                  │
$YFFA80 │       SIM        │        └──────────────────┘
$YFFB00 ├── — RESERVED ── ─┤
$YFFB40 │     RAM CTRL     │
$YFFC00 ├── — RESERVED ── ─┤
        │                  │
        │       QSM        │
        │                  │
$YFFE00 ├──────────────────┤
        │                  │
        │       TPU        │
$YFFFFF └──────────────────┘
```

Y = M111, where M is the modmap signal state on the IMB, which reflects the state of the modmap bit in the module configuration register of the system integration module (Y= $7 or $F).

Module	Size	Address Bus Decoding						Base
	(Bytes)	A23					A0	Address
SIM	128	M111	1111	1111	1010	0XXX	XXXX	$YFFA00
RAM CTRL	64	M111	1111	1111	1011	00XX	XXXX	$YFFB00
QSM	512	M111	1111	1111	110X	0XXX	XXXX	$YFFC00
TPU	512	M111	1111	1111	111X	0XXX	XXXX	$YFFE00

Figure 1-2. Module Memory Map

2

SECTION 2
SIGNAL DESCRIPTIONS

This section contains brief descriptions of the MC68332 input and output signals in their functional groups.

2.1 Signal Index

The input and output signals for the MC68332 are listed in Table 2-1. (See Figure 2-1 for a block diagram of the function signal groups.) Both the names and mnemonics are shown, along with brief descriptions of the signals. For more detail on each signal, refer to the paragraph in this section named for the signal, and the reference in that paragraph to a description of the related operations. Guaranteed timing specifications for the SIM and EBI signals listed in Table 2-1 can be found in **SECTION 10 ELECTRICAL CHARACTERISTICS**.

2.2 Address Bus (A23–A0)

These three-state outputs provide the address for the current bus cycle, except in the CPU32 address space. Refer to **3.4 CPU Space Cycles** for more information on the CPU32 address space. A23 is the most significant address signal. Refer to **3.1.3 Address Bus** for information on the address bus and its relationship to bus operation.

2.3 Data Bus (D15–D0)

These three-state bidirectional signals provide the general-purpose data path between the MC68332 and all other devices. The data path is a maximum of 16 bits wide, but can be dynamically sized to support 8-bit or 16-bit transfers. D15 is the most significant bit of the data bus. **Refer to 3.1.5 Data Bus** for information on the data bus and its relationship to bus operation. The data bus also serves as the mode-select pins during reset as described in **3.7 Reset Operation**.

Figure 2-1. Function Signal Group

SIGNAL DESCRIPTIONS
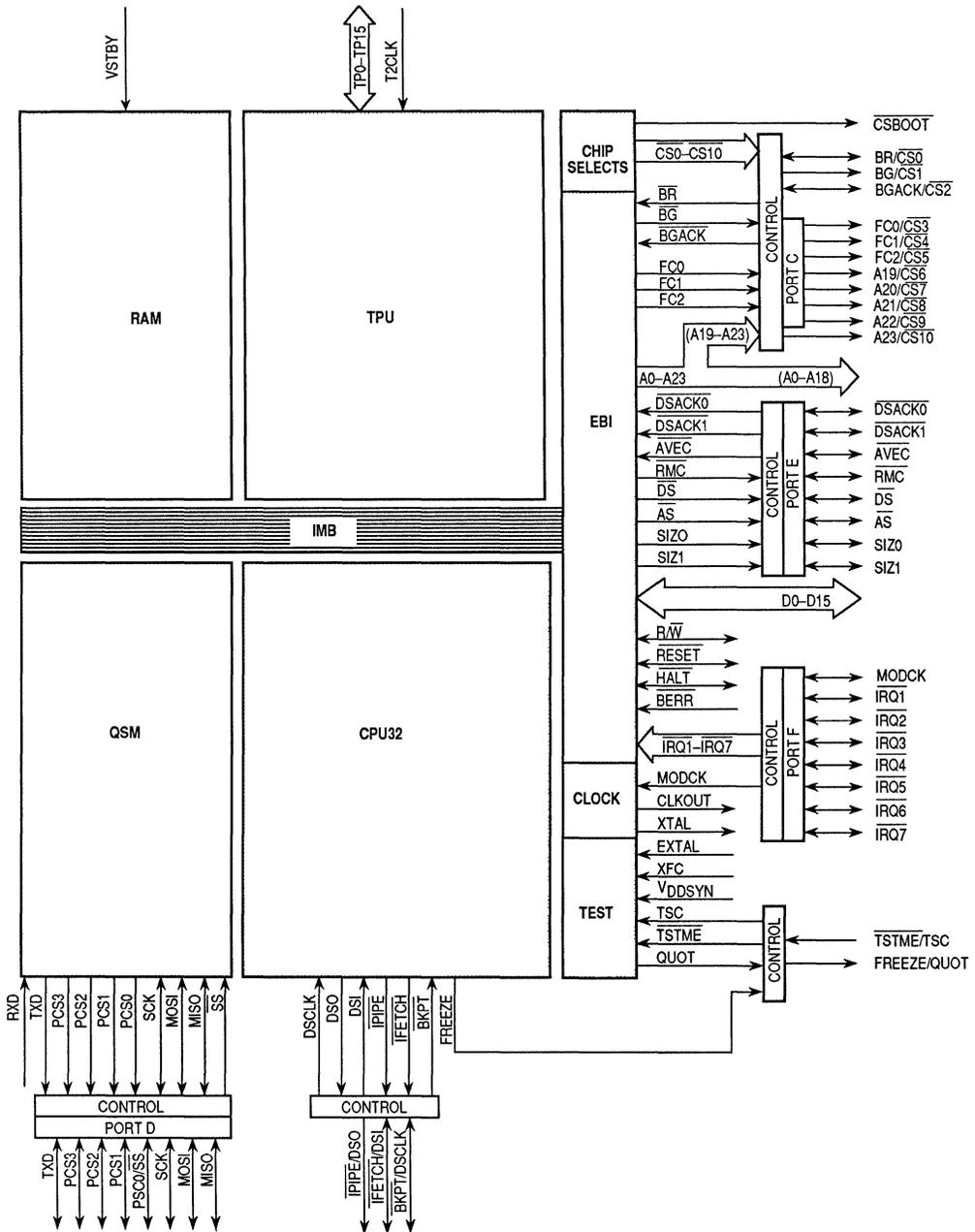
Table 2-1. Signal Index (Sheet 1 of 2)

Signal Name	Mnemonic	Function
Address Bus	A23–A0	24-bit address bus
Data Bus	D15–D0	16-bit data bus used to transfer byte or word data per bus cycle
Function Codes	FC2–FC0	Identify the processor state and the address space of the current bus cycle
Boot Chip Select	CSBOOT	Chip-select boot startup ROM containing user's reset vector and initialization program
Chip Selects	CS10–CS0	Enables peripherals at programmed addresses
Bus Request	BR	Indicates that an external device requires bus mastership
Bus Grant	BG	Indicates that current bus cycle is complete and the MC68332 has relinquished the bus
Bus Grant Acknowledge	BGACK	Indicates that an external device has assumed bus mastership
Data and Size Acknowledge	DSACK1 DSACK0	Provides asynchronous data transfers and dynamic bus sizing
Autovector	AVEC	Requests an automatic vector during an interrupt acknowledge cycle
Read-Modify-Write Cycle	RMC	Identifies the bus cycle as part of an indivisible read-modify-write operation
Address Strobe	AS	Indicates that a valid address is on the address bus
Data Strobe	DS	During a read cycle, DS indicates that an external device should place valid data on the data bus. During a write cycle, DS indicates that valid data is on the data bus.
Size	SIZ1–SIZ0	Indicates the number of bytes remaining to be transferred for this cycle
Read/Write	R/W	Indicates the direction of data transfer on the bus
Interrupt Request Level	IRQ7–IRQ1	Provides an interrupt priority level to the CPU
Reset	RESET	System reset
Halt	HALT	Suspends external bus activity
Bus Error	BERR	Indicates that an erroneous bus operation is being attempted
System Clockout	CLKOUT	Internal system clock
Crystal Oscillator	XTAL, XTAL	Connections for an external crystal to the internal oscillator circuit
External Filter Capacitor	XFC	Connection pin for an external capacitor to filter the circuit of the phase-locked loop
Clock Mode Select	MODCK	Selects the source of the internal system clock
Instruction Fetch	FETCH	Indicates when the CPU is performing an instruction word prefetch and when the instruction pipeline has been flushed
Instruction Pipe	IPIPE	Used to track movement of words through the instruction pipeline
Breakpoint	BKPT	Signals a hardware breakpoint to the CPU
Freeze	FREEZE	Indicates that the CPU has acknowledged a breakpoint
Quotient Out	QUOT	Furnishes the quotient bit of the polynomial divider for test purposes
Test Mode Enable	TSTME	Hardware enable for test mode

2

Table 2-1. Signal Index (Sheet 2 of 2)

Signal Name	Mnemonic	Function
Three-State Control	TSC	Places all output drivers in a high-impedance state
Development Serial In, Out,Clock	DSI, DSO, DSCLK	Serial I/O and clock for background debug mode
TPU Channels	TP15–TP0	TPU channel input/output
TPU Clock In	T2CLK	External clock source to the TPU
SCI Receive Data	RXD	Serial input to the SCI
SCI Transmit Data	TXD	Serial output from the SCI
Peripheral Chip Select	$\overline{PCS3}$–$\overline{PCS0}$	QSPI peripheral chip selects
Slave Select	$\overline{SS}$	Places the QSPI in slave mode
QSPI Serial Clock	SCK	Furnishes the clock from the QSPI in master mode or to the QSPI in slave mode
Master-In Slave-Out	MISO	Furnishes serial input to the QSPI in master mode, and serial output from the QSPI in slave mode
Master-Out Slave-In	MOSI	Furnishes serial output from the QSPI in master mode, and serial input to the QSPI in slave mode
Standby RAM	V_{STBY}	Power supply for RAM
Synchronizer Power	V_{DDSYN}	Power supply to VCO
System Power Supply and Return	V_{DD}, V_{SS}	Power supply and return to the MCU

2.4 Function Codes (FC2–FC0)

These three-state outputs identify the processor state and the address space of the current bus cycle. Refer to **3.4 CPU Space Cycles** for more information.

2.5 Chip Selects ($\overline{CS10}$–$\overline{CS0}$, $\overline{CSBOOT}$)

These output signals enable peripherals at programmed addresses. $\overline{CSBOOT}$ is the dedicated chip select for a boot ROM containing the user's reset vector and initialization program. Refer to **4.3 Chip-Select Submodule** for more information on chip selects.

2.6 Bus Control Signals

These signals control the bus transfer operations of the MC68332.

2.6.1 Data and Size Acknowledge ($\overline{DSACK1}$) ($\overline{DSACK0}$)

These two active-low input signals allow asynchronous data transfers and dynamic data bus sizing between the MC68332 and external devices. Refer to **3.1.7 Bus Cycle Termination Signals** for more information on these signals and their relationship to dynamic bus sizing.

2.6.2 Autovector ($\overline{\text{AVEC}}$)

This active-low input signal requests an automatic vector during an interrupt ac-knowledge cycle. Refer to **3.4.3.2 Autovector Interrupt Acknowledge Cycle** for additional information on autovector.

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

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. Refer to **3.3.3 Read-Modify-Write Cycle** for additional in-formation.

2.6.4 Address Strobe ($\overline{\text{AS}}$)

This output signal is driven by the bus master to indicate that a valid address is on the address bus. The function code, size, and read/write signals are also valid when $\overline{\text{AS}}$ is asserted. Refer to **3.1.4 Address Strobe** for information about the relationship of $\overline{\text{AS}}$ to bus operation.

2.6.5 Data Strobe ($\overline{\text{DS}}$)

During a read cycle, this output signal is driven by the bus master to indicate that an external device should place valid data on the data bus. During a write cycle, the data strobe indicates that valid data is on the data bus. Refer to **3.1.6 Data Strobe** for information about the relationship of $\overline{\text{DS}}$ to bus operation.

2.6.6 Transfer Size (SIZ0, SIZ1)

These active-low output signals are driven by the bus master to indicate the number of operand bytes remaining to be transferred in the current bus cycle. Refer to **3.2.1 Dynamic Bus Sizing** for more information.

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

This active-high input/output signal is driven by the bus master to indicate the direction of 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. Refer to **3.1.1 Bus Control Signals** for more information.

2.7 Bus Arbitration Signals

The following signals are the three bus arbitration control signals used to de-termine the bus master.

2.7.1 Bus Request (BR)

This active-low input signal indicates that an external device needs to become the bus master. Refer to **3.6 Bus Arbitration** for more information.

2.7.2 Bus Grant (BG)

Assertion of this active-low output signal indicates that the bus master has relinquished the bus. Refer to **3.6.2 Bus Grant** for more information.

2.7.3 Bus Grant Acknowledge (BGACK)

Assertion of this active-low input indicates that an external device has become the bus master. Refer to **3.6.3 Bus Grant Acknowledge** for more information.

2.8 Interrupt Request Level (IRQ7–IRQ1)

These active-low input signals are prioritized interrupt request lines. IRQ7 is the highest priority. IRQ6–IRQ1 are internally maskable interrupts and IRQ7 is nonmaskable. Refer to interrupts in the CPU32 manual for more information.

2.9 Exception Control Signals

These signals are used by the microcontroller unit (MCU) to recover from an exception encountered by the system.

2.9.1 Reset (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 SIM) resets the MCU as well as all external devices. A reset signal from the CPU32 (asserted as part of the RESET instruction) resets external devices only; the internal state of the CPU32 and other on-chip modules is not altered. When asserted by the MCU, it is guaranteed to be asserted for a minimum of 512 clock cycles. Refer to **3.7 Reset Operation** for a description of reset bus operation and reset in the CPU32 manual for information about the reset exception.

2.9.2 Halt (HALT)

This active-low open-drain bidirectional signal is asserted to suspend external bus activity, to request a retry when used with BERR, or for single-step operation. As an output, HALT indicates a double bus fault by the CPU. Refer to **3.5 Bus Exception Control Cycles** for a description of the effects of HALT bus operation.

2.9.3 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 **3.5 Bus Exception Control Cycles** for a description of the effects of $\overline{\text{BERR}}$ bus operation.

2.10 Clock Signals

These signals are used by the MCU for controlling or generating the system clocks. Refer to **4.2 Clock Synthesizer** for more information on the various clock signals.

2.10.1 System Clock (CLKOUT)

This output signal is the internal system clock and is used as the bus timing reference by external devices. CLKOUT can be turned off or slowed in low power stop mode. See **4.1.1 Module Configuration Register** for more information.

2.10.2 Crystal Oscillator (EXTAL, XTAL)

These two pins are the connections for an external crystal to the internal oscillator circuit. An external oscillator should serve as input to the EXTAL pin, when used. See **4.2 Clock Synthesizer** for more information.

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 V_{DDSYN}.

2.10.4 Clock Mode Select (MODCK)

The state of this 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. IF MODCK is low during reset, an external frequency appearing at the EXTAL pin furnishes the system clock.

2.11 Instrumentation and Test Signals

These signals are used for test or software debugging.

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

This active-low output signal indicates when the CPU is performing an instruction word prefetch and when the instruction pipeline has been flushed. Refer to instruction fetch in the CPU32 manual for information about $\overline{\text{IFETCH}}$.

2.11.2 Instruction Pipe ($\overline{\text{IPIPE}}$)

This active-low output signal is used to track movement of words through the instruction pipeline. Refer to instruction pipe in the CPU32 manual for information about $\overline{\text{IPIPE}}$.

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

This active-low input signal is used to signal a hardware breakpoint to the CPU. Refer to hardware breakpoints in the CPU32 manual for information about $\overline{\text{BKPT}}$.

2.11.4 Freeze (FREEZE)

Assertion of this active-high output signal indicates that the CPU has acknowledged a breakpoint and has initiated background mode operation. Refer to development support in the CPU32 manual for more information about FREEZE and background mode.

2.11.5 Quotient Out (QUOT)

This active-high output furnishes the quotient bit of the polynomial divider for test purposes.

2.11.6 Test Mode Enable ($\overline{\text{TSTME}}$)

This active-low input signal is a hardware enable required to enter test mode. Refer to **4.5.3 Entering Test Mode** for information about $\overline{\text{TSTME}}$.

2.11.7 Three-State Control (TSC)

When this input signal is driven to 1.6 times V_{DD}, the MCU places all of its output drivers in a high-impedance state.

2.11.8 Development Serial In, Out, Clock (DSI, DSO, DSCLK)

These signals provide serial communications for background debug mode. Refer to development support in the CPU manual for more information on background debug mode.

2.12 Time Processing Unit Signals

These signals are used by the time processor unit (TPU) for sample and control.

2.12.1 TPU Channel Signals (TP15–TP0)

These 16 bidirectional signals provide the inputs and outputs for each channel of the TPU. The direction of each signal, either as input or output, is determined by the TPU. Refer to timer interface signals in the TPU reference manual for more information on the TPU signals.

2.12.2 TPU Clock In (T2CLK)

This input signal furnishes an external clock source to timer count register 2 (TCR2) in the TPU. Refer to timer interface signals in the TPU reference manual for more information on the TPU signals.

2.13 Queued Serial Module Signals

The following signals are used by the queued serial module (QSM) for data and clock signals. All of these pins (except RXD) can be used for discrete input/output if they are not being used for serial communications interface (SCI) or queued serial peripheral interface (QSPI) functions. Refer to **5.3 QSM Pins** for a general discussion on the QSM signals.

2.13.1 SCI Receive Data (RXD)

This input signal furnishes serial data input to the SCI. It may not be used as discrete input/output. Refer to **5.6.5 Receiver Operation** for more information on the SCI signals.

2.13.2 SCI Transmit Data (TXD)

This signal is the serial data output from the SCI. Refer to **5.6.4 Transmitter Operation** for more information on the SCI signals.

2.13.3 Peripheral Chip Selects ($\overline{PCS3}$–$\overline{PCS0}$)

These bidirectional signals provide (four) QSPI peripheral chip selects. Refer to **5.4.3.2 QSM Pin Assignment Register (QPAR)** for more information.

2.13.4 Slave Select ($\overline{SS}$)

Assertion of this bidirectional signal places the QSPI in slave mode. Refer to **5.5.5.2 Slave Mode** for more information.

2.13.5 QSPI Serial Clock (SCK)

This bidirectional signal furnishes the clock from the QSPI in master mode or furnishes the clock to the QSPI in slave mode. Refer to **5.5.5 Operating Modes and Flowcharts** for more information.

2.13.6 Master-In Slave-Out (MISO)

This bidirectional signal furnishes serial data input to the QSPI in master mode, and serial data output from the QSPI in slave mode. Refer to **5.5.5 Operating Modes and Flowcharts** for more information.

2.13.7 Master-Out Slave-In (MOSI)

This bidirectional signal furnishes serial data output from the QSPI in master mode, and serial data input to the QSPI in slave mode. Refer to **5.5.5 Operating Modes and Flowcharts** for more information.

2.14 Standby Ram (V$_{STBY}$)

This pin supplies power to the RAM for data retention when the MCU is without power. Refer to **6.3.2 Standby Operation** for more information concerning V$_{STBY}$.

2.15 Synthesizer Power (V$_{DDSYN}$)

This pin supplies a quiet power source to the VCO to provide greater frequency stability.

2.16 System Power and Ground (V$_{DD}$ and V$_{SS}$)

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

Table 2-2. Signal Summary

Signal Name	Mnemonic	Input/Output	Active State	Three-State
Address Bus	A23–A0	Output	High	Yes
Data Bus	D15–D0	Input/Output	High	Yes
Function Codes	FC2–FC0	Output	High	Yes
Boot Chip Select	CSBOOT	Output	Low	No
Chip Selects	CS10–CS0	Output	Low	No
Bus Request	BR	Input	Low	No
Bus Grant	BG	Output	Low	No
Bus Grant Acknowledge	BGACK	Input	Low	—
Data and Size Acknowledge	DSACK1/DSACK0	Input	Low	—
Autovector	AVEC	Input	Low	—
Read-Modify-Write Cycle	RMC	Output	Low	Yes
Address Strobe	AS	Output	Low	Yes
Data Strobe	DS	Output	Low	Yes
Size	SIZ1/SIZ0	Output	High	Yes
Read/Write	R/W	Output	High/Low	Yes
Interrupt Request Level	IRQ7–IRQ1	Input	Low	—
Reset	RESET	Input/Output	Low	No
Halt	HALT	Input/Output	Low	—
Bus Error	BERR	Input	Low	—
System Clockout	CLKOUT	Input	—	—
Crystal Oscillator	EXTAL, XTAL	Input	—	—
External Filter Capacitor	XFC	Input	—	—
Clock Mode Select	MODCK	Input	High	—
Instruction Fetch	IFETCH	Output	Low	—
Instruction Pipe	IPIPE	Output	Low	—
Breakpoint	BKPT	Input	Low	—
Freeze	FREEZE	Output	High	—
Quotient Out	QUOT	Output	High	—
Test Mode Enable	TSTME	Input	Low	—
Three-State Control	TSC	Input	High	—
Development Serial In, Out, Clock	DSI, DSO, DSCLK	Input/Output	—	—
TPU Channels	TP15–TP0	Input/Output	High	—
TPU Clock In	T2CLK	Input	—	—
SCI Receive Data	RXD	Input	High	—
SCI Transmit Data	TXD	Output	High	—
Peripheral Chip Select	PCS3–PCS0	Input/Output	Low	—
Slave Select	SS	Input/Output	Low	—
QSPI Serial Clock	SCK	Input/Output	—	—
Master-In Slave-Out	MISO	Input/Output	High	—
Master-Out Slave-In	MOSI	Input/Output	High	—
Standby RAM	V_{STBY}	Input	High	—
Synchronizer Power	V_{DDSYN}	Input	—	—
System Power Supply and Return	V_{DD}, V_{SS}	Input	—	—

2

2

SIGNAL DESCRIPTIONS

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 the reset operation. Operation of the bus is the same whether the MCU or an external device is the bus master; the names and descriptions of bus cycles are from the point of view of the bus master. For exact timing specifications, refer to **SECTION 10 ELECTRICAL CHARACTERISTICS.**

The MCU architecture supports byte, word, and long-word operands, allowing access to 8- and 16-bit data ports through the use of asynchronous cycles controlled by the data transfer (SIZ1 and SIZ0) and data size acknowledge pins ($\overline{\text{DSACK1}}$ and $\overline{\text{DSACK0}}$). The MCU requires word and long-word operands to be located in memory on word or long-word boundaries. The only type of transfer that can be misaligned is a single-byte transfer to an odd address, referred to as an odd-byte transfer. For an 8-bit port, multiple bus cycles may be required for an operand transfer due to either misalignment or a port width smaller than the operand size.

3.1 Bus Transfer Signals

The bus transfers information between the MCU and an external memory or peripheral device. External devices can accept or provide 8 bits or 16 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 MCU contains an address bus that specifies the address for the transfer and a data bus that transfers the data. Control signals indicate the beginning of the cycle, the address space and size of the transfer, and the type of cycle. 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. The bus operates in an asynchronous mode for any port width. The bus and control input signals are internally synchronized to the MCU clock, introducing a delay. This delay is the time period required for the MCU to sample an input signal, synchronize the input to the internal clocks, and determine whether it is high or low.

Furthermore, for all inputs, the MCU 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. 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 MCU is not predictable; however, the MCU 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.
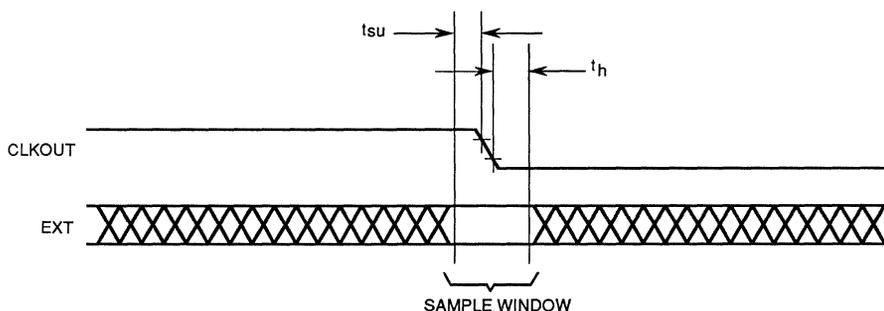
Figure 3-1. Input Sample Window

3.1.1 Bus Control Signals

The MCU initiates a bus cycle by driving the address, size, function code, and read/write outputs. At the beginning of a bus cycle, the size signals (SIZ1, SIZ0) are driven along with the function code signals. SIZ1 and SIZ0 indicate the number of bytes remaining to be transferred during an operand cycle (consisting of one or more bus cycles). Table 3-1 shows the encoding of SIZ1 and SIZ0. These signals are valid while address strobe ($\overline{AS}$) is asserted. The read/write (R/$\overline{W}$) signal determines the direction of the transfer during a bus cycle. This signal changes state, when required, at the beginning of a bus cycle, and 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 two consecutive write cycles. The read-modify-write cycle signal ($\overline{RMC}$) 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.

Table 3-1. Size Signal Encoding

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

3.1.2 Function Codes

The function code signals (FC2–FC0) select one of eight address spaces to which the address applies. These spaces are designated as either user or supervisor, and program or data spaces. One other address space is designated as CPU space to allow the CPU 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.

Function codes can be considered as extensions of the 24-bit linear address that can provide up to eight 16M-byte address spaces. Function codes are automatically generated by the CPU to select address spaces for data and program at the user and supervisor privilege levels, and a CPU address space used for processor functions. User programs can 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 CPU status bit is set for supervisor accesses and cleared for user accesses to provide supervisor/user differentiation. Refer to **3.4 CPU Space Cycles** for more information.

3.1.3 Address Bus

The address bus signals (A23–A0) define the address of the byte (or the most significant byte) to be transferred during a bus cycle. The MCU 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

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

3.1.5 Data Bus

The data bus signals (D15–D0) comprise a bidirectional, nonmultiplexed parallel bus that contains the data being transferred to or from the MCU. A read or write operation may transfer 8 or 16 bits of data (1 or 2 bytes) in one bus cycle. During a read cycle, the data is latched by the MCU on the last falling edge of the clock for that bus cycle. For a write cycle, all 16 bits of the data bus are driven, regardless of the port width or operand size. The MCU places the data on the data bus one-half clock cycle after $\overline{AS}$ is asserted in a write cycle.

3.1.6 Data Strobe

The data strobe ($\overline{DS}$) is a timing signal that applies to the data bus. For a read cycle, the MCU asserts $\overline{DS}$ to signal the external device to place data on the bus. It is asserted at the same time as $\overline{AS}$ during a read cycle. For a write cycle, $\overline{DS}$ signals to the external device that the data to be written is valid on the bus. The MCU asserts $\overline{DS}$ one full clock cycle after the assertion of $\overline{AS}$ during a write cycle.

3.1.7 Bus Cycle Termination Signals

During bus cycles, external devices assert the data transfer and size acknowledge signals ($\overline{DSACK1}$ and/or $\overline{DSACK0}$) as part of the bus protocol. During a read cycle, this signals the MCU 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 MCU the size of the port for the bus cycle just completed, as shown in Table 3-2. Refer to **3.3.1 Read Cycle** for the timing of $\overline{DSACK1}$ and $\overline{DSACK0}$.

The bus error ($\overline{BERR}$) signal is also a bus cycle termination indicator and can be used in the absence of $\overline{DSACKx}$ to indicate a bus error condition. It 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 10 ELECTRICAL CHARACTERISTICS**. Additionally, the $\overline{BERR}$ and $\overline{HALT}$ signals can be asserted together to indicate a retry termination. Again, the $\overline{BERR}$ and $\overline{HALT}$ signals can be asserted simultaneously, in lieu of, or in conjunction with, the $\overline{DSACKx}$ signals.

The internal bus monitor can be used to generate the $\overline{BERR}$ signal for internal and internal-to-external transfers in all the following descriptions. (Refer to 4.1.5 **Bus Monitors** for information on the functions of the bus monitor.) An external bus master must provide its own $\overline{BERR}$ generation and drive the $\overline{BERR}$ pin,

since the internal $\overline{\text{BERR}}$ monitor has no information about transfers initiated by an external bus master.

Finally, the autovector ($\overline{\text{AVEC}}$) signal can be used to terminate interrupt acknowledge cycles, indicating that the MCU should internally generate a vector number to locate an interrupt handler routine. $\overline{\text{AVEC}}$ is ignored during all other bus cycles.

3.2 Data Transfer Mechanism

The MCU architecture supports byte, word, and long-word operands allowing, access to 8- and 16-bit data ports through the use of asynchronous cycles controlled by the data transfer and size acknowledge inputs ($\overline{\text{DSACK1}}$ and $\overline{\text{DSACK0}}$).

3.2.1 Dynamic Bus Sizing

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

Table 3-2. $\overline{\text{DSACK}}$ Codes and Results

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

For example, if the MCU is executing an instruction that reads a long-word operand from a 16-bit port, the MCU 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 16-bit device always returns $\overline{\text{DSACKx}}$ for a 16-bit port (regardless of whether the bus cycle is a byte or 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 16-bit port must reside on data bus bits 15–0, and an 8-bit port must reside on data bus bits 15–8. This requirement minimizes the number of bus cycles needed to transfer data to 8- and 16-bit ports and ensures that the MCU correctly transfers valid data.

The MCU always attempts to transfer the maximum amount of data on all bus cycles; for a word operation, it always assumes that the port is 16 bits wide when beginning the bus cycle. Operand bytes 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 OP0 (most significant) and OP1. The single byte of a byte-length operand is OP0. These designations are used in the figures and descriptions that follow.

Figure 3-2 shows the required organization of data ports on the MCU bus for both 8- and 16-bit devices. The four bytes shown in Figure 3-2 are connected through the internal data bus and data multiplexer to the external data bus. The data multiplexer establishes the necessary connections for different combinations of address and data sizes. The multiplexer takes the two bytes of the 16-bit bus and routes them to their required positions. The positioning of bytes is determined by the size (SIZ1 and SIZ0) and address (A0) outputs. The SIZ1 and SIZ0 outputs indicate the remaining number of bytes to be transferred during the current bus cycle, as shown in Table 3-1. The number of bytes transferred during a write or read bus cycle is equal to or less than the size indicated by the SIZ1 and SIZ0 outputs, depending on port width. For example, during the first bus cycle of a long-word transfer to a word port, the size outputs indicate that four bytes are to be transferred, although only two bytes are transferred on that bus cycle.

The address line A0 also affects the operation of the data multiplexer. During an operand transfer, A23–A1 indicate the word base address of that portion of the operand to be accessed, and A0 indicates the byte offset from the base. Figure 3-2 lists the bytes required on the data bus for read cycles. The entries shown as OPn are portions of the requested operand that are read or written during that bus cycle and are defined by SIZ1, SIZ0, and A0 for the bus cycle. The transfer cases marked misaligned are not allowed in this MCU.

3.2.2 Misaligned Operands

In this architecture, the basic operand size is 16 bits. Operand misalignment refers to whether an operand is aligned on a word boundary or overlaps the word boundary. This is determined by address line A0. When A0 is low, the ad-

dress is even and is a word and byte boundary. When A0 is high, the address is odd and is a byte boundary only. A byte operand is properly aligned at any address; a word or long-word operand is misaligned at an odd address.

```
OPERAND    OP0      OP1      OP2      OP3
       31           OP0      OP1      OP2
              23             OP0      OP1
                        15            OP0
                                7        0
                                Data Bus
```
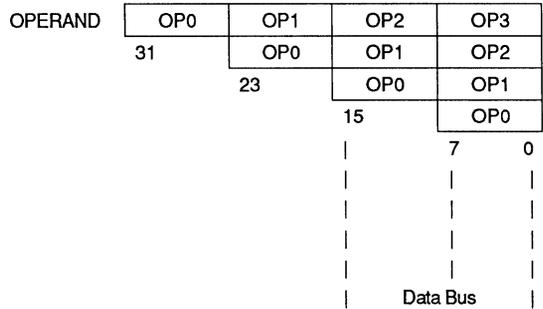
Case	Transfer Case	SIZ1	SIZ0	A0	$\overline{DSACK1}$	$\overline{DSACK0}$	D15 D8	D7 D0
a)	Byte to Byte	0	1	X	1	0	OP0	(OP0)
b)	Byte to Word (Even)	0	1	0	0	X	OP0	(OP0)
c)	Byte to Word (Odd)	0	1	1	0	X	(OP0)	OP0
d)	Word to Byte (Aligned)	1	0	0	1	0	OP0	(OP1)
e)	Word to Byte (Misaligned)*	1	0	1	1	0	OP0	(OP0)
f)	Word to Word (Aligned)	1	0	0	0	X	OP0	OP1
g)	Word to Word (Misaligned)*	1	0	1	0	X	(OP0)	OP0
h)	3 Byte to Byte (Aligned)*†	1	1	0	1	0	OP0	(OP1)
i)	3 Byte to Byte (Misaligned)†	1	1	1	1	0	OP0	(OP0)
j)	3 Byte to Word (Aligned)*†	1	1	0	0	X	OP0	OP1
k)	3 Byte to Word (Misaligned)*†	1	1	1	0	X	(OP0)	OP0
l)	Long Word to Byte (Aligned)	0	0	0	1	0	OP0	(OP1)
m)	Long Word to Byte (Misaligned)*	0	0	1	1	0	OP0	(OP0)
n)	Long Word to Word (Aligned)	0	0	0	0	X	OP0	OP1
o)	Long Word to Word (Misaligned)*	0	0	1	0	X	(OP0)	OP0

NOTES:

Operands in parentheses are ignored by the MC68332 during read cycles.

Misaligned transfer cases, identified by an asterisk (*), are not supported by the MC68332.

†Three-byte transfer cases occur only as a result of a long word to byte transfer.

Figure 3-2. MCU Interface to Various Port Sizes

Each bus cycle can transfer at most a word of data aligned on a word boundary. If the MCU transfers a long-word operand over a 16-bit port, the most significant operand word is transferred on the first bus cycle and the least significant operand word on a following bus cycle.

The CPU restricts alignment of all operands (both data and instruction). That is, both word and long-word operands must be located on a word or long-word boundary. The only type of transfer that can be misaligned is a single-byte transfer to an odd address, referred to as an odd-byte transfer.

3.2.3 Operand Transfer Cases

The following cases are examples of all the allowable alignments of operands to ports.

3.2.3.1 Byte Operand to 8-Bit Port, Even (A0 = 0)

The MCU drives the address bus with the desired address and the size pins to indicate a single-byte operand. Refer to Figure 3-3.

Byte Operand	OP0						
	7 0						
Data Bus	D15 D8 D7 D0	SIZ1	SIZ0	A0	DSACK1	DSACK0	
Cycle 1	OP0 (OP0)	0	1	X	1	0	
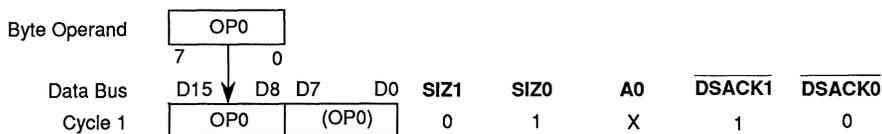
Figure 3-3. Byte Operand to 8-Bit Port, Even (A0 = 0)

For a read operation, the slave responds by placing data on bits 15–8 of the data bus, asserting DSACK0, and negating DSACK1 to indicate an 8-bit port. The MCU then reads the operand byte from bits 15–8 and ignores bits 7–0.

For a write operation, the MCU drives the single-byte operand on both bytes of the data bus because it does not know the size of the port until the DSACKx signals are read. The slave device must recognize the size of the operand and use the address to place the operand in the specified location. The slave then asserts DSACK0 to terminate the bus cycle.

3.2.3.2 Byte Operand to 16-Bit Port, Even (A0 = 0)

The MCU drives the address bus with the desired address and the size pins to indicate a single-byte operand. Refer to Figure 3-4.

Byte Operand | OP0

Data Bus	D15	D8	D7	D0	SIZ1	SIZ0	A0	$\overline{DSACK1}$	$\overline{DSACK0}$
Cycle 1	OP0		(OP0)		0	1	0	0	X

Figure 3-4. Byte Operand to 16-Bit Port, Even (A0 = 0)

For a read operation, the slave responds by placing data on bits 15–8 of the data bus, asserting $\overline{DSACK1}$, and negating $\overline{DSACK0}$ to indicate a 16-bit port. The MCU then reads the operand byte from bits 15–8 and ignores bits 7–0.

For a write operation, the MCU drives the single-byte operand on both bytes of the data bus because it does not know the size of the port until the $\overline{DSACKx}$ signals are read. The slave device then reads the operand from bits 15–8 of the data bus and uses the address to place the operand in the specified location. The slave then asserts $\overline{DSACK1}$ to terminate the bus cycle.

3.2.3.3 Byte Operand to 16-Bit Port, Odd (A0 = 1)

The MCU drives the address bus with the desired address and the size pins to indicate a single-byte operand. Refer to Figure 3-5.

Byte Operand | OP0

Data Bus	D15	D8	D7	D0	SIZ1	SIZ0	A0	$\overline{DSACK1}$	$\overline{DSACK0}$
Cycle 1	OP0		(OP0)		0	1	1	0	X
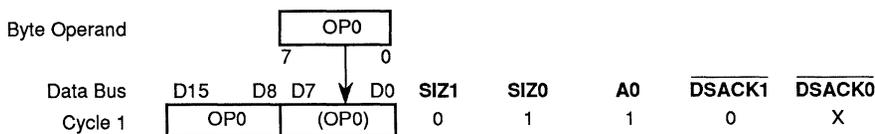
Figure 3-5. Byte Operand to 16-Bit Port, Odd (A0 = 1)

For a read operation, the slave responds by placing data on bits 7–0 of the data bus, asserting $\overline{DSACK1}$, and negating $\overline{DSACK0}$ to indicate a 16-bit port. The MCU then reads the operand byte from bits 7–0 and ignores bits 15–8.

For a write operation, the MCU drives the single-byte operand on both bytes of the data bus because it does not know the size of the port until the $\overline{DSACKx}$ signals are read. The slave device then reads the operand from bits 7–0 of the data bus and uses the address to place the operand in the specified location. The slave then asserts $\overline{DSACK1}$ to terminate the bus cycle.

3.2.3.4 Word Operand to 8-Bit Port, Aligned

The MCU drives the address bus with the desired address and the size pins to indicate a word operand. Refer to Figure 3-6.

Word Operand	OP0	OP1						
	15 8	7 0						
Data Bus	D15 D8	D7 D0	SIZ1	SIZ0	A0	DSACK1	DSACK0
Cycle 1	OP0	(OP1)	1	0	0	1	0
Cycle 2	OP1	(OP1)	0	1	1	1	0
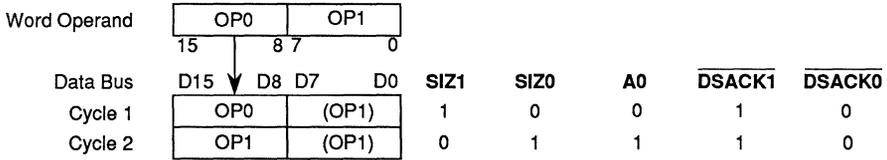
Figure 3-6. Word Operand to 8-Bit Port, Aligned

For a read operation, the slave responds by placing the most significant byte of the operand on bits 15–8 of the data bus and asserting $\overline{DSACK0}$ to indicate an 8-bit port. The MCU reads the most significant byte of the operand from bits 15–8 and ignores bits 7–0. The MCU then decrements the transfer size counter, increments the address, and reads the least significant byte of the operand from bits 15–8 of the data bus.

For a write operation, the MCU drives the word operand on bits 15–0 of the data bus. The slave device then reads the most significant byte of the operand from bits 15–8 of the data bus and asserts $\overline{DSACK0}$ to indicate that it received the data but is an 8-bit port. The MCU then decrements the transfer size counter, increments the address, and writes the least significant byte of the operand to bits 15–8 of the data bus.

3.2.3.5 Word Operand to 16-Bit Port, Aligned

The MCU drives the address bus with the desired address and the size pins to indicate a word operand. Refer to Figure 3-7.

Word Operand	OP0	OP1						
	15	0						
Data Bus	D15 D8	D7 D0	SIZ1	SIZ0	A0	DSACK1	DSACK0
Cycle 1	OP0	OP1	1	0	0	0	X
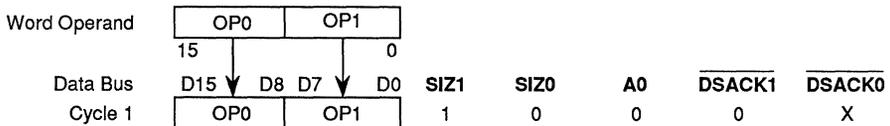
Figure 3-7. Word Operand to 16-Bit Port, Aligned

For a read operation, the slave responds by placing the data on bits 15–0 of the data bus and asserting $\overline{\text{DSACK1}}$ to indicate a 16-bit port. When $\overline{\text{DSACK1}}$ is asserted, the MCU reads the data on the data bus and terminates the cycle.

For a write operation, the MCU drives the word operand on bits 15–0 of the data bus. The slave device then reads the entire operand from bits 15–0 of the data bus and asserts $\overline{\text{DSACK1}}$ to terminate the bus cycle.

3.2.3.6 Long-Word Operand to 8-Bit Port, Aligned

The MCU drives the address bus with the desired address and the size pins to indicate a long word operand. Refer to Figure 3-8.

Long-Word Operand	OP0 OP1	OP2	OP3				
	31 23	15	7 0				

Data Bus	D15 D8	D7 D0	SIZ1	SIZ0	A0	$\overline{\text{DSACK1}}$	$\overline{\text{DSACK0}}$
Cycle 1	OP0	(OP1)	0	0	0	1	0
Cycle 2	OP1	(OP1)	1	1	1	1	0
Cycle 3	OP2	(OP3)	1	0	0	1	0
Cycle 4	OP3	(OP3)	0	1	1	1	0
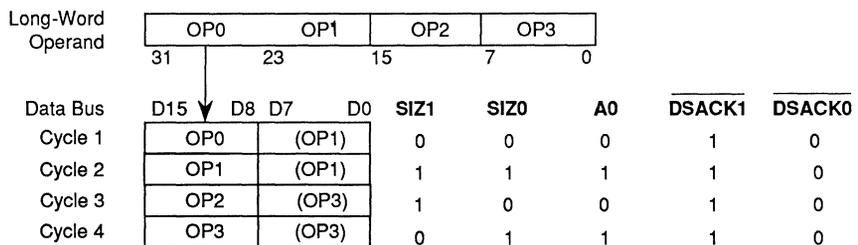
Figure 3-8. Long-Word Operand to 8-Bit Port, Aligned

For a read operation (Figure 3-9), the slave places the most significant byte of the operand on bits 15–8 of the data bus and asserting $\overline{\text{DSACK0}}$ to indicate an 8-bit port. The MCU reads the most significant byte of the operand (byte 0) from bits 15–8 and ignores bits 7–0. The MCU then decrements the transfer size counter, increments the address, initiates a new cycle, and reads byte 1 of the operand from bits 15–8 of the data bus. The MCU repeats the process of decrementing the transfer size counter, incrementing the address, initiating a new cycle, and reading a byte to transfer the remaining two bytes.

For a write operation (Figure 3-10), the MCU drives the two most significant bytes of the operand on bits 15–0 of the data bus. The slave device then reads only the most significant byte of the operand (byte 0) from bits 15–8 of the data bus and asserts $\overline{\text{DSACK0}}$ but not $\overline{\text{DSACK1}}$ to indicate that it received the data but is an 8-bit port. The MCU then decrements the transfer size counter, increments the address, and writes byte 1 of the operand to bits 15–8 of the data bus. The MCU continues to decrement the transfer size counter, increment the address, and write a byte to transfer the remaining two bytes to the slave device.

Figure 3-9. Long-Word Operand Read Timing from 8-Bit Data Port
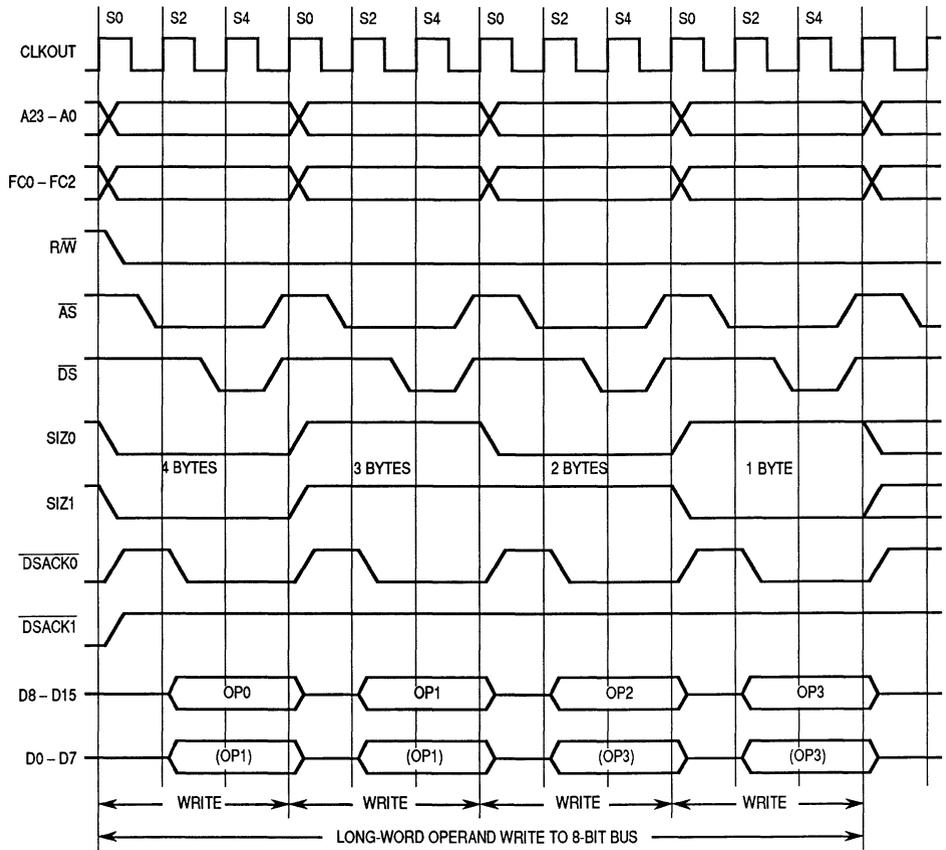
Figure 3-10. Long-Word Write Operand Timing to 8-Bit Port

3.2.3.7 Long-Word Operand to 16-Bit Port, Aligned

Figure 3-11 shows both long-word and word read and write timing to a 16-bit port. The MCU drives the address bus with the desired address and drives the size pins to indicate a long-word operand.

Word Operand	OP0	OP1	OP2	OP3				
	31	23	15	7	0			
Data Bus	D15	D8 D7	D0	SIZ1	SIZ0	A0	$\overline{\text{DSACK1}}$	$\overline{\text{DSACK0}}$
Cycle 1	OP0		OP1	0	0	0	0	X
Cycle 2	OP2		OP3	1	0	0	0	X
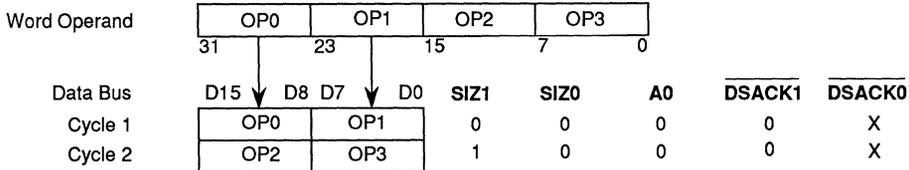
Figure 3-11. Long-Word Operand to 16-Bit Port, Aligned

For a read operation, the slave responds by placing the two most significant bytes of the operand on bits 15–0 of the data bus, and asserting $\overline{\text{DSACK1}}$ to indicate a 16-bit port. The MCU reads the two most significant bytes of the operand (bytes 0 and 1) from bits 15–0. The MCU then decrements the transfer size counter, increments the address, initiates a new cycle, and reads bytes 2 and 3 of the operand from bits 15–0 of the data bus.

For a write operation, the MCU drives the two most significant bytes of the operand on bits 15–0 of the data bus. The slave device then reads the two most significant bytes of the operand (bytes 0 and 1) from bits 15–0 of the data bus and asserts $\overline{\text{DSACK1}}$ to indicate that it received the data and is a 16-bit port. The MCU then decrements the transfer size counter by 2, increments the address by 2, and writes bytes 2 and 3 of the operand to bits 15–0 of the data bus.
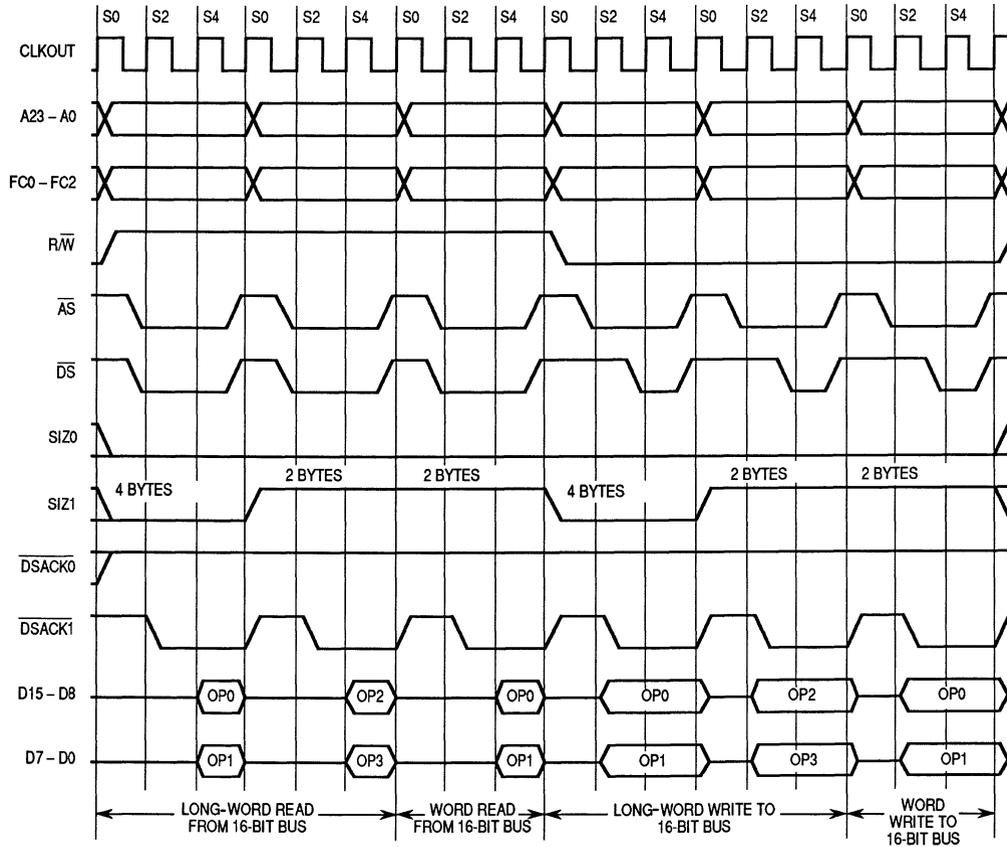
Figure 3-12. Long-Word and Word Read and Write Timing — 16-Bit Port

3

3.2.4 Bus Operation

The MCU bus is used in an asynchronous manner. The external devices connected to the bus can operate at clock frequencies different from the clock for the MCU. 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 the start of a bus cycle, and $\overline{DS}$ is used as a condition for valid data on a write cycle. Decoding the size outputs and lower address line A0 provides strobes that select the active portion of the data bus. The slave device (memory or peripheral) then responds by placing the requested data on the correct portion of the data bus for a read cycle or latching the data on a write cycle, and asserting the $\overline{DSACK1}/\overline{DSACK0}$ combination that corresponds to the port size to terminate the cycle. If no slave responds or the access is invalid, external control logic asserts the $\overline{BERR}$, or $\overline{BERR}$ and $\overline{HALT}$ signal(s) to abort or retry the bus cycle, respectively. The $\overline{DSACKx}$ signals can be asserted before the data from a slave device is valid on a read cycle. The length of time that $\overline{DSACKx}$ may precede data must not exceed a specified value in any asynchronous system to ensure that valid data is latched into the MCU. (Refer to **SECTION 10 ELECTRICAL CHARACTERISTICS** for timing parameters.) Notice that no maximum time is specified from the assertion of $\overline{AS}$ to the assertion of $\overline{DSACKx}$. Although the MCU can transfer data in a minimum of three clock cycles when the cycle is terminated with $\overline{DSACKx}$, the MCU inserts wait cycles in clock period increments until $\overline{DSACKx}$ is recognized. The $\overline{BERR}$ and/or $\overline{HALT}$ signals can be asserted after a $\overline{DSACK}$ signal is asserted. $\overline{BERR}$ and/or $\overline{HALT}$ must be asserted within the time specified after $\overline{DSACKx}$ is asserted in any asynchronous system. *Warning: If this maximum delay time is violated, the MCU may exhibit erratic behavior.*

3.2.5 Synchronization of Asynchronous Inputs to CLKOUT

Although cycles terminated with $\overline{DSACKx}$ signals are classified as asynchronous, cycles terminated with $\overline{DSACKx}$ 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 MCU clock (CLKOUT) in order to be synchronous. Since they terminate bus cycles with the data transfer and size acknowledge signals ($\overline{DSACKx}$), the dynamic bus-sizing capabilities of the MCU are available. The minimum cycle time for these cycles is also three clocks. To support those systems that use the system clock to generate $\overline{DSACKx}$ 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 $\overline{DSACKx}$, the MCU can be guaranteed to recognize that signal level on that specific falling edge of the system clock. If the assertion of $\overline{DSACKx}$ is recognized on a particular falling

edge of the clock, valid data is latched into the MCU (for a read cycle) on the next falling clock edge, provided that the data meets the data setup time. In this case, the parameter for asynchronous operation can be ignored. The timing parameters referred to are described in **SECTION 10 ELECTRICAL CHARACTERISTICS**. Note that 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 $\overline{\text{DSACKx}}$ of three clocks per cycle. To assure proper operation in a synchronous system 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. *Warning: This setup time is critical, and the MCU 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 the $\overline{\text{DS}}$ signal.

3.2.6 Fast Termination Cycles

With an external device that has a fast access time, the chip-select circuit fast-termination option can provide a two-cycle external bus transfer. Since the chip-select circuits are driven from the system clock, the bus cycle termination is inherently synchronized with the system clock.

The $\overline{\text{DSACK}}$ option in the chip-select option registers determine whether internally generated $\overline{\text{DSACKx}}$ or externally generated $\overline{\text{DSACKx}}$ is used. 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 3-13 shows the $\overline{\text{DSACK}}$ timing for two wait states in read, and a fast-termination read and write.

When using the fast-termination option, data strobe is asserted only in a read cycle and not in a write cycle. The STRB field in the option register used must be programmed to address strobe in order to assert the chip select when it is used for a fast-termination write (refer to **4.3.4 Option Registers Description** for more information).

If several chip selects are used to provide control signals to a single device and the match condition occurs simultaneously, the $\overline{\text{DSACK}}$ fields should be programmed to external $\overline{\text{DSACK}}$ or to the same number of wait states with one $\overline{\text{DSACK}}$ field programmed to the internal $\overline{\text{DSACK}}$ option. This prevents a conflict on the internal bus when the wait states are loaded into the $\overline{\text{DSACK}}$

counter, which is shared by all chip selects. Refer to **4.3 Chip-Select Submodule** for more information on chip selects.
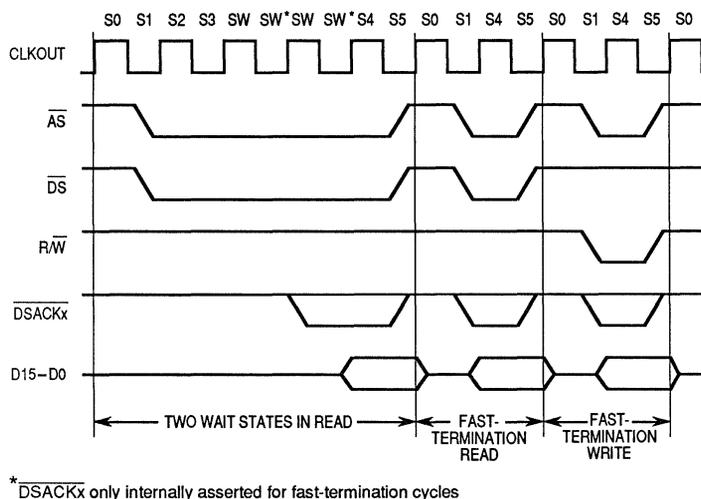
Figure 3-13. Fast-Termination Timing

3.3 Data Transfer Cycles

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

- Address Bus A23–A0
- Data Bus D15–D0
- Control Signals

The address and data buses are both parallel, nonmultiplexed buses. The bus master transfers 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 deskewing all signals it issues at both the start and the end of the cycle. In addition, the bus master is responsible for deskewing the acknowledge and data signals from the slave devices. The following paragraphs define read, write, and read-modify-write cycle operations. Each of the bus cycles is defined as a succession of states. These states apply to the bus operation and are different from the MCU states described for the CPU. 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 MCU receives data from a memory or peripheral device. If the instruction specifies a long-word or word operation, the MCU attempts to read two bytes at once. For a byte operation, the MCU reads one byte. The section of the data bus from which each byte is read depends on the operand size, address signal A0, and the port size. Refer to **3.2.1 Dynamic Bus Sizing** and **3.2.2 Misaligned Operands** for more information on dynamic bus sizing and misaligned operands. Figure 3-14 is a flowchart of a word read cycle.

BUS MASTER   SLAVE

ADDRESS DEVICE

1) SET R/W TO READ
2) DRIVE ADDRESS ON A23-A0
3) DRIVE FUNCTION CODE ON FC2 – FC0
4) DRIVE SIZE PINS FOR OPERAND SIZE
5) ASSERT $\overline{AS}$ AND $\overline{DS}$

PRESENT DATA

1) DECODE ADDRESS
2) PLACE DATA ON D15 – D0
3) DRIVE $\overline{DSACKx}$ SIGNALS

ACQUIRE DATA

1) LATCH DATA
2) NEGATE $\overline{AS}$ AND $\overline{DS}$

TERMINATE CYCLE

1) REMOVE DATA FROM D15 – D0
2) NEGATE $\overline{DSACKx}$

START NEXT CYCLE
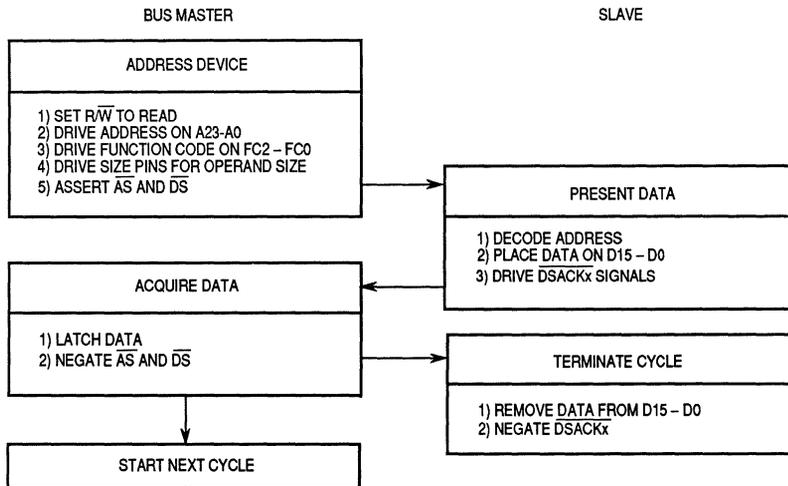
Figure 3-14. Word Read Cycle Flowchart

State 0 — The read cycle starts in state 0 (S0). During S0, the MCU places a valid address on A23–A0 and valid function codes on FC2–FC0. The function codes select the address space for the cycle. The MCU drives R/W high for a read cycle. Size signals SIZ1 and SIZ0 become valid, indicating the number of bytes requested to be transferred.

State 1— One-half clock later in state 1 (S1), the MCU asserts the $\overline{AS}$ indicating that the address on the address bus is valid. The MCU also asserts the $\overline{DS}$ during S1.

State 2 — The selected device uses R/$\overline{\text{W}}$, SIZ1–SIZ0, A0, and $\overline{\text{DS}}$ to place its information on the data bus. One or both of the bytes (D15–D8 and D7–D0) are selected by the size signals and A0. Concurrently, the selected device asserts the $\overline{\text{DSACKx}}$ signals.

State 3 — As long as at least one of the $\overline{\text{DSACKx}}$ 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 $\overline{\text{DSACKx}}$ is not recognized by the start of S3, the MCU inserts wait states instead of proceeding to states 4 and 5. 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 MCU continues to sample the $\overline{\text{DSACKx}}$ signals on the falling edges of the clock until one is recognized.

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

State 5 — The MCU negates $\overline{\text{AS}}$ and $\overline{\text{DS}}$ during S5. It holds the address valid during S5 to provide address hold time for memory systems. R/$\overline{\text{W}}$, SIZ1 and SIZ0, and FC2–FC0 also remain valid throughout S5. The external device keeps its data and $\overline{\text{DSACKx}}$ signals 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 $\overline{\text{DS}}$. *Warning: $\overline{\text{DSACKx}}$ 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 MCU transfers data to memory or a peripheral device. Figure 3-15 is a flowchart of a write-cycle operation for a word transfer.

State 0 — The write cycle starts in S0. During S0, the MCU places a valid address on A23–A0 and valid function codes on FC2–FC0. The function codes select the address space for the cycle. The MCU drives R/$\overline{\text{W}}$ low for a write cycle. Size signals SIZ1 and SIZ0 become valid, indicating the number of bytes to be transferred.

State 1 — One-half clock later in S1, the MCU asserts the $\overline{\text{AS}}$ indicating that the address on the address bus is valid.

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

```
           BUS MASTER                                 SLAVE

    ┌───────────────────────────┐
    │      ADDRESS DEVICE        │
    ├───────────────────────────┤
    │ 1) SET R/W TO WRITE        │
    │ 2) DRIVE ADDRESS ON A23 – A0│
    │ 3) DRIVE FUNCTION CODE ON FC2 – FC0│
    │ 4) DRIVE SIZE PINS FOR OPERAND SIZE│        ┌───────────────────────────┐
    │ 5) ASSERT AS               │──────────────▶│       ACCEPT DATA          │
    │ 6) PLACE DATA ON D15 – D0  │               ├───────────────────────────┤
    │ 7) ASSERT DS               │               │ 1) DECODE ADDRESS          │
    └───────────────────────────┘               │ 2) LATCH DATA FROM D15-D0  │
                                                 │ 3) ASSERT DSACKx SIGNALS   │
    ┌───────────────────────────┐◀──────────────└───────────────────────────┘
    │  TERMINATE OUTPUT TRANSFER │
    ├───────────────────────────┤
    │ 1) NEGATE DS AND AS        │               ┌───────────────────────────┐
    │ 2) REMOVE DATA FROM D15 – D0│─────────────▶│      TERMINATE CYCLE       │
    └───────────────────────────┘               ├───────────────────────────┤
                  │                              │ 1) NEGATE DSACKx           │
                  ▼                              └───────────────────────────┘
    ┌───────────────────────────┐
    │       START NEXT CYCLE     │
    └───────────────────────────┘
```

Figure 3-15. Write Cycle Flowchart

State 3 — The MCU asserts DS during S3. This indicates that the data is stable 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 MCU 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 MCU continues to sample the $\overline{DSACKx}$ signals 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 the data bus (D15–D8 and D7–D0). The size signals and A0 select the bytes of the data bus. If $\overline{DSACKx}$ is not already asserted, the device asserts $\overline{DSACKx}$ to signal that it has successfully stored the data.

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

State 5 — The MCU 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/W, SIZ1 and SIZ0, and FC2–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 $\overline{DS}$. *Warning:* $\overline{DSACKx}$ *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 this MCU, this operation is indivisible, providing semaphore capabilities for multiprocessor systems. During the entire read-modify-write sequence, the MCU asserts the $\overline{RMC}$ signal to indicate that an indivisible operation is occurring. The MCU does not issue a bus grant ($\overline{BG}$) signal in response to a bus request ($\overline{BR}$) signal during this operation. Figure 3-16 is an example of a functional timing diagram of a read-modify-write instruction specified in terms of clock periods.

State 0 — The MCU asserts $\overline{RMC}$ in S0 to identify a read-modify-write cycle. The MCU places a valid address on A23–A0 and valid function codes on FC2–FC0. The function codes select the address space for the operation. Size signals SIZ1–SIZ0 become valid in S0 to indicate the operand size. The MCU drives R/$\overline{W}$ high for the read cycle.

State 1 — One-half clock later in S1, the MCU asserts $\overline{AS}$ indicating that the address on the address bus is valid. The MCU 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. Either or both of the bytes (D15–D8 and D7–D0) are selected by the size signals and A0. Concurrently, the selected device may assert the $\overline{DSACKx}$ signals.

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 MCU 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 MCU 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 MCU latches the incoming data.

State 5 — The MCU 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 MCU holds the address, R/$\overline{W}$, and FC2–FC0 valid in preparation for the write portion of the cycle. The external device keeps its data and \S\UP6(\F(,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 $\overline{DS}$. *Warning:* $\overline{DSACKx}$ *signals that remain asserted beyond this limit may be prematurely detected for the next portion of the operation.*
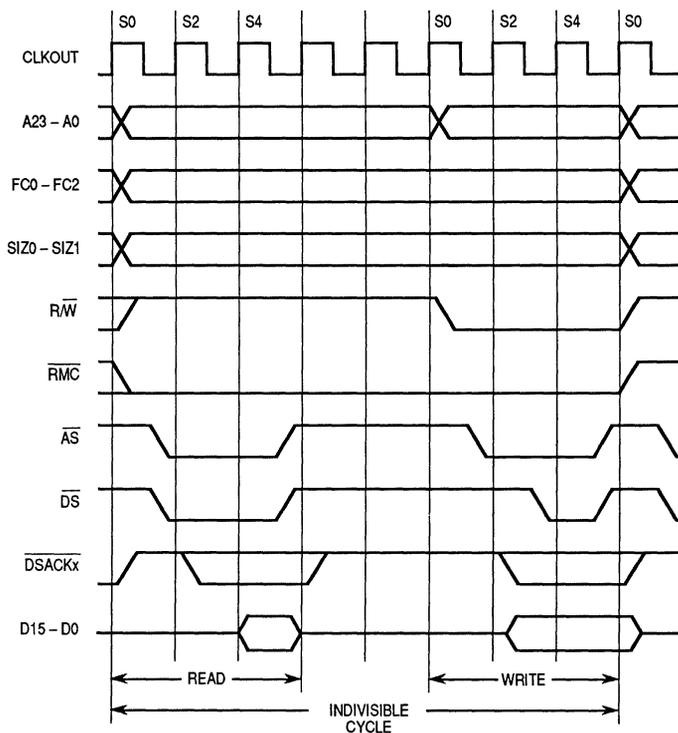
Figure 3-16. Read-Modify-Write Cycle Timing

Idle States — The MCU 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. States 0–5 are omitted if no write cycle is required. If a write cycle is required, the R/$\overline{W}$ signal remains in the read mode until state 0 to prevent bus conflicts

with the preceding read portion of the cycle; the data bus is not driven until state 2.

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

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

State 2 — During S2, the MCU places the data to be written onto the data bus (D15–D0).

State 3 — The MCU asserts the $\overline{DS}$ during S3. This indicates that the data is stable 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 MCU 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 MCU continues to sample $\overline{DSACKx}$ signals on the falling edges of the clock until one is recognized. The selected device uses R/$\overline{W}$, $\overline{DS}$, SIZ1–SIZ0, and A0 to latch data from the appropriate section(s) of the data bus (D15–D8 and D7–D0). The size signals and A0 select the data bus sections. If $\overline{DSACKx}$ is not already asserted, the device asserts $\overline{DSACKx}$ when it has successfully stored the data.

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

State 5 — The MCU 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/$\overline{W}$ and FC2–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 $\overline{DSACKx}$ 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 $\overline{DS}$.

3.4 CPU Space Cycles

The function codes (FC2–FC0) select user and supervisor program and data areas. The area selected by FC2–FC0 = $7 is classified as the CPU space. The interrupt acknowledge, breakpoint acknowledge, and low power stop

(LPSTOP) broadcast cycles described in the following paragraphs utilize CPU space. The CPU space type is encoded on address signals A19–A16 during a CPU space operation and indicates the function that the MCU is performing. On this MCU, three of the encodings are implemented as shown in Figure 3-17. All unused values are reserved by Motorola for future additional CPU space types.

CPU SPACE CYCLES

Figure 3-17. 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.

When a breakpoint instruction (BKPT) is executed, the MC68332 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 address lines A4–A2 and the T-bit (A1) set to zero. If this bus cycle is terminated by $\overline{\text{BERR}}$, the MCU then proceeds to perform illegal instruction exception processing. If the bus cycle is terminated by $\overline{\text{DSACKx}}$, the MCU uses the data on D15–D0 (for 16-bit ports) or two reads from D15–D8 (for 8-bit ports) to replace the BKPT instruction in the internal instruction pipeline, and begins execution of that instruction.

When the assertion of the BKPT pin is acknowledged by the CPU while background mode is not enabled, the MC68332 performs a word read from CPU space, type 0, at an address corresponding to all ones being set on address lines A4–A2 (BKPT #7) and the T-bit (A1) being set to 1. If this bus cycle is terminated by $\overline{\text{BERR}}$, the MCU then proceeds to perform hardware breakpoint exception processing. If the bus cycle is terminated by $\overline{\text{DSACKx}}$, the MCU

ignores any data on the data bus and continues execution of the next instruction.

NOTE

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

The breakpoint operation flow is shown in Figure 3-19. Figures 3-20 and 3-21 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 LPSTOP broadcast cycle is generated by the CPU executing the LPSTOP instruction. The external bus interface must get a copy of the interrupt mask level from the CPU, and so the CPU performs a CPU space type three write with the mask level encoded on the data bus, as shown in Figure 3-18. The CPU space type three cycle is shown externally if the bus is available as an indication to external devices that the MCU is going into low power stop mode. The SIM provides $\overline{\text{DSACK}}$ response to this cycle.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
—	—	—	—	—	—	—	—	—	—	—	—	—	I2	I1	I0

RESET:

| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

Figure 3-18. LPSTOP Interrupt Mask Level

BREAKPOINT OPERATION FLOW

PROCESSOR   EXTERNAL DEVICE

ACKNOWLEDGE BREAKPOINT

IF BREAKPOINT INSTRUCTION EXECUTED.
  1) SET R/$\overline{W}$ TO READ
  2) SET FUNCTION CODE TO CPU SPACE
  3) PLACE CPU SPACE TYPE 0 ON A19 – A16
  4) PLACE BREAKPOINT NUMBER ON A2 – A4
  5) SET T-BIT (A1), TO ZERO
  6) SET SIZE TO WORD
  7) ASSERT $\overline{AS}$ AND $\overline{DS}$

IF BKPT PIN ASSERTED·
  1) SET R/$\overline{W}$ TO READ
  2) SET FUNCTION CODE TO CPU SPACE
  3) PLACE CPU SPACE TYPE0 ON A19 – A16
  4) PLACE ALL ONES ON A4 – A2
  5) SET T-BIT (A1), TO ONE
  6) SET SIZE TO WORD
  7) ASSERT $\overline{AS}$ AND $\overline{DS}$

IF BKPT INSTRUCTON EXECUTED:
  1) PLACE REPLACEMENT OPCODE ON DATA BUS
  2) ASSERT $\overline{DSACKx}$
      OR.
  1) ASSERT $\overline{BERR}$ TO INITIATE EXCEPTION PROCESSING

IF BKPT ASSERTED:
  1) ASSERT $\overline{DSACKx}$
      OR:
  1)ASSERT $\overline{BERR}$ TO INITIATE EXCEPTION PROCESSING

IF BREAKPOINT INSTRUCTION EXECUTED AND
  $\overline{DSACKx}$ IS ASSERTED:
    1) LATCH DATA
    2) NEGATE $\overline{AS}$ AND $\overline{DS}$
    3) GO TO (A)

IF BKPT PIN ASSERTED AND
  $\overline{DSACKx}$ IS ASSERTED:
    1) NEGATE $\overline{AS}$ AND $\overline{DS}$
    2) GO TO (A)

IF $\overline{BERR}$ ASSERTED:
    1) NEGATE $\overline{AS}$ AND $\overline{DS}$
    2) GO TO (B)
    (A)   (B)

IF BKPT INSTRUCTION EXECUTED:
  1) PLACE LATCHED DATA IN INSTRUCTION PIPELINE
  2) CONTINUE PROCESSING

IF BKPT PIN ASSERTED:
  1) CONTINUE PROCESSING

1) NEGATE $\overline{DSACKx}$ or $\overline{BERR}$

IF BKPT INSTRUCTION EXECUTED:
  1) INITIATE ILLEGAL INSTRUCTION PROCESSING

IF BKPT PIN ASSERTED:
  1) INITIATE HARDWARE BREAKPOINT PROCESSING

Figure 3-19. Breakpoint Operation Flow
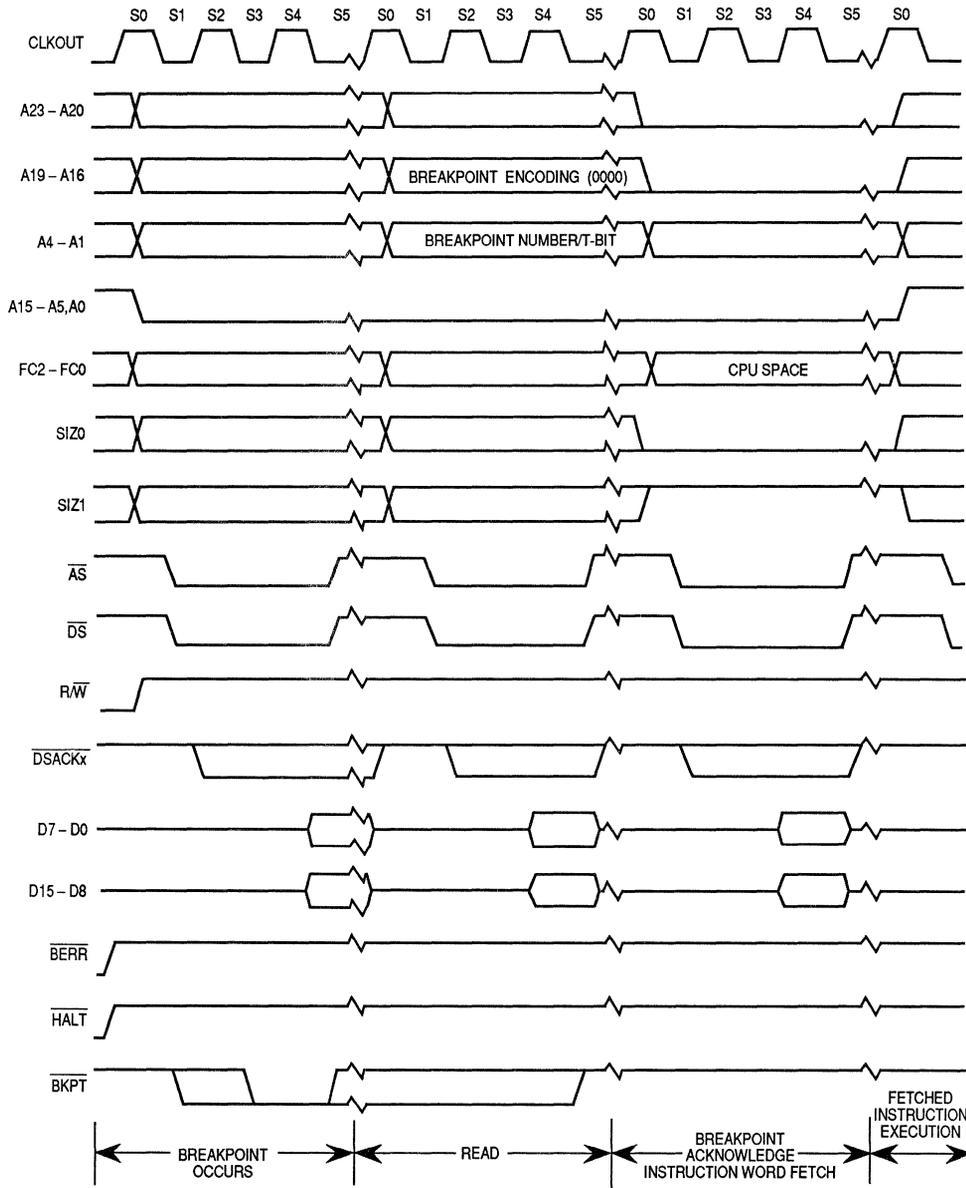
Figure 3-20. Breakpoint Acknowledge Cycle Timing (Opcode Returned)
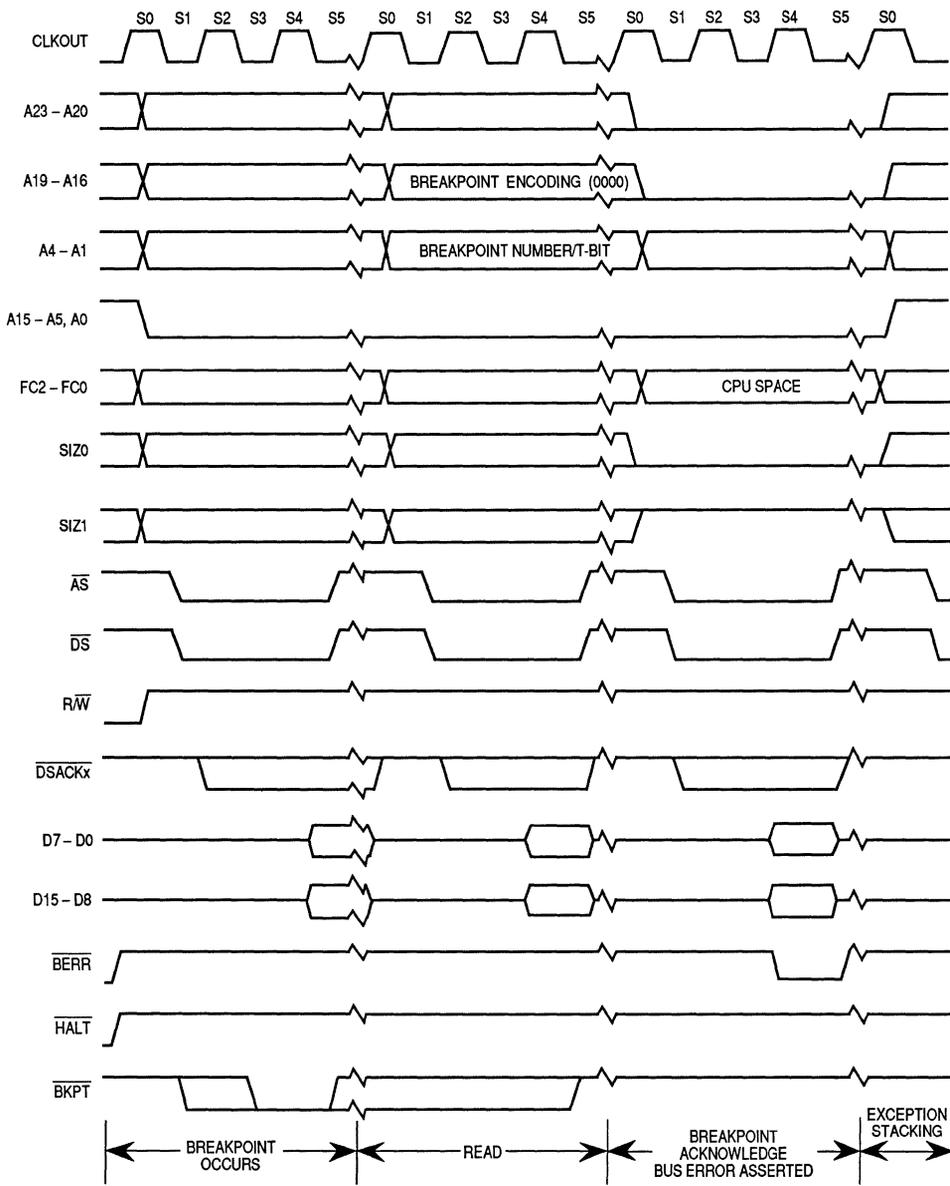
Figure 3-21. Breakpoint Acknowledge Cycle Timing (Exception Signaled)

3.4.3 Interrupt Acknowledge Bus Cycles

When a peripheral device signals the MCU (with the $\overline{\text{IRQ7}}$–$\overline{\text{IRQ1}}$ signals) 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 (or that a transition has occurred in the case of a level 7 interrupt), the MCU makes the interrupt a pending interrupt. The MCU 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 various kinds of interrupt acknowledge bus cycles that can be executed as part of interrupt exception processing.

3.4.3.1 Interrupt Acknowledge Cycle — Terminated Normally

When the MCU 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 cannot supply a vector number and use the autovector cycle described in **3.4.3.2 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:

  a. FC2–FC0 are set to $7 (FC2/FC1/FC0 = 111) for CPU address space.

  b. A3, A2, and A1 are set to the interrupt request level.

  c. The CPU space type field (address signals A19–A16) is set to $F, the interrupt acknowledge code.

  d. 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 D15–D8; for a 16-bit port the vector must be on D7–D0) during the interrupt acknowledge cycle. Beyond this, the cycle is terminated normally with $\overline{\text{DSACKx}}$. Figure 3-22 is the flowchart of the interrupt acknowledge cycle. Figure 3-23 shows the timing for an interrupt acknowledge terminated with $\overline{\text{DSACKx}}$.

3.4.3.2 Autovector Interrupt Acknowledge Cycle

When the interrupting device cannot supply a vector number, it requests an automatically generated vector, or 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. The $\overline{\text{DSACKx}}$ signals may not be asserted during an interrupt acknowledge cycle terminated by $\overline{\text{AVEC}}$. The vector number supplied in an autovector operation is derived from the interrupt level of the current interrupt. When the $\overline{\text{AVEC}}$ signal is asserted instead of $\overline{\text{DSACK}}$ during an interrupt acknowledge cycle, the MCU ignores the state of the data bus and internally generates the vector number, the sum of the interrupt level plus 24 ($18). There are seven distinct autovectors that can be used, corresponding to the seven levels of interrupt available with signals $\overline{\text{IRQ7}}$–$\overline{\text{IRQ0}}$. Figure 3-24 shows the timing for an autovector operation.

INTERRUPTING DEVICE   MC68332

REQUEST INTERRUPT   →   GRANT INTERRUPT

1) SYNCHRONIZE $\overline{\text{IRQ1}}$ – $\overline{\text{IRQ7}}$
2) COMPARE $\overline{\text{IRQ1}}$ – $\overline{\text{IRQ7}}$ TO MASK LEVEL AND
    WAIT FOR INSTRUCTION TO COMPLETE
3) PLACE INTERRUPT LEVEL ON A3-A1;
    TYPE FIELD (A19-A16) = $F
4) SET R/$\overline{\text{W}}$ TO READ
5) SET FC2-FC0 TO 111
6) DRIVE SIZE PINS TO INDICATE A ONE-BYTE
    TRANSFER
7) ASSERT $\overline{\text{AS}}$ AND $\overline{\text{DS}}$

PROVIDE VECTOR NUMBER   ←

1) PLACE VECTOR NUMBER ON LEAST
    SIGNIFICANT BYTE OF DATA BUS
2) ASSERT $\overline{\text{DSACKx}}$ (OR $\overline{\text{AVEC}}$ IF NO VECTOR NUMBER)

                    →   ACQUIRE VECTOR NUMBER

                         1) LATCH VECTOR NUMBER
RELEASE   ←   2) NEGATE $\overline{\text{DS}}$ AND $\overline{\text{AS}}$

1) NEGATE $\overline{\text{DSACKx}}$

                    →   START NEXT CYCLE

Figure 3-22. Interrupt Acknowledge Cycle Flowchart
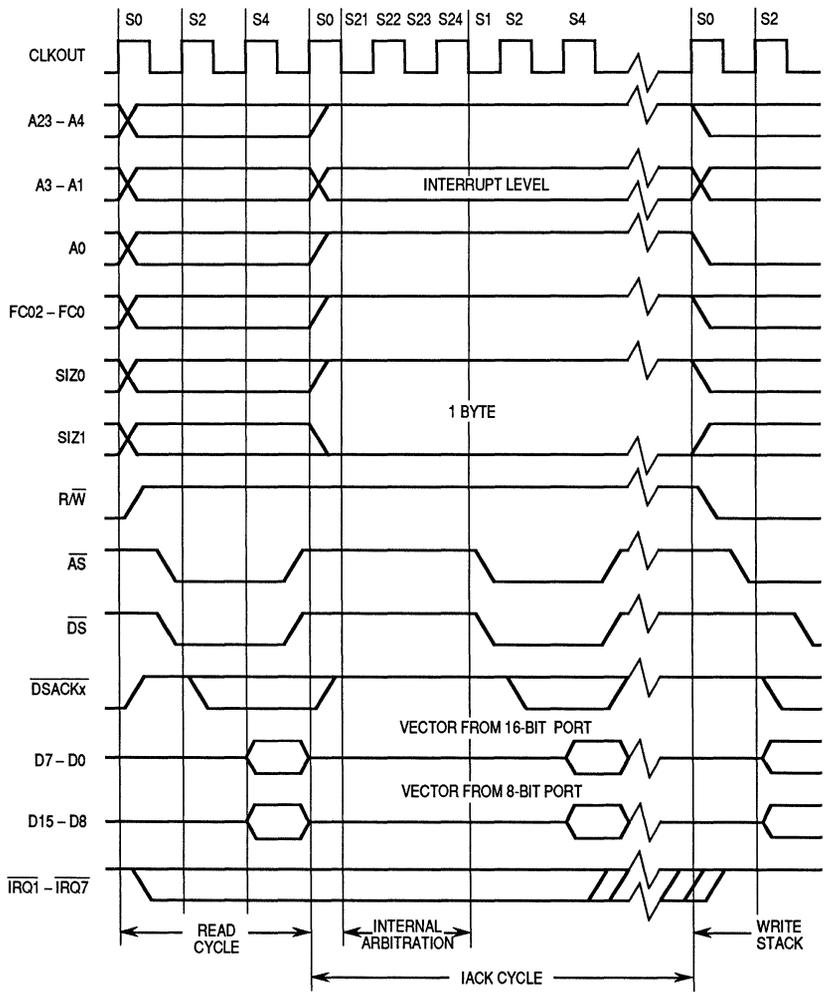
Figure 3-23. Interrupt Acknowledge Cycle Timing
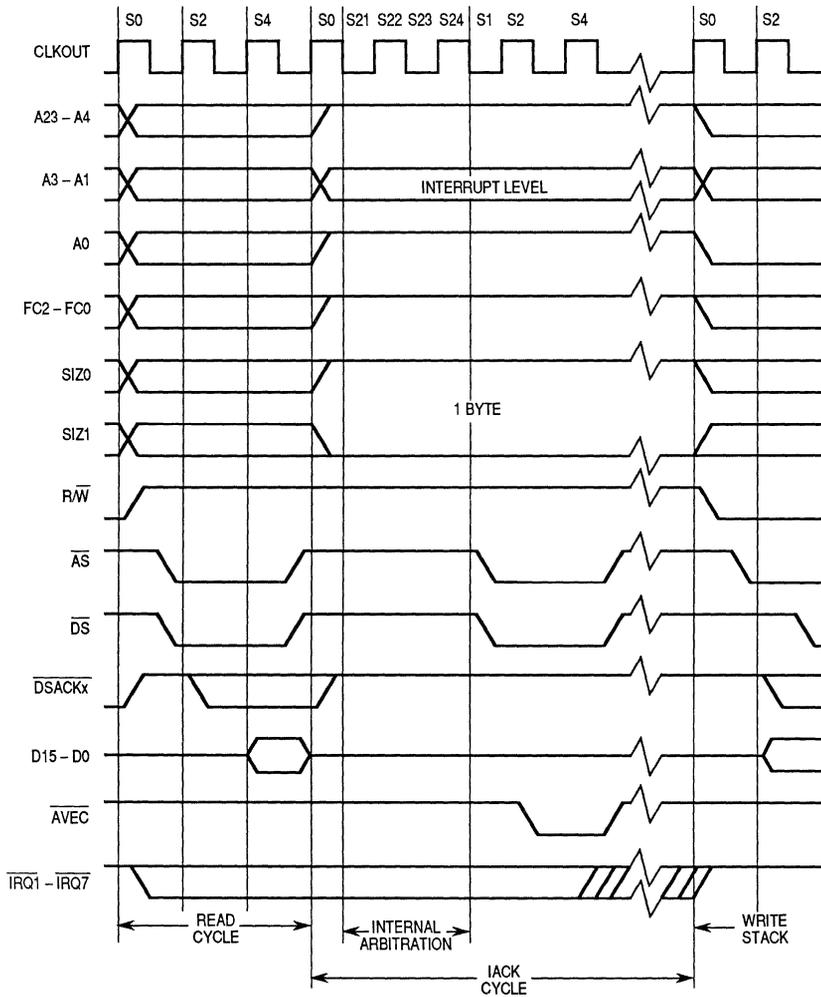
BUS OPERATION

Figure 3-24. Autovector Operation Timing

3.4.3.3 Spurious Interrupt Cycle

When a device (including the system integration module (SIM), which responds for external requests), does not respond during an interrupt acknowledge cycle by arbitrating for the interrupt acknowledge cycle, the spurious interrupt monitor generates an internal bus error signal to terminate the vector acquisition. The MCU automatically generates the spurious interrupt vector number 24 ($18) instead of the interrupt vector number in this case. When a 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 CPU taking the spurious interrupt vector. If the halt signal $\overline{\text{HALT}}$ is also asserted, the MCU retries the interrupt acknowledge cycle instead of using the spurious interrupt vector.

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. $\overline{\text{DSACKx}}$ or $\overline{\text{AVEC}}$ is not asserted in the following cases:

- The external device does not respond.
- No interrupt vector is provided.
- Various other application-dependent errors occur.

This MCU has a bus error input ($\overline{\text{BERR}}$) when no device responds by asserting $\overline{\text{DSACKx}}$ or $\overline{\text{AVEC}}$ within an appropriate period of time after the MCU asserts the $\overline{\text{AS}}$. This allows the cycle to terminate and the MCU to enter exception processing for the error condition. Another signal that is used for bus exception control is the halt signal ($\overline{\text{HALT}}$). 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 MCU clock. This assures that when two signals are asserted simultaneously, the required setup time and hold time for both of them are met for the same falling edge of the MCU clock. (Refer to **SECTION 10 ELECTRICAL CHARACTERISTICS** for timing requirements.) This or some equivalent pre-caution should be designed into the external circuitry that provides these signals, or else the internal bus monitor should 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 3-3).

Normal Termination
   $\overline{DSACKx}$ is asserted; $\overline{BERR}$ and $\overline{HALT}$ remain negated (case 1).

Halt Termination
   $\overline{HALT}$ is asserted at the same time, or before $\overline{DSACKx}$, and $\overline{BERR}$ remains negated (case 2).

Bus Error Termination
   $\overline{BERR}$ is asserted in lieu of, at the same time, or before $\overline{DSACKx}$ (case 3), or after $\overline{DSACKx}$ (case 4), and $\overline{HALT}$ remains negated; $\overline{BERR}$ is negated at the same time or after $\overline{DSACKx}$.

Retry Termination
   $\overline{HALT}$ and $\overline{BERR}$ are asserted in lieu of, at the same time, or before $\overline{DSACKx}$ (case 5) or after $\overline{DSACKx}$ (case 6); $\overline{BERR}$ is negated at the same time or after $\overline{DSACKx}$; $\overline{HALT}$ may be negated at the same time or after $\overline{BERR}$.

Table 3-3 shows various combinations of control signal sequences and the resulting bus cycle terminations. To ensure predictable operation, $\overline{BERR}$ and $\overline{HALT}$ should be negated according to the specifications in **SECTION 10 ELECTRICAL CHARACTERISTICS.** $\overline{DSACKx}$, $\overline{BERR}$, and $\overline{HALT}$ may be negated after $\overline{AS}$. *WARNING: If $\overline{DSACKx}$ or $\overline{BERR}$ remains 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 $\overline{BERR}$ after time out (case 3).

EXAMPLE B: A system uses error detection and correction on RAM contents. The designer may:

   a. Delay $\overline{DSACKx}$ until data is verified, and assert $\overline{BERR}$ and $\overline{HALT}$ simultaneously to indicate to the MCU to automatically retry the error cycle (case 5), or if data is valid assert $\overline{DSACKx}$ (case 1).

   b. Delay $\overline{DSACKx}$ until data is verified and assert $\overline{BERR}$ with or without $\overline{DSACKx}$ if data is in error (case 3). This initiates exception processing for software handling of the condition.

   c. Return $\overline{DSACKx}$ prior to data verification. If data is invalid, $\overline{BERR}$ is asserted on the next clock cycle (case 4). This initiates exception processing for software handling of the condition.

d. 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 3-3. $\overline{\text{DSACK}}$, $\overline{\text{BERR}}$, and $\overline{\text{HALT}}$ Assertion Results

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

NOTES:

N = The number of 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

The bus error signal 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 10 ELECTRICAL CHARACTERISTICS.** *WARNING: If $\overline{\text{BERR}}$ does not meet these constraints, it may cause unpredictable operation of the MCU. If $\overline{\text{BERR}}$ remains asserted into the next bus cycle, it may cause incorrect operation of that cycle.* When the bus error signal is issued to terminate a bus cycle, the MCU may enter exception processing immediately following the bus cycle, or it may 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 MCU 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 signal 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.

When the MCU recognizes a bus error condition, it terminates the current bus cycle in the normal way. Figure 3-25 shows the timing of a bus error for the case in which $\overline{\text{DSACKx}}$ are not asserted. Figure 3-26 shows the timing for a bus error that is asserted after $\overline{\text{DSACKx}}$. Exceptions are taken in both cases. (Refer to bus error exception in the CPU32 reference manual for details of bus error exception processing.)

In the second case where $\overline{\text{BERR}}$ is asserted after $\overline{\text{DSACKx}}$ is asserted, $\overline{\text{BERR}}$ must be asserted within the time specified (refer to **SECTION 10 ELECTRICAL CHARACTERISTICS**) 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. *WARNING: If $\overline{\text{BERR}}$ is not stable at this time, the MCU may exhibit erratic behavior.* $\overline{\text{BERR}}$ has priority over $\overline{\text{DSACKx}}$. In this case, data may be present on the bus, but may not be valid. This sequence may be used by systems that have memory error detection and correction logic and by external cache memories.
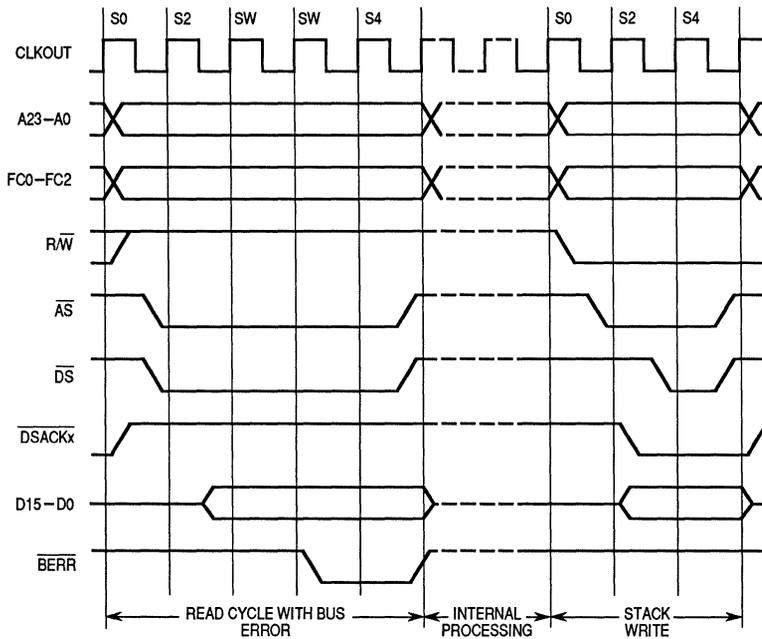
Figure 3-25. Bus Error without $\overline{\text{DSACKx}}$
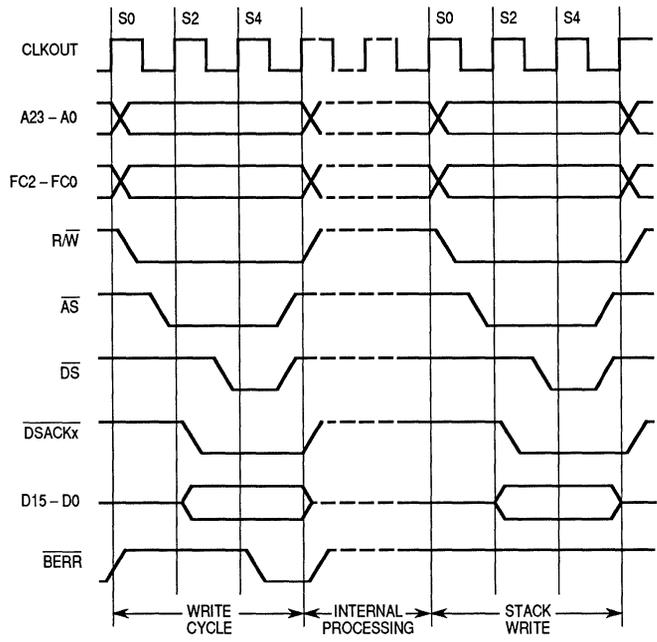
BUS OPERATION

Figure 3-26. Late Bus Error with DSACKx

3.5.2 Retry Operation

When the $\overline{\text{BERR}}$ and $\overline{\text{HALT}}$ signals are both asserted by an external device during a bus cycle, the MCU enters the retry sequence, shown in Figure 3-27. A delayed retry (Figure 3-28), similar to the delayed bus error signal described previously, can also occur. The MCU terminates the bus cycle, places the control signals in their inactive state and does not begin another bus cycle until the $\overline{\text{BERR}}$ and $\overline{\text{HALT}}$ signals are negated by external logic. After a synchronization delay, the MCU retries the previous cycle using the same access information (address, function code, size, etc.). The $\overline{\text{BERR}}$ signal should be negated before S2 of the read cycle to ensure correct operation of the retried cycle.

The MCU retries any read or write cycle of a read-modify-write operation separately; the $\overline{\text{RMC}}$ remains asserted during the entire retry sequence. Asserting the $\overline{\text{BR}}$ along with $\overline{\text{BERR}}$ and $\overline{\text{HALT}}$ provides a relinquish and retry operation. The MCU does not relinquish the bus during a read-modify-write operation. Any device that requires the MCU to give up the bus and retry a bus cycle during a read-modify-write cycle must assert $\overline{\text{BERR}}$ and $\overline{\text{BR}}$ only ($\overline{\text{HALT}}$ must not be included). The bus error handler software should examine the read-modify-

write bit in the special status word (refer to special status word in the CPU32 reference manual) and take the appropriate action to resolve this type of fault when it occurs.
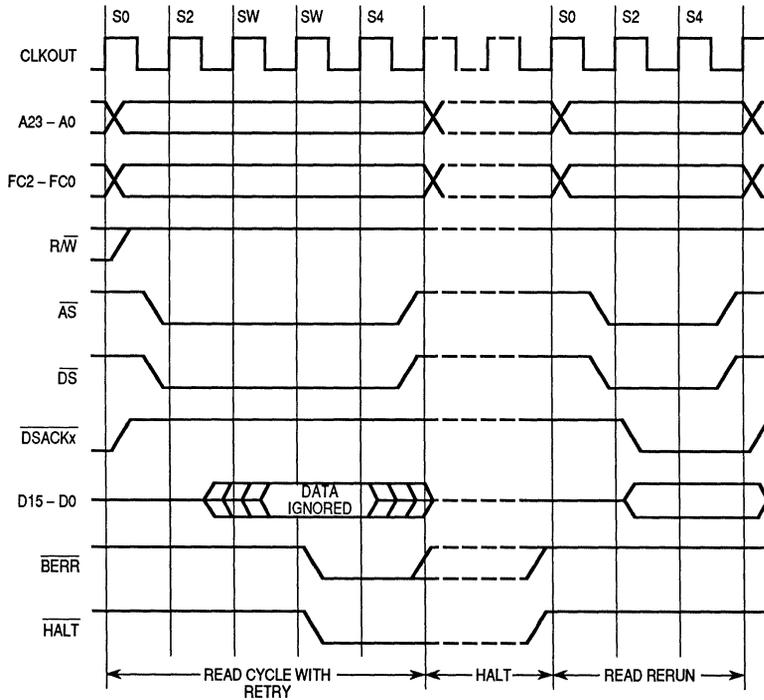
Figure 3-27. Retry Sequence
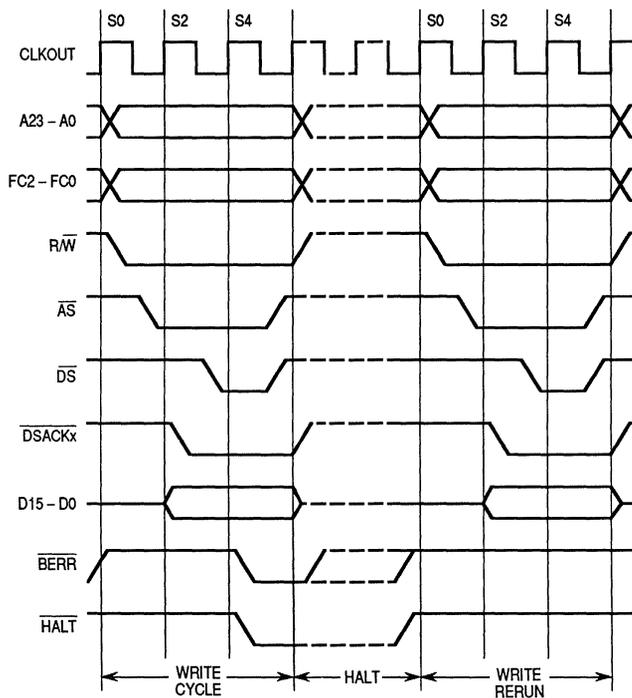
Figure 3-28. Late Retry Sequence

3.5.3 Halt Operation

When halt is asserted and $\overline{\text{BERR}}$ is not asserted, the MCU halts external bus activity after negation of $\overline{\text{DSACKx}}$. The MCU may complete the current word transfer in progress. In a long-word to 8-bit transfer, this could be after bus cycle 2 or 4 (refer to Figure 3.8). On a word to 8-bit transfer, the activity would stop after bus cycle 2 (refer to Fig. 3-6). Negating and reasserting $\overline{\text{HALT}}$ in accordance with the correct timing requirements provide a single-step (bus cycle to bus cycle) operation. The $\overline{\text{HALT}}$ signal affects external bus cycles only, so a program not requiring the use of external bus may continue executing, unaffected by $\overline{\text{HALT}}$ signal. Single-cycle mode allows the user to proceed through (and debug) external MCU operations a bus cycle at a time. Because occurrence of a bus error while $\overline{\text{HALT}}$ is asserted causes a retry operation, retry cycles must be anticipated while debugging in single-cycle mode. In dynamically sized 8-bit transfers, external bus activity may not stop at the next cycle boundry. 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. These MCU capabilities, with a software debugging package, provide debugging flexibility.

When the MCU completes a bus cycle with the $\overline{\text{HALT}}$ signal asserted, D15–D0 is placed in the high-impedance state, and bus control signals are driven inactive (not high-impedance state); the address, function code, size, and read/write signals remain in the same state. The halt operation has no effect on bus arbitration (refer to **3.6 Bus Arbitration**). When bus arbitration occurs while the MCU is halted, the address and control signals are also placed in the high-impedance state. Once bus mastership is returned to the MCU, if $\overline{\text{HALT}}$ is still asserted, the address, function code, size, and read/write signals are again driven to their previous states. The MCU does not service interrupt requests while it is halted.
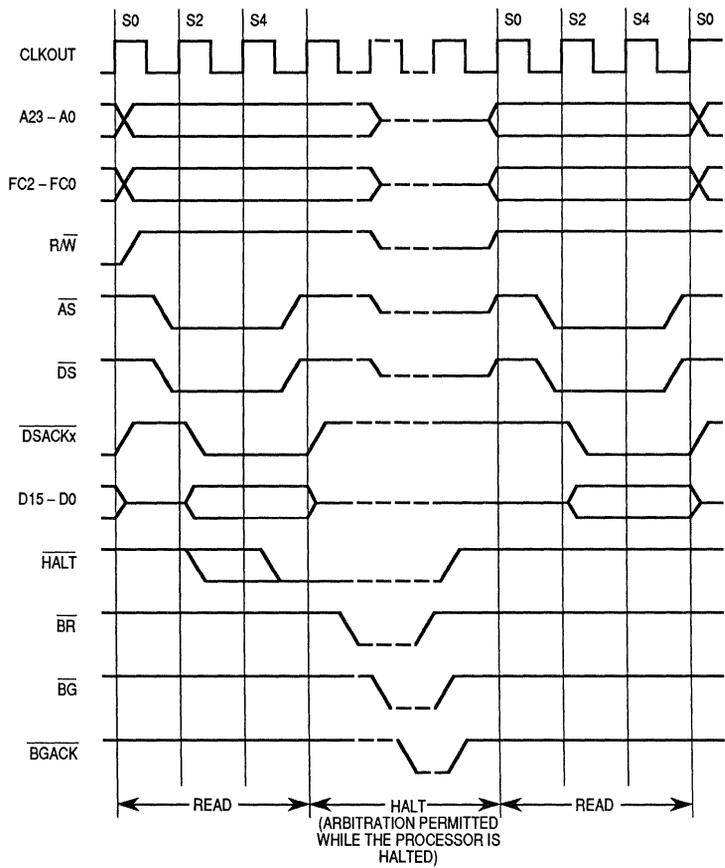
Figure 3-29. $\overline{\text{HALT}}$ Timing

3.5.4 Double Bus Fault

When a bus error or an address error occurs during the exception processing sequence for a previous bus error, a previous address error, a reset, or while the CPU is loading information from a bus error stack frame during a return from exception (RTE) instruction, a double bus fault occurs. For example, the MCU 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 MCU halts and drives the $\overline{\text{HALT}}$ line low. Only a reset operation can restart a halted MCU. However, bus arbitration can still occur (refer to **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 either. The MCU 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, described in **4.1.5.2 Halt Monitor.**

3.6 Bus Arbitration

The bus design of the MCU provides for a single bus master at any one time: either the MCU or an external device. One or more of the external devices on the bus has the capability to become bus master. Bus arbitration is the protocol by which an external device becomes bus master; the bus controller in the MCU manages the bus arbitration signals so that the MCU 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 that include 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:

a. An external device asserts the bus request signal ($\overline{\text{BR}}$);

b. The MCU asserts the bus grant signal to indicate that the bus is available ($\overline{\text{BG}}$);

c. The external device asserts the bus grant acknowledge ($\overline{\text{BGACK}}$) signal to indicate that it has assumed bus mastership.

$\overline{BR}$ may be issued any time during a bus cycle or between cycles. The bus grant ($\overline{BG}$) is asserted in response to $\overline{BR}$. *To guarantee operand coherency, $\overline{BG}$ is only asserted at the end of the 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. The external device asserts $\overline{BGACK}$ when it assumes bus mastership, 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:

- It must have received $\overline{BG}$ through the arbitration process.

- $\overline{BGACK}$ must be inactive, indicating that no other bus master has claimed ownership of the bus.

This technique allows processing of bus requests during data transfer cycles. Figure 3-30 is a flowchart showing the detail involved in bus arbitration for a single device. A timing diagram for the same operation is shown in **SECTION 10 ELECTRICAL CHARACTERISTICS.**

The flowchart shows that $\overline{BR}$ is negated at the time that $\overline{BGACK}$ is asserted. This type of operation applies to a system consisting of the MCU and one device capable of bus mastership. In a system having a number of devices capable of bus mastership, the bus request line from each device can be wire-ORed to the MCU. In such a system, more than one bus request could be asserted simultaneously. The timing diagram in **SECTION 10 ELECTRICAL CHARACTERISTICS** shows that $\overline{BG}$ is negated a few clock cycles after the transition of the bus grant acknowledge signal. However, if bus requests are still pending after the negation of bus grant, the MCU asserts another bus grant within a few clock cycles after it was negated. This additional assertion of bus grant allows external arbitration circuitry to select the next bus master before the current bus master has finished with 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 when the CPU has halted due to a double bus fault.
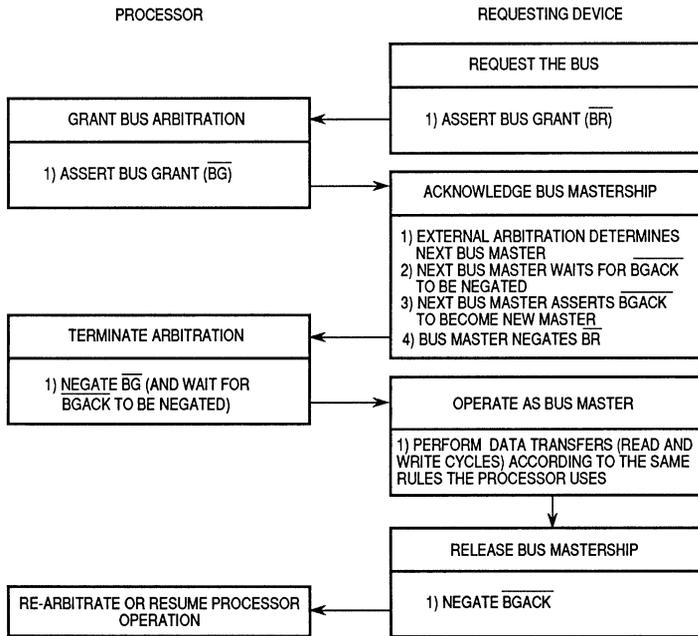
Figure 3-30. Bus Arbitration Flowchart for Single Request

3.6.1 Bus Request

External devices capable of becoming bus masters request the bus by asserting the $\overline{BR}$ signal. This can be a wire-ORed signal that indicates to the MCU that some external device requires control of the bus. The MCU 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 acknowledge is received while the bus request signal is active, the MCU remains bus master once the bus request is negated. This prevents unnecessary interference with ordinary processing if the arbitration circuitry inadvertently responds to noise or an external device determines that it no longer requires use of the bus before it has been granted mastership.

3.6.2 Bus Grant

This MCU supports operand coherency, and so if an operand transfer requires multiple bus cycles, the MCU does not release the bus until the entire transfer is complete. The assertion of bus grant is, therefore, subject to the following constraints:

- The minimum time for $\overline{BG}$ assertion after $\overline{BR}$ is asserted depends on internal synchronization and is specified in **SECTION 10 ELECTRICAL CHARACTERISTICS.**

- During an external operand transfer, the MCU does not assert $\overline{BG}$ until after the last cycle of the transfer (determined by the SIZx and $\overline{DSACKx}$ signals).

- During an external operand transfer, the MCU does not assert $\overline{BG}$ as long as $\overline{RMC}$ is asserted.

- If the show cycle bits are both asserted and the CPU is making internal accesses, the MCU does not assert $\overline{BG}$ until the CPU finishes the internal transfers. Otherwise, the external bus is granted away and the CPU continues to execute internal bus transfers.

Externally, the $\overline{BG}$ signal can be routed through a daisy-chained network or a priority-encoded network. The MCU 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 $\overline{BGACK}$ and it should not be negated until all bus cycles required 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 MCU assumes that another device is requesting the bus and prepares to issue another $\overline{BG}$.

Since external devices have priority over the MCU, the MCU cannot regain control of the external bus until all pending external bus requests have been satisfied.

NOTE

The assertion of $\overline{BGACK}$ alone does not cause $\overline{BG}$ to be asserted.

3.6.4 Bus Arbitration Control

The bus arbitration control unit in the MCU is implemented with a finite-state machine. As discussed previously, all asynchronous inputs to the MCU are internally synchronized in a maximum of two cycles of the clock. As shown in Figure 3-31, input signals labeled R and A are internally synchronized versions of the bus request and bus grant acknowledge signals, respectively. The bus grant 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 the $\overline{RMC}$ signal.

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 MCU immediately following a state change, when bus mastership is returned to the MCU. State 0, in which G and T are both negated, is the state of the bus arbiter while the MCU is bus master. Request R and acknowledge A keep the arbiter in state 0 as long as they are both negated.

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

3.6.5 Slave Mode Arbitration

Slave mode is used only for factory production testing of internal modules. It is not supported as a normal operating mode of th MCU. Applications using this MCU should avoid entering slave mode.

Slave mode is enabled when the SLVEN bit in the module configuration register (MCR) is set to one. This bit can only be set by driving $\overline{DB11}$ to a low state during reset. If SLVEN is set, the external bus interface (EBI) arbitrates for the internal bus whenever external bus arbitration occurs. When $\overline{BG}$ is asserted, the MCU is in slave mode. After an external device gains control of the internal bus, it has full access to all the internal registers, allowing an external master to replace the CPU and functionally test the internal modules.

$\overline{RA}$ + B

$\overline{G}\,\overline{T}$ V

STATE 0

$A\overline{B}$

$\overline{R}A$

$R\overline{A}\overline{B}$

$\overline{G}T\overline{V}$

STATE 3

$\overline{R}\overline{A}$

$\overline{R}\overline{A}$

$\overline{R}A$

$\overline{G}T\overline{V}$

STATE 2

R + A

$\overline{R}\overline{A}$+ A

GT$\overline{V}$

STATE 5

$\overline{R}$

R

R $\overline{A}$

$R\overline{A}$

GT$\overline{V}$

STATE 6

RA

R - BUS REQUEST
A - BUS GRANT ACKNOWLEDGE
B - BUS CYCLE IN PROGRESS

G - BUS GRANT
T - THREE-STATE SIGNAL TO BUS CONTROL
V - BUS AVAILABLE TO BUS CONTROL

NOTE: All figures are shown in positive logic (active high) regardless of their true active voltage level.
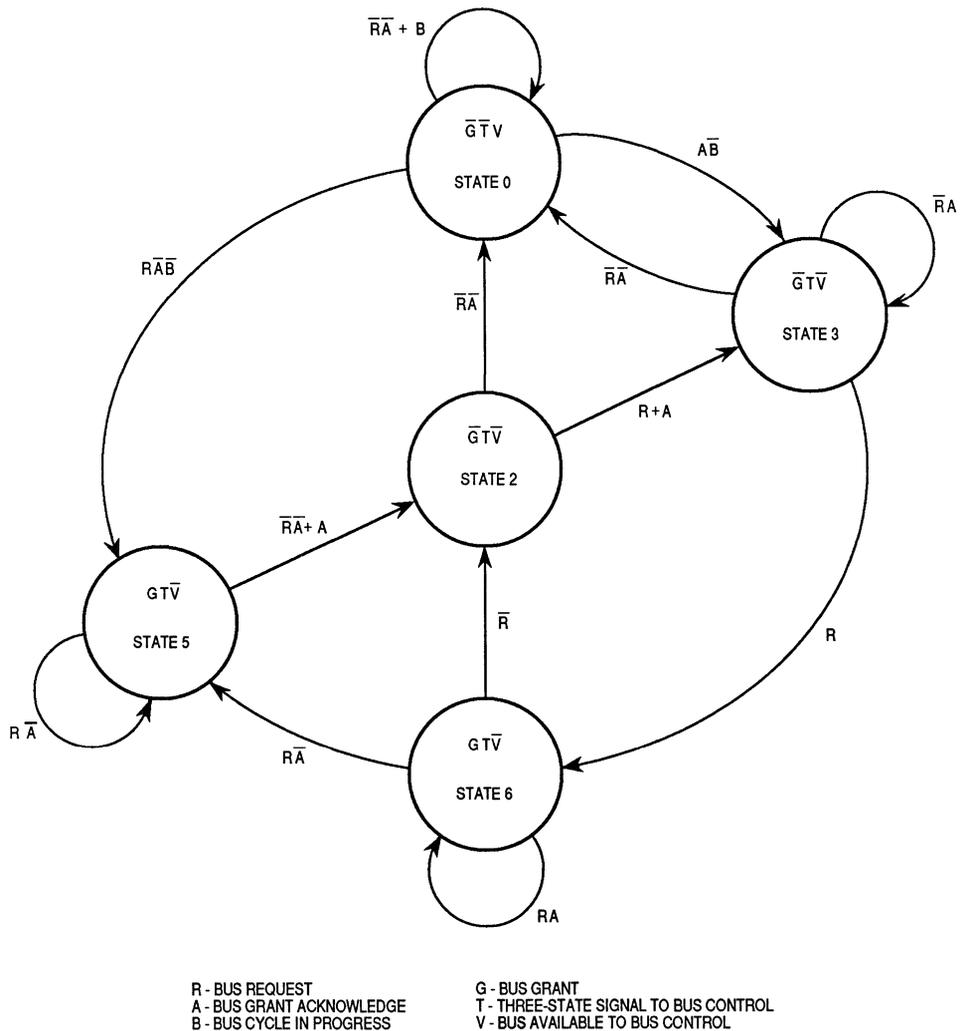
Figure 3-31. Bus Arbitration State Diagram

3.6.6 Show Cycles

The MCU 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 are called show cycles, and are distinguished by the fact that $\overline{AS}$ is not asserted externally.

After reset, show cycles are disabled and must be enabled by writing to the SHEN bits in the MCR. When show cycles are disabled, the address bus, function codes, size, and read/write signals continue to reflect internal bus activity, but $\overline{AS}$ and $\overline{DS}$ are not asserted externally and the external data bus is in a high-impedance state. When show cycles are enabled, $\overline{AS}$ is not asserted but $\overline{DS}$ is, and internal data is driven on the external data bus. Since internal cycles can continue to run when the external bus has been granted away, the SHEN bits allow the user to halt internal bus activity when the bus is granted away.

The following paragraphs are a state-by-state description of show cycles. Refer to **SECTION 10 ELECTRICAL CHARACTERISTICS** for specific timing information.

State 0 — During state 0, the address and function codes become valid, R/$\overline{W}$ is driven to indicate a show read or write cycle, and the size 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, $\overline{DS}$ is asserted to indicate that address information is valid.

State 42 — No action occurs in state 2. The bus controller remains in state 2 until the internal read cycle is complete.

State 43 — $\overline{DS}$ is negated to indicate that show data is valid on the next falling edge of 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 address, function codes, R/$\overline{W}$, and size pins change to begin the next cycle. Data from the preceding cycle is valid through state 0.

3.7 Reset Operation

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

a. EXTRST (external reset) drives the external reset pin.

b. CLKRST (clock reset) resets the clock module.

c. INTRST (internal reset) goes to all other internal circuits.

Table 3-4 summarizes the result of each reset source. Synchronous reset sources are not asserted until the end of the current bus cycle, whether $\overline{\text{RMC}}$ is asserted or not. The internal bus monitor is automatically enabled for synchronous resets, and so if the current bus cycle does not terminate normally, the bus monitor terminates it. Only single byte or aligned word writes are guaranteed valid for synchronous resets. External writes are also guaranteed to complete, provided the external configuration logic on the data bus is conditioned by R/$\overline{\text{W}}$ as shown in Figure 3-32. Asynchronous reset sources indicate a catastrophic failure, and the reset controller asserts reset to the system immediately.
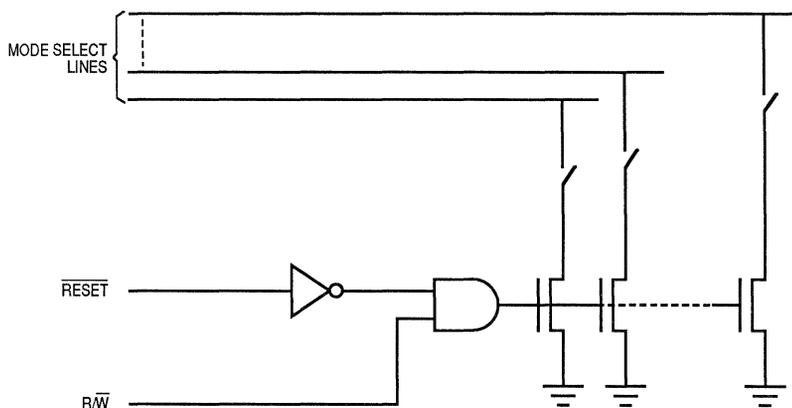
Figure 3-32. Mode Select Conditioning for Reset Operation

If an external device drives the $\overline{\text{RESET}}$ pin low, the reset control logic holds reset asserted internally until the external $\overline{\text{RESET}}$ is released. When the reset control logic detects that the external $\overline{\text{RESET}}$ is no longer being driven, it drives reset low for an additional 512 cycles to guarantee this length of reset to the entire system.

If reset is asserted from any other source, the reset control logic asserts $\overline{\text{RESET}}$ for a minimum of 512 cycles and until the source of reset is negated.

Table 3-4. Reset Source Summary

Type	Source	Timing	Reset Lines Asserted by Controller		
EXT (External)	External	Synchronous	INTRST	CLKRST	EXTRST
POW (Power-Up)	EBI	Asynchronous	INTRST	CLKRST	EXTRST
SW (Software Watchdog)	Sys Prot	Asynchronous	INTRST	CLKRST	EXTRST
HLT (Halt)	Sys Prot	Asynchronous	INTRST	CLKRST	EXTRST
LOC (Loss of Clock)	Clock	Synchronous	INTRST	CLKRST	EXTRST
TST (Test)	Test	Synchronous	INTRST		EXTRST
SYS (System)	CPU	Asynchronous			EXTRST

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

Figure 3-33 is a timing diagram of the power-up reset operation, showing the relationships between RESET, V$_{DD}$, and bus signals. During the reset period, the entire bus (except for non-three-statable signals, which are driven to their inactive state) three-states. Once 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 RESET exception processing begins.

RESET should be asserted for at least 590 clock periods to ensure that the MCU resets. Asserting RESET for 10 clock periods is sufficient for resetting the MCU logic; the additional clock periods prevent a reset instruction from overlapping the external RESET signal. Resetting the MCU causes any bus cycle in progress to terminate as if DSACKx or BERR had been asserted. In addition, the MCU initializes registers appropriately for a reset exception.

When a reset instruction is executed, the MCU drives the RESET signal for 512 clock cycles. In this case, the MCU resets the external devices of the system, and the internal registers of the MCU are unaffected. The external devices connected to the RESET signal are reset at the completion of the reset instruction.

CLKOUT

VCO
LOCK

V$_{DD}$

$\overline{\text{RESET}}$

BUS
CYCLES

|←— 512 CLOCKS —→|←— ≤14 CLOCKS —→|

|←— BUS STATE —→|←———— ADDRESS AND ————→|←— 1 —→|← 2 →|← 3 →|← 4 —→|
       UNKNOWN   CONTROL SIGNALS
                          THREE-STATED
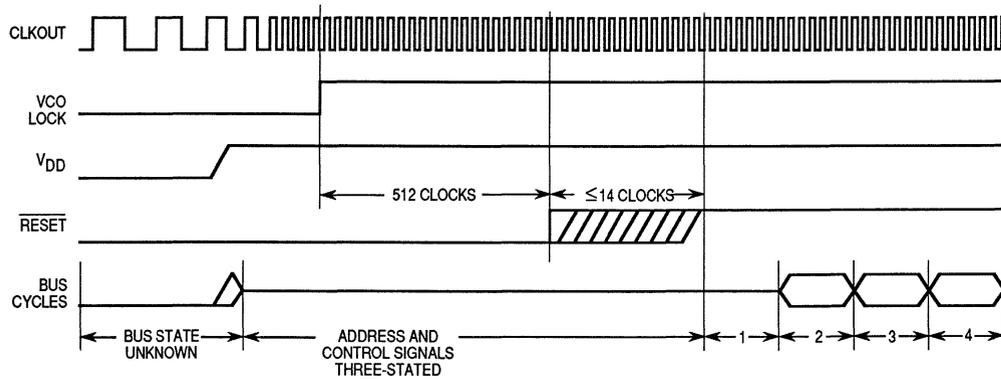
NOTES:
1. Internal start-up time
2. SSP read here
3. PC read here
4. First instruction fetched here

Figure 3-33. Initial Reset Operation Timing

SECTION 4
SYSTEM INTEGRATION MODULE

The MC68332 system integration module (SIM) consists of five submodules that control the microcontroller unit (MCU) system startup, initialization, configuration, and external bus with a minimum of external devices. The five submodules that make up the SIM, shown in Figure 4-1, are as follows:

- System Configuration and Protection
- Clock Synthesizer
- Chip Selects
- External Bus Interface
- System Test
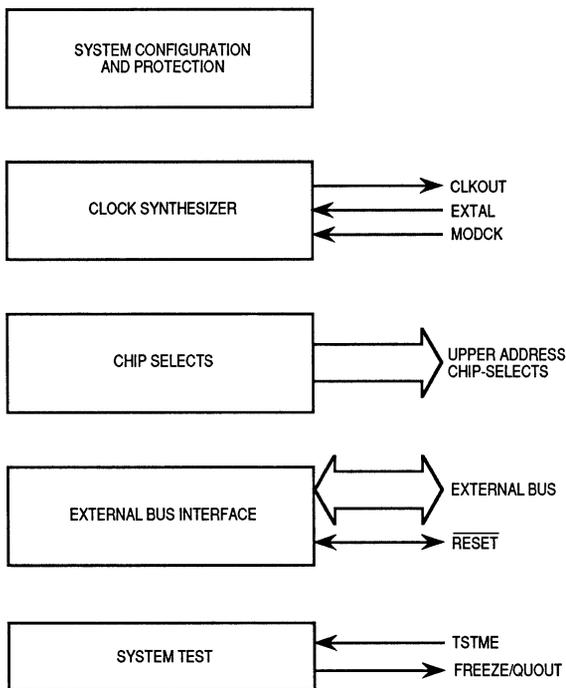
Figure 4-1. System Integration Module Block Diagram

The system configuration and protection submodule, described in **4.1 System Configuration and Protection Submodule**, controls system configuration and provides bus monitors and a software watchdog monitor for system protection.

The clock synthesizer, described in **4.2 Clock Synthesizer**, generates the clock signals used by the SIM as well as other modules and external devices.

The programmable chip-select submodule, described in **4.3 Chip Select Submodule,** provides 12 chip-select signals. Each chip-select signal has an associated base register and option register that contain the programmable characteristics of that chip select.

The external bus interface (EBI), described in **SECTION 3 BUS OPERATION,** handles the transfer of information between the internal CPU and memory, peripherals, or other processing elements in the external address space.

The system test submodule, described in **4.5 Test Submodule**, incorporates all the hardware necessary for testing the MCU, using scan-based testing. The system test submodule is used to perform factory tests, and its use in normal applications is not supported.

Figure 4-2 is a map of all registers in the SIM, and corresponding function codes (FC) that select user and supervisor data spaces.

FC	ADDRESS	15 8 7 0	
101	YFFA00	MODULE CONFIGURATION (MCR)	
101	YFFA02	MODULE TEST (SIMTR)	TEST
101	YFFA04	CLOCK SYNTHESIZER CONTROL (SYNCR)	CLOCK
101	YFFA06	UNUSED	RESET STATUS REGISTER (RSR)	
101	YFFA08	MODULE TEST E (SIMTRE)	EBI
101	YFFA0A	UNUSED	UNUSED	
101	YFFA0C	UNUSED	UNUSED	
101	YFFA0E	UNUSED	UNUSED	
X01	YFFA10	UNUSED	PORTE DATA (PORTE)	EBI
X01	YFFA12	UNUSED	PORTE DATA (PORTE)	
X01	YFFA14	UNUSED	PORTE DATA DIRECTION (DDRE)	
101	YFFA16	UNUSED	PORTE PIN ASSIGNMENT (PEPAR)	
X01	YFFA18	UNUSED	PORTF DATA (PORTF)	EBI
X01	YFFA1A	UNUSED	PORTF DATA (PORTF)	EBI
X01	YFFA1C	UNUSED	PORTF DATA DIRECTION (DDRF)	
101	YFFA1E	UNUSED	PORTF PIN ASSIGNMENT (PFPAR)	EBI
101	YFFA20	UNUSED	SYSTEM PROTECTION CONTROL (SYPCR)	SYS-PROTECT

Figure 4-2. SIM Register Map (Sheet 1of 2)

FC	ADDRESS	15 8	7 0	
101	YYFA22	PERIODIC INTERRUPT CONTROL (PICR)		MOD CONF
101	YFFA24	PERIODIC INTERRUPT TIMING (PITR)		MOD CONF
101	YFFA26	UNUSED	SOFTWARE SERVICE (SWSR)	SYS-PROTECT
101	YFFA28	UNUSED	UNUSED	
101	YFFA30	TEST MODULE MASTER SHIFT A (TSTMSRA)		TEST
101	YFFA32	TEST MODULE MASTER SHIFT B (TSTMSRB)		
101	YFFA34	TEST MODULE SHIFT COUNT.A.(TSTSCA)	TEST MODULE SHIFT COUNT.B.(TSTSCB)	
101	YFFA36	TEST MODULE REPETITION COUNTER (TSTRC)		
101	YFFA38	TEST MODULE CONTROL (CREG)		
X01	YFFA3A	TEST MODULE DISTRIBUTED REGISTER (DREG)		TEST
	YFFA3C	UNUSED	UNUSED	
	YFFA3E	UNUSED	UNUSED	
X01	YFFA40	UNUSED	PORT C DATA (CSPDR)	CHIP-SELECT
X01	YFFA42	UNUSED	UNUSED	
101	YFFA44	CHIP-SELECT PIN ASSIGNMENT (CSPAR0)		
101	YFFA46	CHIP-SELECT PIN ASSIGNMENT (CSPAR1)		
101	YFFA48	CHIP-SELECT BASE BOOT (CSBARBT)		
101	YFFA4A	CHIP-SELECT OPTION BOOT (CSORBT)		
101	YFFA4C	CHIP-SELECT BASE 0 (CSBAR0)		
101	YFFA4E	CHIP-SELECT OPTION 0 (CSOR0)		
101	YFFA50	CHIP-SELECT BASE 1 (CSBAR1)		
101	YFFA52	CHIP-SELECT OPTION 1 (CSOR1)		
101	YFFA54	CHIP-SELECT BASE 2 (CSBAR2)		
101	YFFA56	CHIP-SELECT OPTION 2 (CSOR2)		
101	YFFA58	CHIP-SELECT BASE 3 (CSBAR3)		
101	YFFA5A	CHIP-SELECT OPTION 3 (CSOR3)		
101	YFFA5C	CHIP-SELECT BASE 4 (CSBAR4)		
101	YFFA5E	CHIP-SELECT OPTION 4 (CSOR4)		
101	YFFA60	CHIP-SELECT BASE 5 (CSBAR5)		
101	YFFA62	CHIP-SELECT OPTION 5 (CSOR5)		
101	YFFA64	CHIP-SELECT BASE 6 (CSBAR6)		
101	YFFA66	CHIP-SELECT OPTION 6 (CSOR6)		
101	YFFA68	CHIP-SELECT BASE 7 (CSBAR7)		
101	YFFA6A	CHIP-SELECT OPTION 7 (CSOR7)		
101	YFFA6C	CHIP-SELECT BASE 8 (CSBAR8)		
101	YFFA6E	CHIP-SELECT OPTION 8 (CSOR8)		
101	YFFA70	CHIP-SELECT BASE 9 (CSBAR9)		
101	YFFA72	CHIP-SELECT OPTION 9 (CSOR9)		
101	YFFA72	CHIP-SELECT OPTION 9 (CSOR9)		
101	YYFA74	CHIP-SELECT BASE 10 (CSBAR10)		
101	YFFA76	CHIP SELECT OPTION 10 (CSOR10)		CHIP-SELECT
	YFFA78	UNUSED	UNUSED	
	YFFA7A	UNUSED	UNUSED	
	YFFA7C	UNUSED	UNUSED	
	YFFA7E	UNUSED	UNUSED	

X = Depends on state of SUPV bit in SIM MCR

Y = m111, where m is the modmap bit in the SIM MCR (Y = $7or $F)

Figure 4-2. SIM Register Map (Sheet 2 of 2)

4.1 System Configuration and Protection Submodule

The SIM module allows the user to control some features of system configuration by writing bits in the module configuration register, described in **4.1.1 Module Configuration Register**. This register also contains read-only status bits that show the state of some of the SIM features.

This MCU is designed with the concept of providing maximum system safeguards. Many of the functions that normally must be provided in external circuits are incorporated into this MCU. The features provided in the system configuration and protection submodule are as follows:

System Configuration
The module configuration register allows the user to configure the system according to the particular system requirements.

Internal Bus Monitor
The MCU provides an internal bus monitor to monitor the $\overline{\text{DSACKx}}$ response time for all internal bus accesses. An option allows the monitoring of internal to external bus accesses. There are four selectable response times that allow for the response speed of peripherals used in the system. A bus error ($\overline{\text{BERR}}$) signal is asserted internally if the $\overline{\text{DSACKx}}$ response time is exceeded. When operating as a bus master, the $\overline{\text{BERR}}$ signal is not asserted externally.

Halt Monitor
A halt monitor causes a reset to occur if the internal halt ($\overline{\text{HALT}}$) is asserted by the CPU.

Spurious Interrupt Monitor
If no interrupt arbitration occurs during an interrupt acknowledge (IACK) cycle, the $\overline{\text{BERR}}$ signal is asserted internally.

Software Watchdog
The watchdog asserts reset if the software fails to service the software watchdog for a designated period of time (presumably because it is trapped in a loop or lost). There are four selectable timeout periods, and a prescaler may be used for long timeout periods.

Periodic Interrupt Timer
The MCU provides a timer to generate periodic interrupts. The periodic interrupt time period can vary from 122 μs to15.94 s (with a 32.768-kHz crystal used to generate the system clock).

Figure 4-3 shows a block diagram of the system configuration and protection submodule.
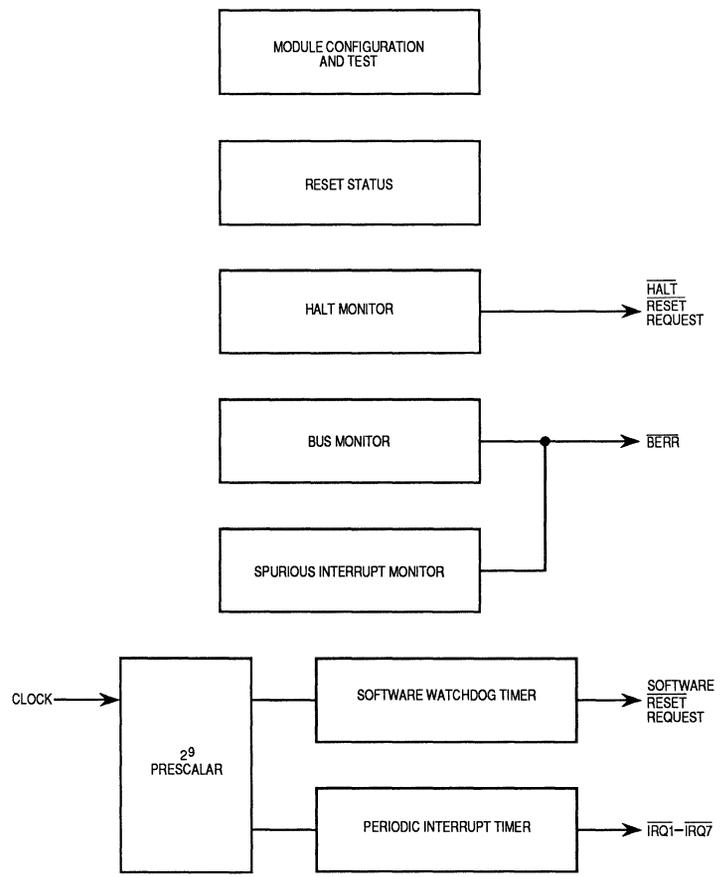
Figure 4-3. System Configuration and Protection Submodule

NOTE

In the registers discussed in the following paragraphs, the numbers in the top line of the register description represent the bit number in the register. The second line contains the mnemonic for the bit. The values shown under the mnemonic in the register diagram are the values of those register bits after reset.

4.1.1 Module Configuration Register (MCR)

The module configuration register controls the SIM configuration. Refer to Figure 4-4. The register can be both read and written at any time, except for module mapping (MM) (bit 6), which can only be written once.

MCR — Module Control Register $YFFA00

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
EXOFF	FRZSW	FRZBM	0	SLVEN	0	SHEN1	SHEN0	SUPV	MM	0	0	IARB3	IARB2	IARB1	IARB0

RESET
| 0 | 1 | 1 | 0 | $\overline{DB11}$ | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 1 | 1 |

Figure 4-4. Module Control Register

EXOFF — External Clock Off
  1 = The CLKOUT pin is placed in a high impedance state.
  0 = The CLKOUT pin is driven from an internal clock source.

FRZSW — 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 **4.1.9 Freeze Operation** for more information on freeze operation.

FRZBM — 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.

SLVEN — Slave Mode Enabled
  1 = Any external master winning control of the external bus also gains direct access to the internal peripherals.
  0 = The internal peripherals are not available to an external master. This bit is a read-only status bit that reflects the state of the data bus bit 11 (DB11) during reset.

SHEN1–0 — Show Cycle Enable

These two control bits determine what the EBI does with the external bus during internal transfer operations. A show cycle allows internal transfers to be externally monitored. Table 4-1 shows for all SHEN bit combinations whether show cycle data is driven externally or not, and also whether external bus arbitration can occur. External peripherals must not be enabled during show cycles to prevent bus conflicts.

Table 4-1. Show Cycle Control Bits

SHEN1	SHEN0	Action
0	0	Show cycles disabled, external arbitration enabled
0	1	Show cycles enabled, external arbitration disabled
1	0	Show cycles enabled, external arbitration enabled
1	1	Show cycles enabled, external arbitration enabled; internal activity is halted by a bus grant.

SUPV — Supervisor/Unrestricted Data Space

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

1 = Registers with access controlled by the SUPV bit are restricted to supervisor access only when FC2 = 1.

0 = Registers with access controlled by the SUPV bit are unrestricted (FC2 is a don't care).

MM — Module Mapping

1 = Internal modules are addressed from $FFF000—$FFFFFF, which is in the absolute short addressing range.

0 = Internal modules are addressed from $7FF000—$7FFFFF.

IARB3–IARB0 — Interrupt Arbitration Bits

Each module that generates interrupts, including the SIM, 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 can share the same IARB value. The reset value of IARB is $0, preventing the SIM from arbitrating during an IACK cycle and causing SIM interrupts to be discarded as spurious interrupts. The system software must initialize the IARB field to a value from $F (highest priority) to $1 (lowest priority).

4.1.2 System Integration Module Test Registers

The following paragraphs describe registers in the SIM used only for factory test purposes.

4.1.2.1 System Integration Module Test Register (SIMTR)

The system integration module test register bits are reserved for factory testing. This register can only be accessed in test mode, and *user access is strongly discouraged.* The mask number is accessible at any time. Refer to Figure 4-5.

SIMTR — System Integration Module Test Register $YYFA02

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
MASK						0	0	SOSEL1	SOSEL0	SHIRQ1	SHIRQ0	FBIT1	FBIT0	BWC1	BWC0

RESET.

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 4-5. System Integration Module Test Register

Mask — Revision Number for this Part
Bits 15–10 indicate a mask number for each specific revision or derivative of this part. These bits are read-only; a write has no effect.

SOSEL1–SOSEL0 — Scan Out Select
These bits define the output scan path or monitor point connected to master shift register B via the SCANB line, as shown in the following table.

Table 4-2. Scan-Out Select

SOSEL1	SOSEL0	Function
0	0	Internal IRQ6 (Scan In)
0	1	External IRQ6 (External Scan In)
1	0	Periodic Interrupt Zero Detect
1	1	Modulo Counter Clock Output

SHIRQ1–SHIRQ0 — Show Interrupt Request
These bits are used to force internal information to appear externally, as listed in the following table.

Table 4-3. Show Interrupt Request

SHIRQ1	SHIRQ0	Function
0	0	Off
0	1	Reserved
1	0	Show Internal IRQ
1	1	Show Internal Bus Signals

FBIT1–FBIT0 — Force Bits

These bits force selected test conditions, as listed in the following table.

Table 4-4. Force Bits

FBIT1	FBIT0	Function
0	0	Off
0	1	Software Watchdog Bypass
1	0	Software Watchdog Reset
1	1	Loss of Clock Reset

BWC1–BWC0 — Bandwidth Control Bits

These bits force selected test conditions for the phase locked loop, as listed in the following table.

Table 4-5. Bandwidth Control Bits

BWC1	BWC0	Function
0	0	Off
0	1	Narrow Bandwidth
1	0	Wide Bandwidth
1	1	Disable Filter

4.1.2.2 System Integration Module Test Register (E Clock) (SIMTRE)

This write-only register is reserved for factory testing. A write to this register in test mode forces the E-clock phase to synchronize with the sytem clock.

4.1.3 Reset Status Register (RSR)

The reset status register contains a bit for each reset source in the MCU. A bit set to one indicates the last type of reset that occurred, and only one bit can be set in the register. The reset status register is updated by the reset control logic when the MCU comes out of reset. This register can be read at any time; a write has no effect. For more information see **3.7 Reset Operation**.

RSR — Reset Status Register   **$YFFA07**

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

Figure 4-6. Reset Status Register

EXT — External Reset
    1 = The last reset was caused by an external signal.

POW — Power-Up Reset
    1 = The last reset was caused by the power-up reset circuit.

SW — Software Watchdog Reset
    1 = The last reset was caused by the software watchdog circuit.

HLT — Halt Monitor Reset
    1 = The last reset was caused by the system protection submodule halt monitor.

LOC — Loss of Clock Reset
    1 = The last reset was caused by a loss of frequency reference to the clock submodule. This reset can only occur if the reset enable (RSTEN) bit in the clock submodule is set and the voltage-controlled oscillator (VCO) is enabled.

SYS — System Reset
    1 = The last reset was caused by the CPU executing a reset instruction. The system reset does not load a reset vector or affect any internal CPU registers or SIM configuration registers, but does reset external devices and other internal modules.

TST — Test Submodule Reset
    1 = The last reset was caused by the test submodule.

4.1.4 System Protection Control Register (SYPCR)

The system protection control register controls the system monitors, the prescaler for the software watchdog clock, and the bus monitor timing.

In operating mode, this register may be written only once following a power-on or external reset, but can be read at any time. In test mode, this register is writable at any time.

SYPCR — System Protection Control Register $YFFA21

7	6	5	4	3	2	1	0
SWE	SWP	SWT1	SWT0	HME	BME	BMT1	BMT0

RESET:

1	$\overline{\text{MODCK}}$	0	0	0	0	0	0

Figure 4-7. System Protection Control Register

SWE — Software Watchdog Enable
    1 = Software watchdog enabled
    0 = Software watchdog disabled
See **4.1.6 Software Watchdog** for more information on the software watchdog.

SWP — Software Watchdog Prescale
    1 = Software watchdog clock prescaled by 512
    0 = Software watchdog clock not prescaled
This bit controls the value of the software watch dog prescaler as shown below. The reset value of this bit is affected by the state of the MODCK pin on the rising edge of reset, as shown in Table 4-6.

**Table 4-6. MODCK
Pin and SWP Bit**

MODCK	SWP
0	1
1	0

SWT1–SWT0 — Software Watchdog Timing

These bits control the divide ratio used to establish the timeout period for the software watchdog timer. The software timeout period is given by the following equation.

$$\text{Timeout Period} = 1/(\text{EXTAL Frequency/Divide Count}) \qquad (4\text{-}1)$$

or

$$\text{Timeout Period} = \text{Divide Count/EXTAL Frequency} \qquad (4\text{-}2)$$

The software timeout period shown in Table 4-7 gives the equation to derive the software watchdog timeout for any clock frequency, and the timeout periods are listed for a 32.768-kHz crystal used with the VCO and a 16.718-MHz external oscillator.

Table 4-7. Software Timeout Periods for Watchdog Timeout

Bits 6–4	Software Timeout Period	32.768-kHz Crystal Period	16.718-MHz External Clock Period
000	2^9/EXTAL Input Frequency	15.6 Milliseconds	30.6 Microseconds
001	2^{11}/EXTAL Input Frequency	62.5 Milliseconds	122.5 Microseconds
010	2^{13}/EXTAL Input Frequency	250 Milliseconds	490 Microseconds
011	2^{15}/EXTAL Input Frequency	1 Second	1.96 Microseconds
100	2^{18}/EXTAL Input Frequency	8 Seconds	15.6 Milliseconds
101	2^{20}/EXTAL Input Frequency	32 Seconds	62.7 Milliseconds
110	2^{22}/EXTAL Input Frequency	128 Seconds	250 Milliseconds
111	2^{24}/EXTAL Input Frequency	512 Seconds	1 Second

CAUTION

When the SWT1–SWT0 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 **4.1.6 Software Watchdog** for more information.

HME — Halt Monitor Enable
1 = Enable halt monitor function
0 = Disable halt monitor function
For more information refer to **4.1.5.2 Halt Monitor**.

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 refer to **4.1.5.1 Internal Bus Monitor**.

BMT — Bus Monitor Timing
> These bits select the timeout period for the bus monitor according to the following table.

Table 4-8. Bus Monitor Timing

Bits 1–0	Bus Monitor Timeout Period
00	64 System Clocks (CLK)
01	32 System Clocks (CLK)
10	16 System Clocks (CLK)
11	8 System Clocks (CLK)

4.1.5 Bus Monitors

The SIM provides a bus monitor that monitors all internal bus accesses and optionally monitors internal to external bus accesses for excessive response times.

4.1.5.1 Internal Bus Monitor

The internal bus monitor continually checks for the $\overline{\text{DSACKx}}$ response time during a normal bus cycle or the autovector $(\overline{\text{AVEC}})$ response time during an interrupt acknowledge (IACK) bus cycle. The monitor initiates $\overline{\text{BERR}}$ if the response time is excessive. The internal bus monitor does not check $\overline{\text{DSACKx}}$ response on the external bus unless it initiates the bus cycle. If the system contains external bus masters, an external bus monitor must also be implemented, and the internal to external bus monitor option disabled.

The $\overline{\text{DSACKx}}$ or $\overline{\text{AVEC}}$ response time is measured in clock cycles, and the maximum-allowable response time is programmable. Four selectable response-time periods for the bus monitor are listed in **4.1.4 System Protection Control Register** under the BMT bit description. These are provided to allow for the different response times of peripherals that might be used in the system.

The BME bit in the SYPCR enables the internal bus monitor for internal to external bus cycles.

4.1.5.2 Halt Monitor

The halt monitor responds to an assertion of $\overline{\text{HALT}}$ on the internal bus. Refer to **3.5.4 Double Bus Fault** for more information. A flag in the reset status register (RSR) indicates that the last reset was caused by the halt monitor. The halt monitor reset can be inhibited by the HME bit in the SYPCR.

4.1.5.3 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 submodules recognize that the CPU is responding to their own interrupt and arbitrate for the privilege of returning a vector or asserting $\overline{\text{AVEC}}$. *This feature cannot be disabled.*

4.1.6 Software Watchdog

Once enabled, the software watchdog requires a special service sequence to be executed on a periodic basis. If this periodic servicing action does not take place, the software watchdog times out and issues a reset. This protects the system against the possibility of the software becoming trapped in loops or running away. There are selectable watchdog timeout periods that are tabulated in the SYPCR description. The watchdog clock rate is affected by the SWP and SWT bits in the SYPCR, described in the SYPCR bit descriptions.

Figure 4-8 shows a block diagram of the watchdog timer and the clock control for the periodic interrupt timer.
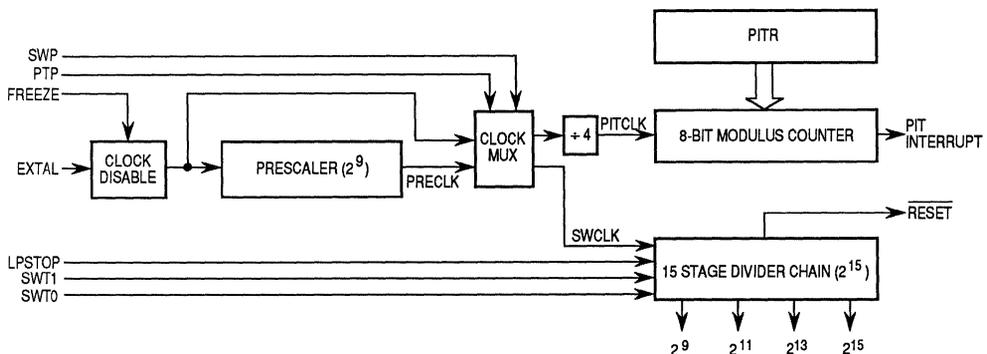
Figure 4-8. Watchdog Timer

The software watchdog service sequence consists of the following two steps:

1. Write $55 to the software service register (SWSR). Refer to Figure 4-9.
2. Write $AA to the SWSR.

Both writes must occur in the order listed prior to the watchdog timeout, but any number of instructions can be executed between the two writes.

4

SWSR — Software Service Register   **$YFFA27**

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
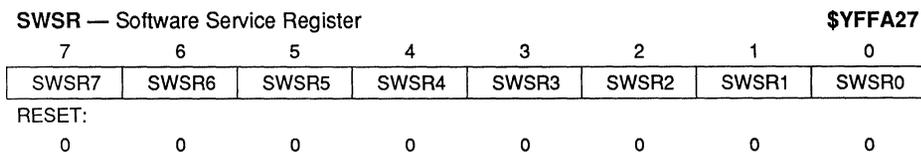
Figure 4-9. Software Service Register

The SWSR is the location the watchdog timer servicing sequence is written. This register can be written at any time, but returns all zeros when read.

The software watchdog can be enabled or disabled by the SWE bit in the SYPCR.

4.1.7 Periodic Interrupt Timer (PITR)

The periodic interrupt timer consists of an 8-bit modulus counter that is loaded with the value contained in the PITR, described below.

PITR — Periodic Interrupt Timing Register   **$YFFA24**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	PTP	PITR7	PITR6	PITR5	PITR4	PITR3	PITR2	PITR1	PITR0

RESET:

0	0	0	0	0	0	0	$\overline{MODCK}$	0	0	0	0	0	0	0	0
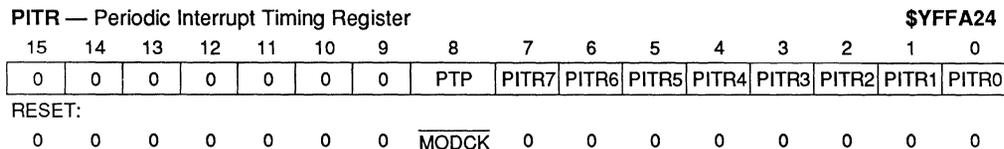
Figure 4-10. Periodic Interrupt Timing Register

The periodic timer prescale bit (PTP) contains the prescaler control for the periodic timer.

PTP — Periodic Timer Prescaler Control

    1 = Periodic timer clock prescaled by a value of 512
    0 = Periodic timer clock not prescaled
The reset value of this bit is affected by the state of the clock mode select (MODCK) pin on the rising edge of reset, as shown in the following table.

Table 4-9. MODCK Pin and PTP Bit

MODCK	PTP
0	1
1	0

PITR7–PITR0 — PITM Field (Periodic Interrupt Timing Modulus)

    The periodic interrupt timing register (PITR) contains the count value for the periodic timer. A zero value turns off the periodic timer. This register can be read or written at any time.

    Figure 4-8 shows a block diagram of the clock control circuits for the periodic interrupt timer as well as the watchdog timer. The modulus counter is clocked by a signal derived from the buffered crystal oscillator (EXTAL) input pin unless an external frequency source is used. When an external frequency source is used (MODCK low at the end of reset), the default state of the prescaler control bits (SWP and PTP) is changed to enable both prescalers.

    Either clock source (EXTAL or EXTAL ÷ 512) is divided by four before driving the modulus counter (PITCLK). When the modulus counter value reaches zero, an interrupt is generated. The value in the PITR (the PITM field) 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.

    This register can be read or written at any time. Bits 15–9 are not implemented, and always return zero when read. A write does not affect these bits.

4.1.7.1 Periodic Interrupt Control Register (PICR)

    The periodic interrupt control register sets up the interrupt request level for the interrupt periodic timer and also contains the interrupt vector generated during an IACK cycle in response to an interrupt from the periodic timer.

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 |

Figure 4-11. Periodic Interrupt Control Register

PIRQL2–PIRQL0 — Periodic Interrupt Request Level

These bits contain the periodic interrupt request level. Table 4-10 shows what 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. Periodic Interrupt Request Level

Bits 10–8	Interrupt Request Level
000	Periodic Interrupt Disabled
001	Interrupt Request Level 1
010	Interrupt Request Level 2
011	Interrupt Request Level3
100	Interrupt Request Level 4
101	Interrupt Request Level 5
110	Interrupt Request Level 6
111	Interrupt Request Level 7

PIV7–PIV0 — Periodic Interrupt Vector

These bits contain the value of the vector generated during an IACK cycle in response to an interrupt from the periodic timer. When the SIM 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.

Bits 10–0 can be read or written at any time. Bits 15–11 are unimplemented and always return zero. A write to these bits has no effect.

4.1.7.2 Periodic Timer Period Calculation

The period of the periodic timer can be calculated using the following equation:

$$\text{PIT Period} = \text{PITM}/(\text{EXTAL}/\text{Prescaler})/4 \qquad (4\text{-}3)$$

where:
  PIT Period = Periodic Interrupt Timer Period
  PITM = Periodic Interrupt Timer Register Modulus (PITR7–PITR0)
  EXTAL = Crystal Frequency
  Prescaler = 512 or 1 depending on the state of the PTP bit in the PITR

Solving the equation using a crystal frequency of 32.768 kHz with the prescaler disabled (PTP = 0) gives:

$$\text{PIT Period} = \text{PITM}/(32768/1)/4 \qquad (4\text{-}4)$$
$$= \text{PITM}/8192$$

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

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

$$\text{PIT Period} = \text{PITM}/(32768/512)/4 \qquad (4\text{-}5)$$
$$= \text{PITM}/16$$

This gives a range from 62.5 ms with a PITM value of $01 (00000001 binary) to 15.94 s with a PITM of $FF (11111111 binary).

Table 4-11 lists some of the periods available using a 32.768-kHz clock:

Table 4-11. PIT Periods for 32.768-kHz Clock

PITR	PIT Period
$0000	Periodic Interrupt Timer Off
$0001	122 Microseconds
$0002	244 Microseconds
$0004	488 Microseconds
$0008	977 Microseconds
$000F	1.83 Milliseconds
$0020	3.90 Milliseconds
$0040	7.88 Milliseconds
$0080	15.6 Milliseconds
$00A0	19.5 Milliseconds
$00FF	31.1 Milliseconds
$0100	Periodic Interrupt Timer Off
$0101	62.5 Milliseconds
$0102	125 Milliseconds
$0104	250 Milliseconds
$0108	500 Milliseconds
$0110	1 Second
$0120	2 Seconds
$0140	4 Seconds
$0180	8 Seconds
$01A0	10 Seconds
$01FF	15.9 Seconds

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

With prescaler disabled:

$$\text{PIT Period} = \text{PITM} \ (122 \ \mu s) \tag{4-6}$$

With prescaler enabled:

$$\text{PIT Period} = \text{PITM} \ (62.5 \ ms) \tag{4-7}$$

4.1.7.3 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 1-s period. Rearranging the periodic timer period equation to solve for the desired count value,

$$PITM = (PIT\ Period)(EXTAL) / (Prescaler)(4) \qquad (4\text{-}8)$$
$$= (1)(32768)/(512)(4)$$
$$= 16\ (decimal)$$

Therefore, the PITR should be loaded with a value of $10, with the prescaler enabled, to generate interrupts at a 1-s rate.

4.1.8 Low Power STOP Operation (LPSTOP)

Execution of the LPSTOP instruction disables the clock to the software watchdog timer in the low state. The software watchdog timer remains stopped until the LPSTOP state is ended and then begins to run again on the next rising clock edge.

NOTE

When the CPU executes the STOP instruction (as opposed to LPSTOP), the software watchdog timer continues to run. If the software watchdog is enabled, it resets the MCU when timeout occurs.

The periodic interrupt timer does not respond to an LPSTOP instruction so that it can be used to bring the MCU out of the LPSTOP condition as long as the interrupt request level is higher than the CPU interrupt mask level. The periodic interrupt timer is clocked by the EXTAL clock, and so runs at the same frequency as the EXTAL pin during LPSTOP.

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, halt monitor, and spurious interrupt monitor are all inactive during LPSTOP.

4.1.9 Freeze Operation

FREEZE is asserted by the CPU if a breakpoint is encountered with background mode enabled. When FREEZE is asserted, only the bus monitor, software watchdog, and periodic interrupt timer are affected. The halt monitor and spuri-

ous interrupt monitor continue to operate normally. Setting the FRZBM bit disables the bus monitor when FREEZE is asserted, and setting the FRZSW bit disables the software watchdog and the periodic interrupt timer when FREEZE is asserted. The FRZBM and FRZSW bits are located in the MCR.

4.2 Clock Synthesizer

The clock synthesizer can operate from an on-chip phase-locked loop (PLL) using an external crystal connected between the EXTAL and XTAL pins as a reference frequency source. A 32.768-kHz watch crystal provides an inexpensive reference, but the reference crystal frequency can be any frequency from 25 to 50 kHz. Outside off this range, an external oscillator can be used with the on-chip frequency synthesizer and VCO, or the system clock frequency can be driven directly into the EXTAL pin (the XTAL pin should be left floating for this case).

The system clock frequency is programmable from 131 kHz to the maximum clock frequency, specified in **SECTION 10 ELECTRICAL CHARACTERISTICS**, with a resolution of 131 kHz.

A separate power pin (V_{DDSYN}) is used to allow the clock circuits to run with the rest of the MCU powered down and to provide increased noise immunity for the clock circuits. The source for V_{DDSYN} should be a quiet power supply with adequate external bypass capacitors placed as close as possible to the V_{DDSYN} pin to ensure a stable operating frequency. Figure 4-12 shows a block diagram of the clock submodule and suggested values for the bypass and PLL external capacitors.

The PLL requires an external low-leakage filter capacitor, typically in the range from 0.01 to 0.1 μF, connected between the external filter capacitor (XFC) and V_{DDSYN} pins. Smaller values of the external filter capacitor provide a faster response time for the PLL, and larger values provide greater frequency stability.

NOTES·
  1. Must be low-leakage capacitor.
  2. EXTAL can be driven with an external oscillator.
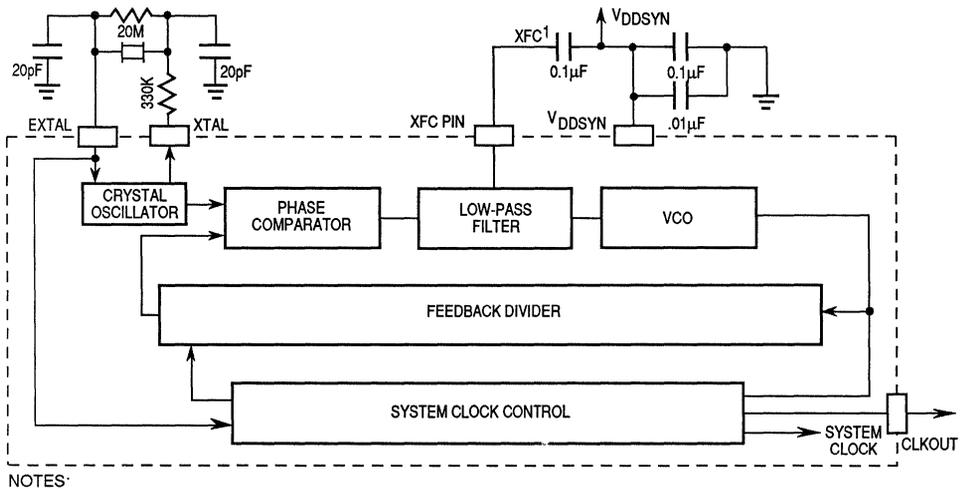
Figure 4-12. Clock Submodule Block Diagram

4.2.1 Clock Synthesizer Control Register (SYNCR)

The clock synthesizer control register can be read or written only in supervisor mode. The reset state of SYNCR produces an operating frequency of 8.38 MHz when the PLL is referenced to a 32.768-kHz crystal. The system frequency is controlled by the frequency control bits in the upper byte of the SYNCR as follows:

SYNCR — Clock Synthesizer Control Register $YFFA04

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

RESET:

0	0	1	1	1	1	1	1	0	0	0	U	U	0	0	0

U = Unaffected by reset

Figure 4-13. Clock Synthesizer Control Register

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 four, as specified in equation 4-9, which requires time for the VCO to relock.

X — Frequency Control Bit

This bit controls a divide by two prescaler that is not in the synthesizer feedback loop. Setting the bit doubles the system clock speed without changing the VCO speed, as specified in equation 4-9 for determining system frequency, and so no delay is incurred to relock the VCO.

Y5–Y0 — Frequency Control Bits

The Y bits, with a value from 0 to 63, control the modulus down counter in the synthesizer feedback loop, causing it to divide by the value of Y+1. (Refer to equation 4-9 for determining system frequency). Changing these bits requires a time delay for the VCO to relock.

EDIV — E-Clock Divide Rate
    1 = E-clock = system clock divided by 16
    0 = E-clock = system clock divided by 8

SLIMP — Limp Mode
    1 = A loss of crystal reference has been detected and the VCO is running at approximately half of maximum speed, determined from an internal voltage reference.
    0 = External crystal frequency is VCO reference.

SLOCK — Synthesizer Lock
    1 = VCO has locked on the desired frequency (or system clock is driven externally).
    0 = VCO is enabled, but has not yet locked.
The MCU maintains the reset state until the synthesizer locks, but the SLOCK bit does not indicate synthesizer lock status until after the user writes to SYNCR.

RSTEN — Reset Enable
    1 = Loss of crystal causes a system reset.
    0 = Loss of crystal causes the VCO to operate at a nominal speed without external reference (limp mode), and the MCU continues to operate at that speed.

STSIM — Stop System Integration Module Clock
    1 = When the LPSTOP instruction is executed, the SIM clock is driven from the VCO.
    0 = When the LPSTOP instruction is executed, the SIM clock is driven from the crystal oscillator and the VCO is turned off to conserve power.

STEXT — Stop External Clock

    1 = When the LPSTOP instruction is executed, the external clock pin (CLKOUT) is driven from the SIM clock, as determined by the STSIM bit.

    0 = When the LPSTOP instruction is executed, the external clock is held low to conserve power.

4.2.2 Phase Comparator and Filter

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

4.2.3 Frequency Divider

The frequency divider circuits divide the VCO frequency down to the reference frequency for the phase comparator. The frequency divider consists of the following elements:

- 3-bit prescaler controlled by the W bit in SYNCR
- 6-bit modulo down counter 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 MCU. The frequency of the system clock is given by the following equation:

$$F_{SYSTEM} = F_{CRYSTAL} [4(Y+1)2^{2W + X}] \qquad (4\text{-}9)$$

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

$$F_{VCO} = F_{SYSTEM}(2 - X) \qquad (4\text{-}10)$$

Since clearing 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 10 ELECTRICAL CHARACTERISTICS.**

Table 4-12 lists the frequencies available from various combinations of SYNCR bits with a reference frequency of 32.768 kHz.

Table 4-12. System Frequencies from 32.768-kHz Reference (Sheet 1 of 2)

Y	W = 0 X = 0	W = 0 X = 1	W = 1 X = 0	W = 1 X = 1
0 = 000000	131	262	524	1049
1 = 000001	262	524	1049	2097
2 = 000010	393	786	1573	3146
3 = 000011	524	1049	2097	4194
4 = 000100	655	1311	2621	5243
5 = 000101	786	1573	3146	6291
6 = 000110	918	1835	3670	7340
7 = 000111	1049	2097	4194	8389
8 = 001000	1180	2359	4719	9437
9 = 001001	1311	2621	5243	10486
10 = 001010	1442	2884	5767	11534
11 = 001011	1573	3146	6291	12583
12 = 001100	1704	3408	6816	13631
13 = 001101	1835	3670	7340	14680
14 = 001110	1966	3932	7864	15729
15 = 001111	2097	4194	8389	16777
16 = 010000	2228	4456	8913	
17 = 010001	2359	4719	9437	
18 = 010010	2490	4981	9961	
19 = 010011	2621	5243	10486	
20 = 010100	2753	5505	11010	
21 = 010101	2884	5767	11534	
22 = 010110	3015	6029	12059	
23 = 010111	3146	6291	12583	
24 = 011000	3277	6554	13107	
25 = 011001	3408	6816	13631	
26 = 011010	3539	7078	14156	
27 = 011011	3670	7340	14680	
28 = 011100	3801	7602	15204	
29 = 011101	3932	7864	15729	
30 = 011110	4063	8126	16253	
31 = 011111	4194	8389	16777	
32 = 100000	4325	8651		
33 = 100001	4456	8913		
34 = 100010	4588	9175		
35 = 100011	4719	9437		
36 = 100100	4850	9699		
37 = 100101	4981	9961		
38 = 100110	5112	10224		
39 = 100111	5243	10486		
40 = 101000	5374	10748		

Y	W = 0 X = 0	W = 0 X = 1	W = 1 X = 0	W = 1 X = 1
41 = 101001	5505	11010		
42 = 101010	5636	11272		
43 = 101011	5767	11534		
44 = 101100	5898	11796		
45 = 101101	6029	12059		
46 = 101110	6160	12321		
47 = 101111	6291	12583		
48 = 110000	6423	12845		
49 = 110001	6554	13107		
50 = 110010	6685	13369		
51 = 110011	6816	13631		
52 = 110100	6947	13894		
53 = 110101	7078	14156		
54 = 110110	7209	14418		
55 = 110111	7340	14680		
56 = 111000	7471	14942		
57 = 111001	7602	15204		
58 = 111010	7733	15466		
59 = 111011	7864	15729		
60 = 111100	7995	15991		
61 = 111101	8126	16253		
62 = 111110	8258	16515		
63 = 111111	8389	16777		

Note: All frequencies in kHZ

4.2.4 Clock Control

The clock control circuits determine the source used for both internal and external clocks during special circumstances, such as an LPSTOP execution.

Table 4-13 summarizes the clock activity during LPSTOP, with MODCK = 1 during reset. Any clock in the off state is held low.

Table 4-13. Clock Control Signals

Inputs		Clock Outputs	
STSIM	STEXT	SIMCLK	CLK
0	0	EXTAL	Off
0	1	EXTAL	EXTAL
1	0	VCO	Off
1	1	VCO	VCO

In LPSTOP mode, SIMCLK runs the periodic interrupt RESET and IRQ pin synchronizers.

4.3 Chip-Select Submodule

Typical microcomputer systems require external hardware to provide select signals to external peripherals. This MCU integrates these functions on-chip to provide the cost, speed, and reliability benefits at a higher level of integration. The chip-select signals can also be programmed as output enable, read or write strobe, or IACK signals.

Since initialization software would probably reside in a peripheral memory device controlled by the chip-select circuits, a CSBOOT register provides default reset values to support bootstrap operation.

The chip-select submodule supports the following programmable features:

- Twelve Programmable Chip-Select Circuits — Twelve chip-select signals are available (CSBOOT and CS10–CS0) that use the CSBOOT pin, bus arbitration pins BR, BG, and BGACK, function code pins FC2–FC0, and address pins A23–A19. The CSBOOT pin is dedicated to a single function because it must function after a reset with no initialization; the other chip-select circuits share functions on their output pins. All 12 chip-select circuits are independently programmable from the same list of selectable features. Each chip-select circuit has an individual base register and option register that contain the programmable characteristics of that chip select. Using these address lines as chip-select signals does not restrict the large linear address space of the MCU since the chip select logic always uses the internal address lines.

- Variable Block Sizes — The block size starting from the specified base address can be programmed as 2K, 8K, 16K, 64K, 128K, 256K, 512K, or 1M byte.

- Both 8-Bit and 16-Bit Ports Supported — Eight-bit ports are accessible on both odd and even addresses when connected to data bus bits 15–8. Sixteen-bit ports can be accessed as odd bytes, even bytes, or words.

- Read Only, Write-Only, or Read/Write Capability — Chip selects can be asserted synchronized with read, write, or both read and write.

- Address Strobe and Data Strobe Timing Option — Chip-select signals can be synchronized with either address strobe or data strobe, so that control signals such as output enable or write enable can be easily generated.

- Internal $\overline{\text{DSACK}}$ Generation with Wait States — The port size programmed in the pin assignment register can be referenced for generating $\overline{\text{DSACK}}$, and the proper number of wait states for a particular device programmed by the user.

- Address Space Checking — Supervisor, user, and CPU space accesses can be optionally checked.

- Interrupt Priority Level Checking (IPL) — In the IACK cycle, the acknowledged interrupt level can be compared with the user-specified level programmed in the option field. If autovector option is selected, $\overline{\text{AVEC}}$ is internally asserted.

- Discrete Output — Port C pins A22–A19 and FC2–FC0 can be programmed for discrete output, with data stored in the pin data register (CSPDR).

- M6800-Type Peripheral Support — M6800-type peripherals that require an E clock for synchronization can be supported. Chip-select is asserted, synchronized with the E clock on pin A23, providing correct data bus timing for the MCU.

The pin assignment registers (CSPAR0, CSPAR1) determine the function of the pins that can be assigned to chip selects. The initial pin assignment is determined by the state of the data bus pins when the MCU comes out of reset. Table 4-14 lists the allocation of chip-selects and discrete outputs on the pins of the MCU. The active states are defined as active low.

Table 4-14. Pin Allocation of Chip Selects

Pin	Chip Select	Discrete Outputs
CSBOOT	$\overline{\text{CSBOOT}}$	—
BR	$\overline{\text{CS0}}$	—
BG	$\overline{\text{CS1}}$	—
BGACK	$\overline{\text{CS2}}$	—
FC0	$\overline{\text{CS3}}$	DO0
FC1	$\overline{\text{CS4}}$	DO1
FC2	$\overline{\text{CS5}}$	DO2
A19	$\overline{\text{CS6}}$	DO3
A20	$\overline{\text{CS7}}$	DO4
A21	$\overline{\text{CS8}}$	DO5
A22	$\overline{\text{CS9}}$	DO6
A23	$\overline{\text{CS10}}$	E Clock

The CSPDR stores seven bits of data for the seven discrete outputs. A single $\overline{\text{DSACK}}$ generator is shared by all chip-select circuits. Figure 4-14 shows a block diagram of a single chip-select circuit.
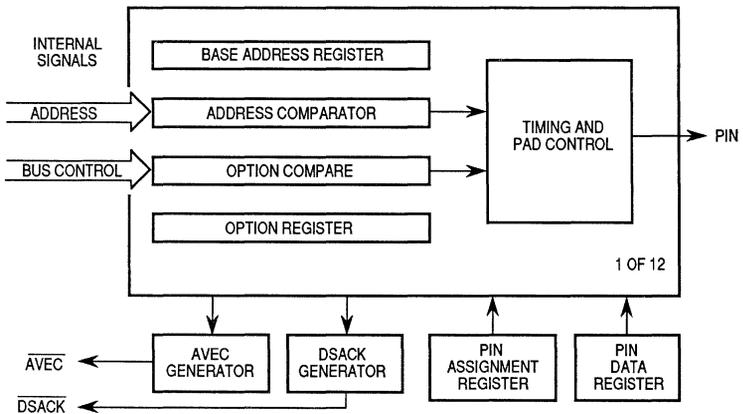
Figure 4-14. Chip-Select Circuit Block Diagram

4.3.1 Chip-Select Operation

Figure 4-15 is a flow diagram for the assertion of chip select. Since there are no differences in flow for chip selects between supervisor/user space and CPU space, the base and option registers must be properly programmed for each type of external bus cycle.

In CPU space, bits 15–3 of the base register must all be configured to match bits 23–11 of the address bus for a CPU space cycle, since the address is compared to a particular address generated by the CPU during a CPU space cycle.

To use the chip select as an IACK to an external device, the following conditions must be observed:

- The base address field must be programmed to all ones.
- The block size must be programmed to no more than 64K bytes to allow the address comparator to check more significant bits than A16 (since the CPU places the CPU space type on address lines A19–A16).
- The read/write field must be read only since an IACK cycle is performed as a read cycle.
- The upper/lower byte field must be lower byte if assigned to a 16-bit port, since an external vector is fetched from the lower byte for a 16-bit port.

Whenever the MCU makes an access, the chip-select circuits compare the following items:

- Function codes with the address space field
- Address lines with the base address and block size fields
- Read/write with the read/write field
- Enabled byte with the upper/lower field for 16-bit ports
- If the access is an IACK cycle, the level number of the interrupt being acknowledged (placed on A3–A1) is compared to the interrupt priority level field.

When a match occurs, the chip select is asserted synchronously with $\overline{AS}$ or $\overline{DS}$ in asynchronous mode or during the bus cycle of E clock in synchronous mode. In asynchronous mode, the $\overline{DSACK}$ field determines the need to generate $\overline{DSACK}$ internally.
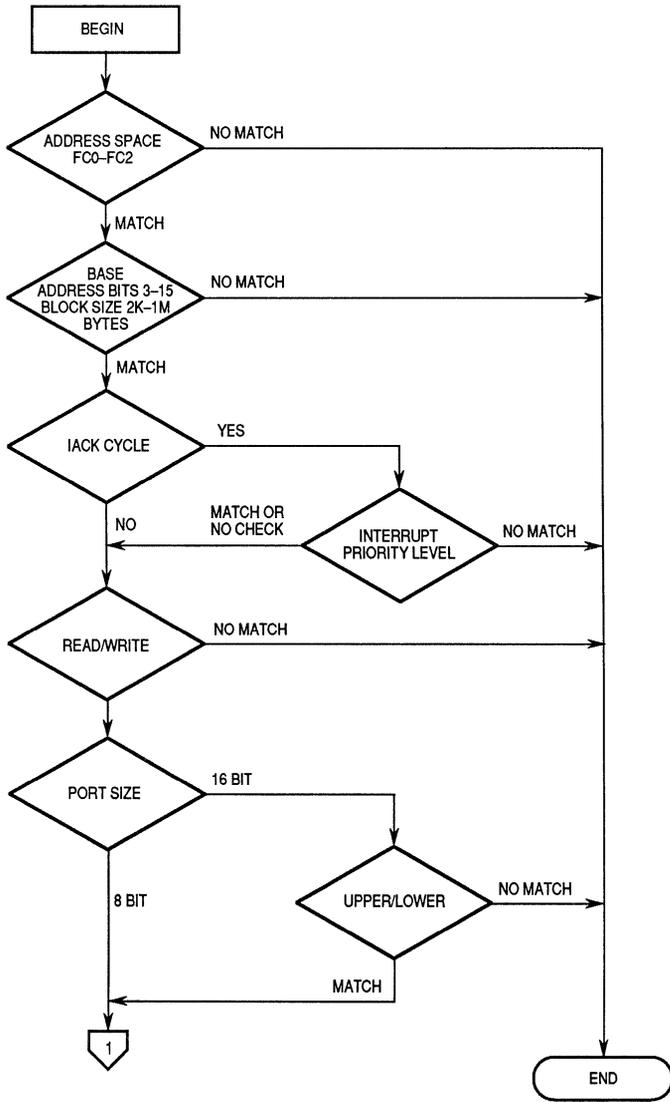
Figure 4-15. Flow Diagram for Chip Select (Sheet 1 of 3)
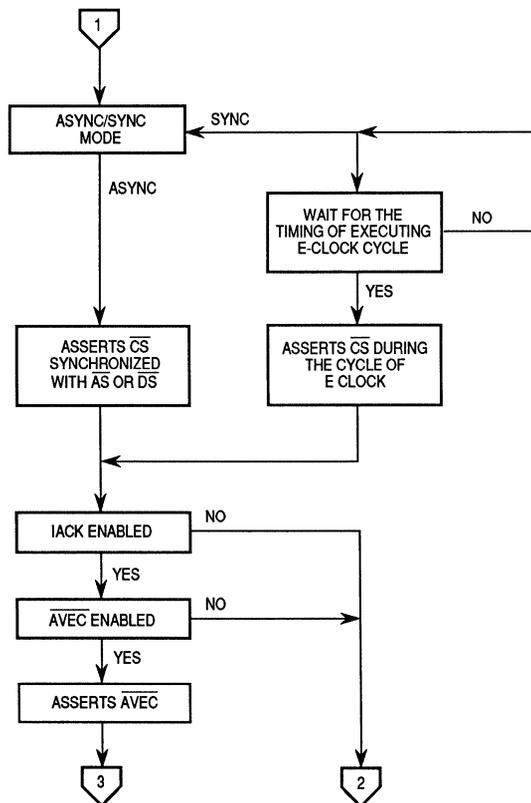
Figure 4-15. Flow Diagram for Chip Select (Sheet 2 of 3)
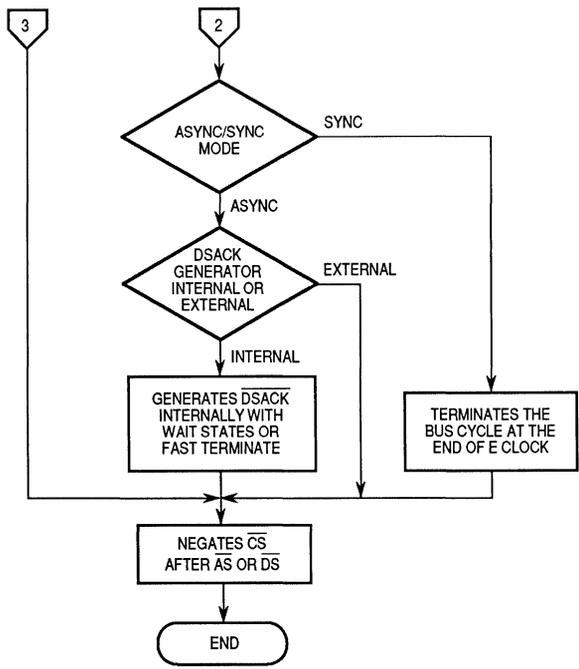
Figure 4-15. Flow Diagram for Chip Select (Sheet 3 of 3)

4.3.2 Pin Assignment Registers Description (CSPAR0, CSPAR1)

Chip-select pin assignment registers 0 and 1 (CSPAR0, CSPAR1) contain pairs of bits in two-bit binary format that determine the function of pins in the other chip-select registers. Parenthetic mnemonics in these registers are alternate functions for the associated pins.

The notation of DB2 in the reset value for bit 13 means that bit 13 of CSPAR0 assumes the value present at data bus pin 2 after reset.

CSPAR0 — Chip-Select Pin Assignment Register 0 $YFFA44

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	CS5 (FC2)		CS4 (FC1)		CS3 (FC0)		CS2 (BGACK)		CS1 (BG)		SWSR1		SWSR0	

RESET:

0	0	DB2	1	DB2	1	DB2	1	DB1	1	DB1	1	DB1	1	1	DB0

Bits 15–14 — Not Used

These bits always read zero; write has no effect.

CSPAR1 — Chip-Select Pin Assignment Register 1 $YFFA46

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	CS10 (A23)		CS9 (A22)		CS8 (A21)		CS7 (A20)		CS6 (A19)	

RESET:

0	0	0	0	0	0	DB7	1	DB6	1	DB5	1	DB4	1	DB3	1

Bits 15–10 — Not Used

These bits always read zero; write has no effect.

Figure 4-16. CSPAR0 and CSPAR1 Registers

Table 4-15 shows the hierarchical selection method of CSPAR1 for the bits that are controlled by the state of the data bus pins during reset. Table 4-16 lists the encoding of the bits in the pin assignment registers. Refer to **4.3.6.1 Pin Assignment Registers Operation** for additional information on RESET mode selection.

Table 4-15. Hierarchical Selection Structure of CSPAR1

Data Bus Pins At Reset					Default/Alternate Function				
DB7	DB6	DB5	DB4	DB3	CS10/A23	CS9/A22	CS8/A21	CS7/A20	CS6/A19
1	1	1	1	1	CS10	CS9	CS8	CS7	CS6
1	1	1	1	0	CS10	CS9	CS8	CS7	A19
1	1	1	0	X	CS10	CS9	CS8	A20	A19
1	1	0	X	X	CS10	CS9	A21	A20	A19
1	0	X	X	X	CS10	A22	A21	A20	A19
0	X	X	X	X	A23	A22	A21	A20	A19

Table 4-16. Pin Assignment Register Bit Encoding

Bits	Description
00	Discrete Output (E Clock on A23)*
01	Default Function
10	Chip Select (8-Bit Port)
11	Chip Select (16-Bit Port)

*Except for BR, BG, and BGACK

If a pin is programmed as a discrete output, the pin drives an external signal to the value specified in the pin data register, with the following exceptions:

- No discrete output function is available on pins $\overline{BR}$, $\overline{BG}$, or $\overline{BGACK}$. If these pins are programmed to discrete output, they still perform the function indicated by their pin names.
- Pin A23 provides E-clock on the output pin rather than a discrete output signal.

When a pin is programmed as a discrete output or default function, the internal chip-select logic still functions and can be used to generate $\overline{DSACK}$ or $\overline{AVEC}$ internally on an address match.

Port size must be determined when a pin is assigned as a chip select. If a pin is assigned as chip-select 8-bit port, the chip select is asserted at all addresses within the range of its block size. If a pin is assigned as chip-select 16-bit port, the upper/lower byte field of the option register determines with which byte the chip select is associated.

4.3.3 Base Address Registers (CSBARBT, CSBAR0–CSBAR10) Description

The base address is the starting address for the block enabled by a given chip select. The block size determines the extent of the block in the address space above the base address. Each chip select has its own associated base register, so that an efficient address map can be constructed for each application. Refer to Figure 4-17 for the chip-select base address register boot (CSBART) and chip-select base address registers 0–10 (CSBAR0–CSBAR10) registers.

CSBARBT — Chip-Select Base Address Register Boot$YFFA48

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
A23	A22	A21	A20	A19	A18	A17	A16	A15	A14	A13	A12	A11	BLKSZ		

RESET:

0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1

CSBAR0–CSBAR10 — Chip-Select Base Address Registers$YFFA4C–YFFA74

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
A23	A22	A21	A20	A19	A18	A17	A16	A15	A14	A13	A12	A11	BLKSZ		

RESET:

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 4-17. Base Address Registers

BLKSZ — Block Size Field
This field determines the size of the block above the base address that must be enabled by the chip select.

Table 4-17 lists the bit encoding for the base address registers block size field.

Table 4-17. Base Address Register Block Size Encoding

Block Size Field	Block Size	Address Lines Compared
000	2K	A23–A11
001	8K	A23–A13
010	16K	A23–A14
011	64K	A23–A16
100	128K	A23–A17
101	256K	A23–A18
110	512K	A23–A19
111	1M	A23–A20

NOTE

If a block size of 1M byte is selected and an external device requires an address A19–A0, pin A19 must be assigned as an address line and, therefore, cannot be used as a chip select.

Bits 15–3 — Base Address Field

In supervisor/user space, this field sets the starting address of a particular address space. Since the address compare logic uses only the most significant bits to cause an address match within its block size, the value of the base address must be a multiple of the block size. For example, with a block size of 64K bytes, the compare logic only uses bits 15–8 of the base register, which corresponds to address A23–A16. The register illustration for the base address registers shows the address lines compared by each bit in the registers.

4.3.4 Option Registers (CSORBT, CSOR0–CSOR10) Description

The option registers consist of eight fields that determine the timing and conditions for asserting the chip-select signals. Refer to Figures 4-18 and 4-19. These conditions make the chip selects useful for generating a variety of control signals for peripherals used with the MCU.

The option register for $\overline{\text{CSBOOT}}$, called CSORBT, contains reset values that differ from the other option registers to support bootstrap operations from peripheral memory devices.

CSORBT — Chip-Select Option Register Boot $YFFA4A

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
MODE	BYTE		R/$\overline{\text{W}}$		STRB		DSACK			SPACE		IPL			$\overline{\text{AVEC}}$

RESET:

0	1	1	1	1	0	1	1	0	1	1	1	0	0	0	0

Figure 4-18. Chip-Select Option Register Boot for $\overline{\text{CSBOOT}}$

CSOR0–CSOR10 — Chip-Select Option Registers $YFFA4E–YFFA76

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
MODE	BYTE		R/$\overline{\text{W}}$		STRB		DSACK			SPACE		IPL			$\overline{\text{AVEC}}$

RESET:

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 4-19. Chip-Select Option Registers 0–10

Table 4-18 provides a summary of the option register functions for quick reference.

Table 4-18. Option Register Functions Summary

Mode	Byte	R/W̄	STRB	DSACK̄	Space	IPL	AVEC̄
0 = ASYNC	00 = Off	00 = Rsvd	0 = AS̄	0000 = 0 WAIT	00 = CPU SP	000 = All	0 = Off
1 = SYNC	01 = Lower	01 = Read	1 = DS̄	0001 = 1 WAIT	01 = User SP	001 = Level 1	1 = On
	10 = Upper	10 = Write		0010 = 2 WAIT	10 = Supv SP	010 = Level 2	
	11 = Both	11 = Both		0011 = 3 WAIT	11 = S/U SP	011 = Level 3	
				0100 = 4 WAIT		100 = Level 4	
				0101 = 5 WAIT		101 = Level 5	
				0110 = 6 WAIT		110 = Level 6	
				0111 = 7 WAIT		111 = Level 7	
				1000 = 8 WAIT			
				1001 = 9 WAIT			
				1010 = 10 WAIT			
				1011 = 11 WAIT			
				1100 = 12 WAIT			
				1101 = 13 WAIT			
				1110 = F term			
				1111 = External			

The following bit descriptions apply to both the CSORBT and CSOR10–CSOR0 option registers.

MODE — Asynchronous/Synchronous Mode
  1 = Synchronous mode selected
  0 = Asynchronous mode selected
In asynchronous mode, the chip select is asserted synchronized with $\overline{AS}$ or $\overline{DS}$.

In synchronous mode, the $\overline{DSACK}$ field is not used, since a bus cycle is only performed as a synchronous operation. When a match condition occurs on a chip select programmed for synchronous operation, the chip select signals the EBI that an E-clock cycle is pending. The chip select is asserted with timing that meets the M6800 peripheral bus specification.

On a word or long-word transfer to an 8-bit port in synchronous mode, the MCU performs the consecutive cycles without inserting any E-clock cycles between the consecutive cycles. Refer to **SECTION 10 ELECTRICAL CHARACTERISTICS** for illustration of specific timing information.

In synchronous mode, the bus monitor timeout in the SYPCR must be programmed to a time longer than the number of clock cycles required for two E-clock cycles as programmed by the EDIV bit in the SYNCR.

BYTE — Upper/Lower Byte Option
This field is used only when the chip-select 16-bit port option is selected in the pin assignment register. These options are used to handle any bus transfer to a 16-bit port properly. Table 4-19 lists the upper/lower byte options.

Table 4-19. Byte Field

Bits	Description
00	Disable
01	Lower Byte
10	Upper Byte
11	Both Bytes

The disable option is used to disable chip-select logic. This option causes the associated pin to be driven high, and internally generated signals such as $\overline{DSACK}$ or $\overline{AVEC}$ are not asserted. This option can be used with both 8-bit and 16-bit port options.

For a single-byte transfer to an odd address, the chip-select logic can determine the byte accessed by the CPU by checking SIZ1–SIZ0 and A0 on the internal bus.

The both-bytes option is used to generate a control signal that enables both upper and lower bytes on an external device. In this case, the internal address and size lines are not checked.

If a MOVEP (move peripheral) instruction is used to access an 8-bit peripheral, the port size must be set to 16 bit, and either the upper or lower byte selected in the option fields to match the way the peripheral is connected to the data bus.

$R/\overline{W}$ — Read/Write
Table 4-20 shows the options for this field. This option causes the chip select to be asserted only for a read, only for a write, or for both read and write.

Table 4-20. R/$\overline{W}$ Field

Bits	Description
00	Reserved
01	Read Only
10	Write Only
11	Read/Write

STRB — Address Strobe/Data Strobe
    1 = Data strobe
    0 = Address strobe
This option controls the timing for assertion of a chip select in asynchronous mode. Selecting address strobe causes chip select to be asserted synchronized with address strobe. Selecting data strobe causes chip select to be asserted synchronized with data strobe.

This option is only used in asynchronous mode. In synchronous mode, this bit does not affect the timing of the chip select.

$\overline{DSACK}$ — Data Strobe Acknowledge
This option field specifies the source of the $\overline{DSACK}$ (externally or internally generated) in asynchronous mode. Table 4-21 shows the options for this field. It also allows the user to adjust the bus timing with internal $\overline{DSACK}$ generation by controlling the number of wait states that are inserted to optimize the bus speed in a particular application. The following table shows the $\overline{DSACK}$ field encoding.

A wait state has a duration of one clock cycle. The wait states are inserted beginning with S2 of the external bus cycle. For example, with the $\overline{DSACK}$ field set to two wait states, the internal $\overline{DSACK}$ is generated after two clock cycles, which is counted from S2. The total bus cycle time is, therefore, five clock cycles. The cycle is terminated by the first $\overline{DSACK}$ that occurs, and so if the external $\overline{DSACK}$ occurs earlier than the internal $\overline{DSACK}$, the external $\overline{DSACK}$ will terminate the bus cycle.

If an external device is fast enough, the bus cycle can be terminated at S3 by selecting the fast termination option (Refer to **3.2.6 Fast Termination Cycles**). For more specific timing information, refer to **SECTION 10 ELECTRICAL CHARACTERISTICS.**

Table 4-21. DSACK Field

Bits	Description
0000	No Wait States
0001	1 Wait State
0010	2 Wait States
0011	3 Wait States
0100	4 Wait States
0101	5 Wait States
0110	6 Wait States
0111	7 Wait States
1000	8 Wait States
1001	9 Wait States
1010	10 Wait States
1011	11 Wait States
1100	12 Wait States
1101	13 Wait States
1110	Fast Termination
1111	External $\overline{\text{DSACK}}$

SPACE — Address Space

This option field is used to check the address spaces indicated by the function codes generated by the CPU. Table 4-22 lists the address space option field encoding.

Table 4-22. Space Field

Bits	Description
00	CPU Space
01	User Space
10	Supervisor Space
11	Supervisor/User Space

IPL — Interrupt Priority Level

In an IACK cycle, the chip-select logic checks the acknowledged interrupt level on address lines A3–A1. If that level matches the level set in the IPL field, then the chip select can be asserted if the match conditions in the other fields are met. Table 4-23 lists the IPL field encoding.

Table 4-23. IPL Field

Bits	Description
000	Any Level
001	IPL1
010	IPL2
011	IPL3
100	IPL4
101	IPL5
110	IPL6
111	IPL7

Any level means that chip select is asserted regardless of the level of the IACK cycle.

This option field only affects the response of chip selects and does not affect interrupt recognition by the CPU.

$\overline{\text{AVEC}}$ — Autovector Enable
     1 = Autovector enabled
     0 = External interrupt vector enabled
This option field selects one of two methods of acquiring the interrupt vector during the IACK cycle.

If the chip select is configured to trigger on an IACK cycle and the $\overline{\text{AVEC}}$ field is set to one, the chip select automatically generates an $\overline{\text{AVEC}}$ in response to the IACK cycle. Otherwise, the vector must be supplied by the requesting device.

4.3.5 Chip-Select Pin Data Register (CSPDR) Description

The pin data register controls the state of pins programmed as discrete outputs. When a pin is assigned as a discrete output, the value in this register appears at the output. Data bits D06–D00 correspond to CS9–CS3 in order, as shown in Figure 4-20.

CSPDR — Chip-Select Pin Data Register   **$YFFA41**

7	6	5	4	3	2	1	0
0	DO6	DO5	DO4	DO3	DO2	DO1	DO0

RESET:

0	0	0	0	0	0	0	0
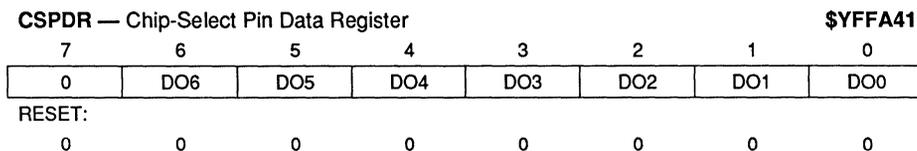
Figure 4-20. Chip-Select Pin Data Register

This is a read/write register. Bit 7 is not used. Writing to this bit has no effect, and it always reads zero when read.

4.3.6 Reset Mode Selection

When the external reset pin is asserted (whether internally or from external logic), except from the CPU executing the RESET instruction, the MCU reads data bus pin information. The data bus is pulled high internally to cause a specific default pin configuration, but the user can pull bits low to achieve a desired alternate configuration. Table 4-24 shows which pins determine a given configuration when the reset pin is released. For more information on reset operation, refer to **3.7 Reset Operation**.

Table 4-24. Mode Selection during Reset

Group	Mode Select Pin	Default Function (Pin Left High)	Alternate Function (Pin Pulled Low)
1	DB0	$\overline{\text{CSBOOT}}$ 16-Bit	$\overline{\text{CSBOOT}}$ 8-Bit
2	DB1	$\overline{\text{CS0}}$	$\overline{\text{BR}}$
		$\overline{\text{CS1}}$	$\overline{\text{BG}}$
		$\overline{\text{CS2}}$	$\overline{\text{BGACK}}$
3	DB2	$\overline{\text{CS3}}$	FC0
		$\overline{\text{CS4}}$	FC1
		$\overline{\text{CS5}}$	FC2
4	DB3	$\overline{\text{CS6}}$	A19
	DB4	$\overline{\text{CS7}}-\overline{\text{CS6}}$	A20–A19
	DB5	$\overline{\text{CS8}}-\overline{\text{CS6}}$	A21–A19
	DB6	$\overline{\text{CS9}}-\overline{\text{CS6}}$	A22–A19
	DB7	$\overline{\text{CS10}}-\overline{\text{CS6}}$	A23–A19
E	DB8	Bus Control $\overline{\text{DSACK0}}$, $\overline{\text{DSACK1}}$, $\overline{\text{AVEC}}$, $\overline{\text{DS}}$, $\overline{\text{AS}}$, SIZE, $\overline{\text{RMC}}$	PORTE
F	DB9	$\overline{\text{IRQ7}}-\overline{\text{IRQ1}}$ MODCK	PORTF
	DB11	Slave Mode Disabled	Slave Mode Enabled
	MODCK	VCO = System Clock	EXTAL = System Clock
	BKPT	Background Mode Disabled	Background Mode Enabled

4.3.6.1 Pin Assignment Registers Operation

The $\overline{\text{CSBOOT}}$ register selects a boot ROM containing a reset vector and initialization firmware. To do this, bit 1 in CSPAR0 has a reset value of one to assign $\overline{\text{CSBOOT}}$ as a chip select. Bit 0 is set to the value of DB0 when the MCU comes out of reset and the port size of $\overline{\text{CSBOOT}}$ is determined to be 8 or 16 bits.

All of the least significant bits of CS10–CS0 in CSPAR0 and CSPAR1 have a reset value of one. All of the most significant bits are set according to levels on DB7–DB1 at the end of reset. This determines whether the pins function as chip selects or their original functions.

Although $\overline{\text{CSBOOT}}$ is enabled at reset, CS10–CS0 are disabled at reset, since they should not be active until an initialization program sets up the base and option registers.

The chip-select pins are grouped to allow the pins to be conveniently connected to assign their function at reset. Table 4-24 shows these pin groupings.

In group one, DB0 determines the port size of $\overline{\text{CSBOOT}}$. Pulling DB0 low sets the port size of $\overline{\text{CSBOOT}}$ to 8 bit, or, if DB0 is left open, internal connections pull DB0 high and set the port size to 16 bit.

In groups two and three, one pin selects the entire group. If DB1 or DB2 is pulled low, the group is assigned to original functions. Otherwise, the group is assigned to chip selects.

In group four, an address line is associated with one data pin. However, any data pin pulled low assigns its associated pin and all less significant address line pins as address lines. For example, if DB5 is pulled low during reset, pins A19, A20, and A21 are assigned as address lines, and pins A22 and A23 are assigned as chip selects.

4.3.6.2 Base and Option Registers Operation

After reset, the MCU fetches the interrupt stack pointer and program counter from address $0000 0000. To support bootstrap operation from reset, the base address field in chip-select base address register boot (CSBARBT) has a reset value of all zeros. This allows a ROM device, containing reset vectors at the top of its address space, to be enabled by $\overline{\text{CSBOOT}}$ after a reset. The block size field in CSBARBT has a reset value of 1M byte.

The byte field in option register CSORBT has a reset value of both bytes, but CSOR10–CSOR0 have a reset value of disable, since they should not select external devices until an initial program sets up the base and option registers. Table 4-25 shows the reset values in the base and option registers for $\overline{\text{CSBOOT}}$.

Table 4-25. $\overline{\text{CSBOOT}}$ Base and Option Register Reset Values

Fields	Reset Values
Base Address	$0000 0000
Block Size	$1M Byte
Async/Sync Mode	Asynchronous Mode
Upper/Lower Byte	Both Bytes (CSORBT)
	Disable (CSOR10–CSOR0)
Read/Write	Read/Write
$\overline{\text{AS}}/\overline{\text{DS}}$	$\overline{\text{AS}}$
$\overline{\text{DSACK}}$	13 Wait States
Address Space	Supervisor/User Space
IPL	Any Level
Autovector	Interrupt Vector Externally

4.4 External Bus Interface Control

The following paragraphs describe the registers used to control the output pins that can be used with the external bus interface.

These registers can configure each Port E pin to be a bus control or an input/output. The state of DB8 during reset controls whether the port E pins are used as bus control signals or discrete I/O lines. If DB8 is low during reset, a value of $FF is set in the port E pin assignment register.

Refer to **SECTION 3 BUS OPERATION** for more information about the external bus interface. For a list of pin numbers used with port E and port F, see the MCU pinout diagram in **SECTION 12 MECHANICAL DATA.** See Figure 2-1 Function Signal Group for a block diagram of the port control circuits.

4.4.1 Port E Pin Assignment Register (PEPAR)

The bits in this register (Figure 4-21) control the function of each port E pin. Any bit set to one defines the corresponding pin to be a bus control signal, with the function defined in the register diagram. Any bit cleared to zero defines the corresponding pin to be an I/O pin, controlled by the port E data and data direction registers.

PEPAR — Port E Pin Assignment Register $YFFA17

7	6	5	4	3	2	1	0
PEPA7	PEPA6	PEPA5	PEPA4	PEPA3	PEPA2	PEPA1	PEPA0
(SIZ1)	(SIZ0)	($\overline{AS}$)	($\overline{DS}$)	($\overline{RMC}$)	($\overline{AVEC}$)	$\overline{DSACK1}$	$\overline{DSACK0}$

RESET:

DB8	DB8	DB8	DB8	DB8	DB8	DB8	DB8

Figure 4-21. Port E Pin Assignment Register

The state of data bus bit 8 (DB8) controls the state of this register following reset. If DB8 is high during reset, the register is set to $FF, which defines all port E pins to be bus control signals. If DB8 is low during reset, this register is set to $00, defining all port E pins to be I/O pins.

4.4.2 Port E Data Direction Register (DDRE)

The bits in this register (Figure 4-22) control the direction of the pin drivers when the pins are configured as I/O. Any bit in this register set to one configures the corresponding pin as an output. Any bit in this register cleared to zero configures the corresponding pin as an input.

This register can be read or written at any time.

DDRE — Port E Data Direction Register $YFFA15

7	6	5	4	3	2	1	0
DDE7	DDE6	DDE5	DDE4	DDE3	DDE2	DDE1	DDE0

RESET:

0	0	0	0	0	0	0	0

Figure 4-22. Port E Data Direction Register

4.4.3 Port E Data Register (Port E)

A write to the port E data register (Figure 4-23) is stored in the internal data latch, and if any port E pin is configured as an output, the value stored for that bit is driven on the pin. A read of PORTE returns the value at the pin only if the pin is configured as a discrete input. Otherwise, the value read is the value stored in the register.

PORTE — Port E Data Register $YFFA11, YFFA13

7	6	5	4	3	2	1	0
PE7	PE6	PE5	PE4	PE3	PE2	PE1	PE0

RESET:
U   U   U   U   U   U   U   U

Figure 4-23. Port E Data Register

Port E is a single register that can be accessed in two locations. These registers can be read or written at any time.

4.4.4 Port F Pin Assignment Register (PFPAR)

The bits in this register (Figure 4-24) control the function of each port F pin. Any bit set to one defines the corresponding pin to be an interrupt request input as defined in the register diagram. Any bit cleared to zero defines the corresponding pin to be an I/O pin, controlled by the port F data and data direction registers. The MODCK signal has no function after reset.

PFPAR — Port F Pin Assignment Register $YFFA1F

7	6	5	4	3	2	1	0
PFPA7	PFPA6	PFPA5	PFPA4	PFPA3	PFPA2	PFPA1	PFPA0
($\overline{\text{IRQ7}}$)	($\overline{\text{IRQ6}}$)	($\overline{\text{IRQ5}}$)	($\overline{\text{IRQ4}}$)	($\overline{\text{IRQ3}}$)	($\overline{\text{IRQ2}}$)	($\overline{\text{IRQ1}}$)	(MODCK)

RESET:
DB9   DB9   DB9   DB9   DB9   DB9   DB9   DB9

Figure 4-24. Port F Pin Assignment Register

The state of data bus bit 9 controls the state of this register following reset. If DB9 is high during reset, the register is set to $FF, which defines all port F pins to be interrupt request inputs. If DB9 is low during reset, this register is set to $00, defining all port F pins to be I/O pins.

4.4.5 Port F Data Direction Register (DDRF)

The bits in this register (Figure 4-25) control the direction of the pin drivers when the pins are configured as I/O. Any bit in this register set to one configures the corresponding pin as an output. Any bit in this register cleared to zero configures the corresponding pin as an input.

DDRF — Port F Data Direction Register $YFFA1D

7	6	5	4	3	2	1	0
DDF7	DDF6	DDF5	DDF4	DDF3	DDF2	DDF1	DDF0

RESET:

| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

Figure 4-25. Port F Data Direction Register

4.4.6 Port F Data Register (PORTF)

The write to the port F data register (Figure 4-26) is stored in the internal data latch, and if any port F pin is configured as an output, the value stored for that bit is driven on the pin. A read of PORTF returns the value at the pin only if the pin is configured as a discrete input. Otherwise, the value read is the value stored in the register.

PORTF — Port F Data Register $YFFA19,YFFA1B

7	6	5	4	3	2	1	0
PF7	PF6	PF5	PF4	PF3	PF2	PF1	PF0

RESET:

| U | U | U | U | U | U | U | U |

Figure 4-26. Port F Data Register

Port F is a single register that can be accessed in two locations. These registers can be read or written at any time.

4.5 Test Submodule

The test submodule is only used for factory production testing of the MCU. It is not intended for general use and is not supported for normal applications. The description of the test submodule contained in this manual is for informational purposes only and is not intended to provide operational information. Applications using this MCU should avoid invoking the test features of this submodule.

4.5.1 Entering Test Mode

Test mode is entered by a combination hardware and software method: a register bit must be set while an external pin is in the correct state. To enter test mode, the following conditions must be met:

- The TSTME pin (test mode enabled — active low) must be pulled low.
- The enter test mode (ETM) bit in the test submodule control register (CREG) must be set. This is a write-once bit, and so it must not have been written since reset, and it must be set to one on the first attempt to write it. Writing the bit to zero in initialization software prevents any accidental entry to test mode in normal operation.

The TSTME pin must remain low to stay in test mode. If TSTME goes high while the MCU is in test mode, the MCU is reset, exiting test mode.

The TSTME pin has a second function: when driven to 1.6 times V_{DD}, the MCU places all output driver circuits in a high-impedance state, isolating the MCU from the remainder of the system.

4.5.2 Test Submodule Control Register (CREG)

The test submodule control register (Figure 4-27) configures the test submodule for the test operation desired, provides the bus master with the means for controlling the test sequence, and retains status information on the test submodule.

CREG — Test Submodule Control Register $YFFA38

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
BUSY	TMARM	COMP	IMBTST	CPUTR	QBIT	MUXEL	—	—	—	—	ACUT	SCONT	SSHOP	SATO	ETM

RESET:

| 1 | TSTME | U | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

Figure 4-27. Test Submodule Control Register

ETM — Enter Test Mode
   1 = Enter test mode
   0 = Stay in normal mode
This bit can be written only once after reset. It can be set only if the TSTME pin is asserted. This bit can be read anytime.

SATO — Start Automatic Test Operation
   1 = Start an automatic test operation
   0 = Stay in normal mode
This bit can be read and written at any time.

SSHOP — Start Shifting Operation
    1 = Start a shifting operation
    0 = Stay in normal mode
    This bit can be read and written at any time.

SCONT — Start Continuous Operation
    1 = Start continuous operation
    0 = Stop continuous operation
    This bit can be read and written at any time.

ACUT — Activate Circuit Under Test
    1 = Assert the ACUTL line
    0 = Stay in normal mode
    This bit can be written at any time, but always reads zero because it clears in less than a bus cycle.

MUXSEL — Multiplexer Select Bit
    1 = Shift in source for master shift register B (MSRB) is the external interrupt pin.
    0 = Shift in source for MSRB is the internal test line.
    This bit can be written only in test mode.

QBIT — Quotient Bit
    1 = The least significant bit of master shift register B is available at the quotient/freeze (FREEZE/QUOT) pin.
    0 = The internal freeze status is available at the FREEZE/QUOT pin.
    This bit can be read and written only in test mode.

CPUTR — CPU Test Register
    1 = Scan lines connected to the CPU test register
    0 = Scan lines disconnected from the CPU test register
    This bit can be written only in test mode, but read at any time.

IMBTST — Intermodule Bus Test
    1 = Internal interconnect lines are configured as test lines.
    0 = Internal interconnect lines have normal function.
    This bit can be read and written only in test mode.

COMP — Compare Status Bit
    1 = Master shift register B contains the correct answer for the user self-test basic test.
    0 = Master shift register B does not contain the correct answer for the user self-test basic test.
    This status bit can be read at any time, but cannot be written.

TMARM — Test Mode Armed Status Bit
  1 = TSTME pin is asserted; test mode can be entered by setting the ETM control
    bit.
  0 = TSTME pin is negated; test mode cannot be entered.
This status bit can be read at any time, but cannot be written.

BUSY — Test Submodule Busy Status Bit
  1 = Test submodule is busy.
  0 = Test submodule is not busy.
Busy indicates a test function is in progress. This status bit can be read at any time,
but cannot be written.

4.5.3 Distributed Register (DREG)

The distributed register configures the master shift registers during factory
production tests.

DREG — Distributed Register $YFFA3A

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
—	—	—	—	—	WAIT3	WAIT2	WAIT1	MSRA18	MSRA17	MSRA16	MSRAC	MSRB18	MSRB17	MSRB16	MSRBC

RESET:

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 4-28. Distributed Register

WAIT3–WAIT1 — Wait Counter Preset 3–1
  These bits program the delay time between automatic test sequences. These bits
  can be read or written at any time.

MSRA18–MSRA16 — Master Shift Register A Bits 18–16
  These bits are the three most significant bits of master shift register A. These bits
  can be read or written at any time.

MSRAC — Master Shift Register A Configuration
  This bit can be read or written at any time.

MSRB18–MSRB16 — Master Shift Register B Bits 18–16
  These bits are the upper three bits of master shift register B. These bits can be read
  or written at any time.

MSRBC — Master Shift Register B Configuration
This bit can be read or written at any time.

4.5.4 Master Shift Register A (MSRA)

Master shift register A can be read and written by the bus master. Reset does not affect contents of master shift register A.

4.5.5 Shift Count Register A

Shift count register A is an 8-bit shift register that can be read and written by the bus master. After reset, both shift count register A is initialized to zero.

4.5.6 Master Shift Register B (MSRB)

Master shift register B can be read and written by the bus master. Reset does not affect the contents of master shift register B.

As a simple shift register, test responses are manually shifted to master shift register B from the module under test. This uncompressed data can then be read 16 bits at a time by the bus master.

4.5.7 Shift Count Register B

Shift count register B is an 8-bit shift register that can be read and written by the bus master. After reset, shift count register B is initialized to zero.

4.5.8 Reps Counter

After reset, the reps counter is set to zero.

4

SECTION 5
QUEUED SERIAL MODULE

The queued serial module (QSM) provides the microcontroller unit (MCU) with two serial communication interfaces divided into two submodules: the queued serial peripheral interface (QSPI) and the serial communications interface (SCI). The QSPI is a full-duplex, synchronous serial interface for communicating with peripherals and other MCUs. It is enhanced by the addition of a queue for receive and transmit data. The SCI is a full-duplex universal asynchronous receiver transmitter (UART) serial interface. These submodules operate independently. This section provides a block diagram, memory map, pin description, and register descriptions of the QSM, with a breakdown of both the QSPI and SCI submodules. Operation of the QSPI submodule includes master mode and slave mode. For a detailed description refer to **5.5.5.1 Master Mode** and **5.5.5.2 Slave Mode.** In addition, operation of the SCI submodule is divided into transmit and receive. A description of these operations is given in **5.6.4 Transmitter Operation** and **5.6.5 Receiver Operation.** To aid in grasping an understanding of the numerous bits and fields of the registers that appear throughout the text, a quick reference guide identifies all bit/field acronyms. (Refer to Table 5-3.)

5.1 Block Diagram

Figure 5-1 depicts the major components of the QSM, which consist of the global registers, logic control, and the QSPI and SCI submodules. Refer to **5.5 QSPI Submodule** and **5.6 SCI Submodule** for further definition of these components.
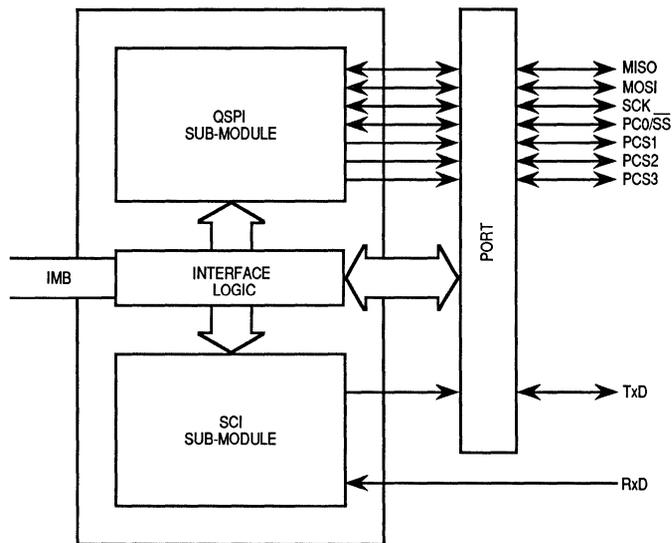
Figure 5-1. QSM Block Diagram

5.2 Memory Map

The QSM memory map is comprised of the global registers, the QSPI and SCI control and status registers, and the QSPI RAM as shown in Figure 5-2. For an accurate location of the QSM memory in the MCU memory map, refer to **1.4 Module Memory Map.** The QSM memory map may be divided into two segments: supervisor-only data space and assignable data space.

	15	8 7	0	
$YFFC00	QMCR			SUPERVISOR-ONLY DATA SPACE
$YFFC02	QTEST			
$YFFC04	QILR		QIVR	
$YFFC06	RESERVED			
$YFFC08	SCCR0			
$YFFC0A	SCCR1			
$YFFC0C	SCSR			
$YFFC0E	SCDR			
$YFFC10	RESERVED			
$YFFC12	RESERVED			
$YFFC14	RESERVED		QPDR	
$YFFC16	QPAR		QDDR	ASSIGNABLE DATA SPACE
$YFFC18	SPCR0			(SUPERVISOR-ONLY OR UNRESTRICTED)
$YFFC1A	SPCR1			
$YFFC1C	SPCR2			
$YFFC1E	SPCR3		SPSR	
$YFFC20-FF	RESERVED			
$YFFD00-1F	REC.RAM			
$YFFD20-3F	TRAN.RAM			QUEUE RAM
$YFFD40-4F	COMD RAM			

Y = m111 where m is the modmap bit in the SIM MCR (Y = $7 or $F).

Figure 5-2. QSM Memory Map

The supervisor-only data space segment contains the QSM global registers. These registers define parameters needed by the QSM to integrate with the MCU. Access to these registers is permitted only when the CPU is operating in supervisor mode (CPU status register, S bit = 1).

Assignable data space can be either restricted to supervisor-only access or unrestricted to both supervisor and user accesses. The supervisor (SUPV) bit in the QSM module configuration register (QMCR) designates the assignable data space as either supervisor or unrestricted. If SUPV is set, then the space is designated as supervisor-only space. Access is then permitted only when the CPU is operating in supervisor mode. All attempts to read supervisor data spaces when not in supervisor mode (CPU status register, S bit = 0) return a value of zero, and all attempts to write have no effect. If SUPV is clear, both user and supervisor accesses are permitted. To clear SUPV in the QMCR, the CPU must be in supervisor mode (CPU status register, S-bit = 1). Refer to Processing States in the CPU32 manual for more information on supervisor mode.

The QSM assignable data space segment contains the submodules, QSPI and SCI, control/status registers, and the QSPI RAM. All registers and RAM may be accessed on byte, word, and long-word boundaries. The 80 bytes of static RAM are distinct from the QSM register set. All bytes not used by the QSPI may be used as general-purpose RAM. When operating, the QSPI submodule uses three noncontiguous blocks of the 80-byte RAM for receive, transmit, and control data. More information on the QSPI RAM can be found in **5.5.4.6 QSPI Ram.**

The contents of most locations in the memory map may be rewritten with the identical value to that location, with one exception. (Refer to **5.5.4.3 QSPI Control Register 2 (SPCR2).**) Writing a different value to certain control registers when a submodule using that register is enabled can cause unpredictable results. For predictable operation, if register bits are to be changed, the CPU should disable the submodule in an orderly fashion before altering the registers.

5.3 QSM Pins

The QSM has nine external pins as shown in Figure 5-1. Eight of the pins, if not in use for their submodule function, can be used as general-purpose I/O port pins, The ninth pin, RXD, is an input-only pin used exclusively by the SCI submodule.

Table 5-1 summarizes the QSM pin functions, which are determined by the QSPI mode of operation (master or slave), SCI operation, the pin assignment register (QPAR), and by the appropriate QSM data direction register (QDDR) bit.

Table 5-1. QSM Pin Summary

	Pin	Mode	QDDR Bit	Pin Function
QSPI Pins	MISO	Master	0	Serial Data Input to QSP1
			1	Output Value from QPDR
		Slave	0	Input Value to QPDR
			1	Serial Data Output from QSP1
	MOSI	Master	0	Input Value to QPDR
			1	Serial Data Output from QSPI
		Slave	0	Serial Data Input to QSPI
			1	Output Value from QPDR
	SCK	Master	0	Input Value to QPDR
			1	Clock Output from QSPI
		Slave	0	Clock Input to QSPI
			1	Output Value from QPDR
	PCS0/$\overline{SS}$	Master	0	Input (May Cause Mode Fault)
			1	Output Selects Peripheral(s)
		Slave	0	Input Selects the QSPI
			1	Output Value from QPDR
	PCS3-PCS1	Master	0	Input Value to QPDR
			1	Output Selects Peripherals
		Slave	0	Input Value to QPDR
			1	Output Value from QPDR
SCI Pins	TXD	Transmit	X	Serial Data Output from SCI (TE = 1)
	RXD	Receive	NA	Serial Data Input to SCI

X = QDDR bit ignored

The QSM pin control registers — QDDR, QSM pin assignment register (QPAR), and QSM port data register (QPDR) — affect pins being used as general-purpose I/O pins. The QSPI control register 0 (SPCR0) has one bit that affects seven pins employed as general-purpose output pins. Within this register the wired-OR mode (WOMQ) control bit determines whether MISO, MOSI, SCK, and PCS3–PCS0 function as open-drain output pins or as normal output pins, regardless of their use as general-purpose I/O pins or as QSPI output pins. Likewise, the SCI control register 1 (SCCR1) has one bit that affects the TXD pin when it is employed as a general-purpose output. In this register the wired-

OR mode (WOMS) control bit determines whether TXD functions as an open-drain output pin or a normal output pin, regardless of this pin's use as a general-purpose output pin or as an SCI output pin.

5.4 Registers

Registers of the QSM are divided into four categories: QSM global registers, QSM pin control registers, QSPI submodule registers, and SCI submodule registers. The QSPI and SCI registers are defined in **5.5 QSPI Submodule** and **5.6 SCI Submodule**, respectively. Writes to unimplemented bits have no meaning or effect, and reads from unimplemented bits always return a logic zero value.

The modmap bit of the system integration module (SIM) module configuration register (MCR) defines the most significant bit (A23) of the address, shown in each register figure as Y (Y = $7 or $F). This bit, concatenated with the rest of the address given, forms the absolute address of each register.

Table 5-2 is a summary of the registers, bits, and reset states for the full QSM module.

As previously mentioned, Table 5-3 is a quick reference guide to all the bits/fields of the QSM module. Along with the function, the register and register location of each bit/field are identified.

Table 5-2. QSM Register Summary

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
QMCR $YFFC00	STOP	FRZ1	FRZ0	0	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
QTEST $YFFC02	0	0	0	0	0	0	0	0	0	0	0	0	TSBD	SYNC	TQSM	TMM
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
QILR-QIVR $YFFC04	0	0	ILQSPI		ILSCI			INTV								
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1
$YFFC06	Reserved															
SCCR0 $YFFC08	0	0	0	SCBR												
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0
SCCR1 $YFFC0A	0	LOOPS	WOMS	ILT	PT	PE	M	WAKE	TIE	TCIE	RIE	ILIE	TE	RE	RWU	SBK
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
SCSR $YFFC0C	0	0	0	0	0	0	0	TDRE	TC	RDRF	RAF	IDLE	OR	NF	FE	PF
RESET:	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0
SCDR $YFFC0E	0	0	0	0	0	0	0	R8/T8	R7/T7	R6/T6	R5/T5	R4/T4	R3/T3	R2/T2	R1/T1	R0/T0
RESET:	0	0	0	0	0	0	0	U	U	U	U	U	U	U	U	U
$YFFC10	Reserved															
$YFFC12	Reserved															
QPDR $YFFC14	0	0	0	0	0	0	0	0	D7 (TXD)	D6 (PCS3)	D5 (PCS2)	D4 (PCS1)	D3 (PCS0*)	D2 (SCK)	D1 (MOSI)	D0 (MISO)
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
QPAR-QDDR $YFFC16	0	PCS3	PCS2	PCS1	PCS0*	0	MOSI	MISO	TXD	PCS3	PCS2	PCS1	PCS0*	SCK	MOSI	MISO
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
SPCR0 $YFFC18	MSTR	WOMQ	BITS				CPOL	CPHA	SPBR							
RESET:	0	0	0	0	0	0	0	1	0	0	0	0	0	1	0	0
SPCR1 $YFFC1A	SPE	DSCKL							DTL							
RESET:	0	0	0	0	0	1	0	0	0	0	0	0	0	1	0	0
SPCR2 $YFFC1C	SPIFIE	WREN	WRTO	0	ENDQP				0	0	0	0	NEWQP			
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
SPCR3-SPSR $YFFC1E	0	0	0	0	0	LOOPQ	HMIE	HALT	SPIF	MODF	HALTA	0	CPTQP			
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
$YFFC20- $YFFCFF	Reserved															
REC.RAM $YFFD00- $YFFD1F	QSPI Receive Data (16 Words)															
TRAN.RAM $YFFD20- $YFFD3F	QSPI Transmit Data (16 Words)															
COMD RAM $YFFD40- $YFFD4F	CONT	BITSE	DT	DSCK	PCS3	PCS2	PCS1	PCS0*	CONT	BITSE	DT	DSCK	PCS3	PCS2	PCS1	PCS0*

Y = m111, where m is the modmap bit in the module configuration register fo the SIM (Y = $7 or $F).

* The PCS0 bit listed above represents the dual-function PCS0/$\overline{SS}$.

Table 5-3. Bit/Field Quick Reference Guide (Sheet 1 of 2)

Bit/Field Mnemonic	Function	Register	Register Location
SPBR	Serial Clock Baud Rate	SPCR0	QSPI
BITS	Bits Per Transfer	SPCR0	QSPI
BITSE	Bits Per Transfer Enable	QSPI RAM	QSPI
SCBR	Baud Rate	SCCR0	SCI
CONT	Continue	QSPI RAM	QSPI
CPHA	Clock Phase	SPCR0	QSPI
CPOL	Clock Polarity	SPCR0	QSPI
CPTQP	Completed Queue Pointer	SPSR	QSPI
DSCK	Peripheral Select Chip (PSC) to Serial Clock (SCK) Delay	QSPI RAM	QSPI
DSCKL	Delay before Serial Clock (SCK)	SPCR1	QSPI
DT	Delay after Transfer	QSPI RAM	QSPI
DTL	Length of Delay after Transfer	SPCR1	QSPI
ENDQP	Ending Queue Pointer	SPCR2	QSPI
FE	Framing Error Flag	SCSR	SCI
FRZ1-0	Freeze1-0	QMCR	QSM
HALT	Halt	SPCR3	QSPI
HALTA	Halt Acknowledge Flag	SPSR	QSPI
HMIE	Halt Acknowledge Flag (HALTA) and Mode Fault Flag (MODF) Interrupt Enable	SPCR3	QSPI
IARB	Interrupt Arbitration Identification Number	QMCR	QSM
IDLE	Idle Line Detected Flag	SCSR	SCI
ILIE	Idle Line Interrupt Enable	SCCR1	SCI
ILQSPI	Interrupt Level for QSPI	QILR	QSM
ILSCI	Interrupt Level of SCI	QILR	QSM
ILT	Idle Line Detect Type	SCCR1	SCI
INTV	Interrupt Vector	QIVR	QSM
LOOPS	SCI Loop Mode	SCCR1	SCI
LOOPQ	QSPI Loop Mode	SPCR3	QSPI
M	Mode Select (8/9 Bit)	SCCR1	SCI
MISO	Master in Slave Out	QPAR/QDDR/QPDR	QSM
MODF	Mode Fault Flag	SPSR	QSPI
MOSI	Master Out Slave In	QPAR/QDDR/QPDR	QSM
MSTR	Master/Slave Mode Select	SPCR0	QSPI
NEWQP	New Queue Pointer Value	SPCR2	QSPI

Table 5-3. Bit/Field Quick Reference Guide (Sheet 2 of 2)

Bit/Field Mnemonic	Function	Register	Register Location
NF	Noise Error Flag	SCSR	SCI
OR	Overrun Error Flag	SCSR	SCI
PCS0/SS	Peripheral Chip Select/Slave Select	QPAR/QDDR/QPDR	QSM
PCS3–PCS1	Peripheral Chip Selects	QPAR/QDDR/QPDR	QSM
PE	Parity Enable	SCCR1	SCI
PF	Parity Error Flag	SCSR	SCI
PT	Parity Type	SCCR1	SCI
R8-R0	Receive 8–0	SCDR	SCI
RAF	Receiver Active Flag	SCSR	SCI
RDRF	Receive Data Register Full Flag	SCSR	SCI
RE	Receiver Enable	SCCR1	SCI
RIE	Receiver Interrupt Enable	SCCR1	SCI
RWU	Receiver Wakeup	SCCR1	SCI
SBK	Send Break	SCCR1	SCI
SCK	Serial Clock	QDDR/QPDR	QSM
SPE	QSPI Enable	SPCR1	QSPI
SPIF	QSPI Finished Flag	SPSR	QSPI
SPIFIE	SPI Finished Interrupt Enable	SPCR2	QSPI
STOP	Stop	QMCR	QSM
SUPV	Supervisor/Unrestricted	QMCR	QSM
SYNC	SCI Baud Clock Sync Signal	QTEST	QSM
T8-T0	Transmit 8–0	SCDR	SCI
TC	Transmit Complete Flag	SCSR	SCI
TCIE	Transmit Complete Interrupt Enable	SCCR1	SCI
TDRE	Transmit Data Register Empty Flag	SCSR	SCI
TE	Transmit Enable	SCCR1	SCI
TIE	Transmit Interrupt Enable	SCCR1	SCI
TMM	Test Memory Map	QTEST	QSM
TQSM	Test QSM Enable	QTEST	QSM
TSBD	SPI Test Scan Path Select	QTEST	QSM
TXD	Transmit Data	QDDR/QPDR	QSM
WAKE	Wakeup Type	SCCR1	SCI
WOMQ	Wired-OR Mode for QSPI Pins	SPCR0	QSPI
WOMS	Wired-OR Mode for SCI Pins	SCCR1	SCI
WREN	Wrap Enable	SPCR2	QSPI
WRTO	Wrap To Select	SPCR2	QSPI

5

5.4.1 Overall QSM Configuration Summary

After reset, the QSM remains in an idle state, requiring initialization of several registers before any serial operations may begin execution. The following registers, fields, and bits are fully described later in this section. A general sequence guide for initialization follows:

- QMCR (refer to **5.4.2.1 QSM Configuration Register (QMCR)**)
  This register must be initialized to properly configure:
  — Interrupt arbitration identification number used by the entire QSM module
  — Supervisor/unrestricted bit (SUPV)
  — FREEZE and/or STOP configuration should remain cleared to zero for normal operation.

- QIVR and QILR (refer to **5.4.2.3 QSM Interrupt Level Register (QILR)** and **5.4.2.4 QSM Interrupt Vector Register (QIVR)**)
  These registers are written to choose the base vector number for the entire QSM module and individual interrupt levels for the QSPI and SCI submodules.

- QPDR and QDDR (refer **5.4.3.1 QSM Port Data Register (QPDR)** and **5.4.3.3 QSM Data Direction Register (QDDR)**)
  The pin control registers should be initialized in the order QPDR and then QDDR, thus establishing the default state and direction of the QSM pins.

For configuration of the QSPI submodule, initialize as follows:

- RAM (refer to **5.5.4.6 QSPI Ram**)
- QPAR (refer **5.4.3.2 QSM Pin Assignment Register (QPAR)**)

Assignment of appropriate pins to the QSPI must be made with this register.

- SPCR0 (refer **5.5.4.1 QSPI Control Register 0 (SPCR0)**)
  The system designer must choose a transfer rate (baud) for operation in master mode, an appropriate clock phase, clock polarity, and the number of bits to be transferred in a serial operation. Master/slave mode select (MSTR) must be set to configure the QSPI for master mode or cleared to configure operation in slave mode. WOMQ should be set to enable or cleared to disable wired-OR mode operation.

- SPCR1 (refer to **5.5.4.2 QSPI Control Register (SPCR1)**)
    - SPE must be set to enable the QSPI; this register should be written last.
    - DTL allows the user to program a delay after any serial transfer, which is invoked by the DT bit for any serial transfer.
    - DSCKL allows the user to set a delay before SCK (after PCS valid), which is invoked by the DSCK bit for any transfer.

- SPCR2 (refer to **5.5.4.3 QSPI Control Register 2 (SPCR2)**)
    - NEWQP and ENDQP, respectively, determine the beginning of a queue and the number of serial transfers (up to 16) to be considered a complete queue.
    - WREN is set to enable queue wraparound, and WRTO helps determine the address used in wraparound mode.
    - SPIFIE is set to enable interrupts when SPIF is asserted.

- SPCR3 (refer to **5.5.4.4 QSPI Control Register 3 (SPCR3)**)
    HALT may be used for program debug, and HMIE is set to enable CPU interrupts when HALTA or MODF is asserted; LOOPQ is set only to enable a feedback loop that can be used for self-test mode.

For configuration of the SCI submodule, initialize as follows:

- SCCR0 (refer to **5.6.3.1 SCI Control Register 0 (SCCR0)**)
    The system designer must choose a transfer rate (baud) for serial transfer operation.

- SCCR1 (refer to **5.6.3.2 SCI Control Register 1 (SCCR1)**)
    - The type of serial frame (8- or 9-bit) and the use of parity must be determined by M, PE, and PT.
    - For receive operation, the system designer must consider use and type of wakeup (WAKE, RWU, ILT, ILIE). The receiver must be enabled (RE) and, usually, RIE should be set.
    - For transmit operation, the transmitter must be enabled (TE) and, usually, TIE should be set. The use of wired-OR mode (WOMS) must also be decided. Once the transmitter is configured, data is not sent until TDRE and TC are cleared. To clear TDRE and TC, the SCSR read must be followed by a write to SCDR (either the lower byte or the entire word).

5.4.2 QSM Global Registers

The QSM global registers contain system parameters used by both the QSPI and the SCI submodules. These registers define parameters used by the QSM to interface with the CPU and other system modules. The four global registers are listed in Table 5-4.

Table 5-4. QSM Global Registers

Address	Name	Usage
$YFFC00	QMCR	QSM Configuration Register
$YFFC02	QTEST	QSM Test Register
$YFFC04	QILR	QSM Interrupt Level Register
$YFFC05	QIVR	QSM Interrupt Vector Register

5.4.2.1 QSM Configuration Register (QMCR)

QMCR (Figure 5-3) contains parameters for interfacing to the CPU and the intermodule bus (IMB). This register can be modified only when the CPU is in supervisor mode.

QMCR — QSM Configuration Register   **$YFFC00**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
STOP	FRZ1	FRZ0	0	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

Figure 5-3. QSM Configuration Register

STOP — Stop Enable
   1 = QSM clock operation stopped
   0 = Normal QSM clock operation
STOP places the QSM into a low power state by disabling the system clock in most parts of the module. QMCR is the only register guaranteed to be readable while STOP is asserted. The QSPI RAM is not readable; however, writes to RAM or any register are guaranteed valid while STOP is asserted. STOP may be negated by the CPU and by reset.

The system software must stop each submodule before asserting STOP to avoid complications at restart and to avoid data corruption. The SCI submodule receiver and transmitter should be disabled, and the operation should be verified for completion before asserting STOP. The QSPI submodule should be stopped by asserting the HALT bit in SPCR3 and by asserting STOP after the HALTA flag is set.

FRZ1 — Freeze1
    1 = Halt the QSM (on a transfer boundary)
    0 = Ignore the FREEZE signal on the IMB
FRZ1 determines what action is taken by the QSM when the FREEZE signal of the IMB is asserted. FREEZE is asserted whenever the CPU enters the background mode.

Warning: *Ignoring the FREEZE signal can cause unpredictable results in the background mode operation of the QSM, because the CPU is unable to service interrupt requests in this mode.* If FRZ1 equals one when the FREEZE line is asserted, the QSM comes to an orderly halt on a transfer boundary as if HALT had been asserted. The output pins continue to drive their last state. Once the FREEZE signal is negated, the QSM module restarts automatically.

FRZ0 — Freeze 0
    Reserved for future enhancement.

Bits 12–8 — Not Implemented

SUPV — Supervisor/Unrestricted
    1 = Supervisor access
        All registers in the QSM are placed in supervisor-only space. For any access from within user mode, address acknowledge (AACK) is not returned and the bus cycle is transferred externally.
    0 = User access
        Because the QSM contains a mix of supervisor and user registers, AACK returns for accesses with either supervisor or user mode, and the bus cycle remains internal. If a supervisor-only register is accessed in user mode, the module responds as if an access had been made to an unimplemented register location.
    SUPV defines the assignable QSM registers as either supervisor-only data space or unrestricted data space.

Bits 6–4 — Not Implemented

IARB — Interrupt Arbitration Identification Number

Each module that generates interrupts, including the QSM, must have an IARB field. The value in this field is used to arbitrate for the IMB when two or more modules generate simultaneous interrupts of 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 QSM from arbitrating during an interrupt acknowledge cycle (IACK). The IARB field should be initialized by system software to a value between $F (top priority) and $1 (lowest priority). Otherwise, any interrupts generated are identified by the CPU as spurious.

5.4.2.2 QSM Test Register (QTEST)

QTEST (Figure 5-4) is used in testing the QSM. Accesses to QTEST must be made while the MCU is in test mode. Refer to **4.5 Test Submodule** for information on test mode.

QTEST — QSM Test Register   **$YFFC02**

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	TSBD	SYNC	TQSM	TMM

RESET:

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 5-4. QSM Test Register

TSBD — SPI Test Scan Path Select
    1 = Enable delay to SCK scan path
    0 = Enable SPI baud clock scan path

SYNC — SCI Baud Clock Synchronization Signal
    1 = Inhibit SCI source signal (QCSCl1)
    0 = Activate SCI source signal

TQSM — QSM Test Enable
    1 = Enable QSM to send test scan paths
    0 = Disable scan path

TMM — Test Memory Map
    1 = QSM responds to test memory addresses
    0 = QSM responds to QSM memory addresses

55.4.2.3 QSM Interrupt Level Register (QILR)

The QILR (Figure 5-5) determines the priority level of interrupts requested by the QSM and the vector used when acknowledging an interrupt. Separate fields exist to hold the interrupt levels for the QSPI and the SCI submodules. Priority is used to determine which interrupt is serviced first when two or more modules or external peripherals simultaneously request an interrupt. This register may only be accessed only when the CPU is in supervisor mode.

QILR — QSM Interrupt Level Register $YFFC04

15	14	13	12	11	10	9	8							0
0	0	ILQSPI			ILSCI			XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX						

RESET:

0	0	0	0	0	0	0	0
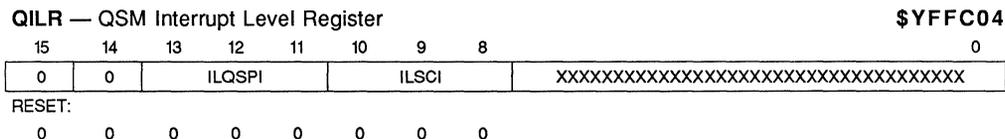
Figure 5-5. QSM Interrupt Level Register

ILQSPI — Interrupt Level for QSPI
    ILQSPI determines the priority level of all QSPI interrupts. This field should be programmed to a value between $0 (interrupts disabled) to $7 (highest priority). If both the QSPI and the SCI modules contain the same priority level (not equal to zero) and both modules simultaneously request interrupt servicing, the QSPI is given priority.

ILSCI — Interrupt Level of SCI
    LSCI determines the priority level of all SCI interrupts. This field should be programmed to a value between $0 (interrupts disabled) to $7 (highest priority).

5.4.2.4 QSM Interrupt Vector Register (QIVR)

At reset, QIVR (Figure 5-6) is initialized to $0F, which corresponds to the uninitialized interrupt vector in the exception table. This vector is selected until QIVR is written. QIVR should be programmed to one of the user-defined vectors ($40-$FF) during initialization of the QSM.

After initialization, QIVR determines which two vectors in the exception vector table are to be used for QSM interrupts. The QSPI and SCI submodules have separate interrupt vectors adjacent to each other. Both submodules use the same interrupt vector with the least significant bit (LSB) determined by the submodule causing the interrupt. The value of INTV0 used during an IACK cycle is supplied by the bus interface unit (BIU). During an IACK, INTV7–INTV1

are driven on the DATA7–DATA1 IMB lines. The BIU drives line DATA0 with a zero for an SCI interrupt and with a one for a QSPI interrupt. Writes to INTV0 have no meaning or effect. Reads of INTV0 return a value of one.

QIVR — QSM Interrupt Vector Register   **$YFFC05**

15		7	6	5	4	3	2	1	0
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX					INTV				

RESET:

| | | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 |
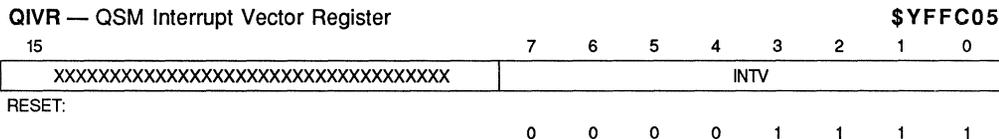
Figure 5-6. QSM Interrupt Vector Register

5.4.3 QSM Pin Control Registers

Table 5-5 identifies the three pin control registers of the QSM. The QSM determines the use of nine pins, eight of which form a parallel port on the MCU. Although these pins are used by the serial subsystems, any pin may alternately be assigned as general-purpose I/O on a pin-by-pin basis. For use of these pins as general-purpose I/O, they must not be assigned to the QSPI submodule in register QPAR. To avoid briefly driving incorrect data, the first byte to be output should be written before register QDDR is configured for any output pins. QDDR should then be written to determine the direction of data flow on the pins and to output the value contained in register QPDR for all pins defined as outputs. Subsequent data for output is then written to QPDR.

Table 5-5.   QSM Pin Control Registers

Address	Name	Usage
$YFFC15	QPDR	QSM Port Data Register
$YFFC16	QPAR	QSM Pin Assignment Register
$YFFC17	QDDR	QSM Data Direction Register

5.4.3.1 QSM Port Data Register (QPDR)

QPDR (Figure 5-7) determines the actual input or output value of a QSM port pin, if the pin is defined in QPAR as general-purpose I/O. All QSM port pins may be used as general-purpose I/O. Writes to this register affect the pins defined as outputs; reads of this register return the actual value of the pins.

15							7	6	5	4	3	2	1	0
RESERVED							D7 (TXD)	D6 (PCS3)	D5 (PCS2)	D4 (PCS1)	D3 (PCS0/$\overline{SS}$)	D2 (SCK)	D1 (MOSI)	D0 (MISO)

RESET:

| | | | | | | | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

Figure 5-7. QSM Port Data Register

5.4.3.2 QSM Pin Assignment Register (QPAR)

QPAR (Figure 5-8) determines which of the QSPI pins, with the exception of the SCK pin, are actually used by the QSPI submodule, and which pins are available for general-purpose I/O. Pins may be assigned to the QSPI or to function as general-purpose I/O on a pin-by-pin basis. QSPI pins designated by QPAR as general-purpose I/O are controlled only by QDDR and QPDR. The QSPI has no effect on these pins. QPAR does not affect the operation of the SCI submodule.

15	14	13	12	11	10	9	8	0
0	PCS3	PCS2	PCS1	PCS0/$\overline{SS}$	0	MOSI	MISO	XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

RESET:

| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | |

0 = General-purpose I/O
1 = QSPI module

Figure 5-8. QSM Pin Assignment Register

Bit 15 — Not Implemented
(TE in register SCCR1 determines whether the TXD pin is controlled by the SCI or functions as a general-purpose I/O pin.)

PCS3–PCS1 — Peripheral Chip Selects 3–1
PCS0/$\overline{SS}$ — Peripheral Chip Select 0/Slave Select
These bits determine whether the associated QSM port pins function as general-purpose I/O pins or are assigned to the QSPI submodule.

Bit 10 — Not Implemented
(When the QSPI is enabled, the SCK pin is required.)

MOSI — Master Out Slave In
MISO — Master In Slave Out
These bits determine whether the associated QSM port pin functions as a general-purpose I/O pin or is assigned to the QSPI submodule.

5.4.3.3 QSM Data Direction Register (QDDR)

QDDR (Figure 5-9) determines each I/O pin, except for TXD, as an input or an output regardless of whether the QSPI submodule is enabled or disabled. All QSM pins are configured during reset as general-purpose inputs. (The QSPI and SCI are disabled.) The RXD pin remains an input pin dedicated to the SCI submodule and does not function as a general-purpose I/O pin.

QDDR — QSM Data Direction Register $YFFC17

15		7	6	5	4	3	2	1	0
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX		TXD	PCS3	PCS2	PCS1	PCS0/$\overline{SS}$	SCK	MOSI	MISO

RESET.

| | | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
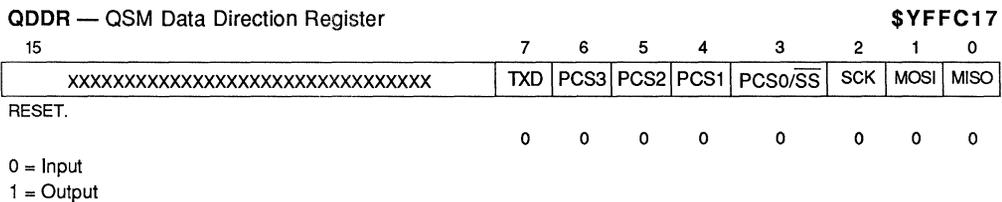
0 = Input
1 = Output

Figure 5-9. QSM Data Direction Register

TXD — Transmit Data
This bit determines the direction of the TXD pin (input or output), only if the SCI transmitter is disabled. If the SCI transmitter is enabled, the TXD bit is ignored, and the TXD pin is forced to function as an output.

PSC3–PCS1 — Peripheral Chip Selects 3–1
PSC0/SS — Peripheral Chip Select 0/Slave Select
SCK — Serial Clock
MOSI — Master Out Slave In
MISO — Master In Slave Out

All of these bits determine the QSPI port pin operation to be input or output.

5.5 QSPI Submodule

The QSPI submodule communicates with external peripherals and other MCUs via synchronous serial bus. The QSPI is fully compatible with the serial peripheral interface (SPI) systems found on other Motorola devices such as the

M68HC11 and M68HC05 Families. It has all of the capabilities of the SPI system as well as several new features. The following paragraphs describe the features, block diagram, pin descriptions, programmer's model (memory map) inclusive of registers, and the master and slave operation of the QSPI.

5.5.1 Features

Standard SPI features are listed below, followed by a list of the additional features offered on the QSPI:

- Full Duplex, Three-Wire Synchronous Transfers
- Half Duplex, Two-Wire Synchronous Transfers
- Master or Slave Operation
- Programmable Master Bit Rates
- Programmable Clock Polarity and Phase
- End-of-Transmission Interrupt Flag
- Master-Master Mode Fault Flag
- Easily Interfaces to Simple Expansion Parts (A/D converters, EEPROMS, display drivers, etc.)

QSPI-Enhanced features are as follows:

- Programmable Queue — up to 16 preprogrammed transfers
- Programmable Peripheral Chip Selects — four pins select up to 16 SPI chips
- Wraparound Transfer Mode — for autoscanning of serial A/D (or other) peripherals, with no CPU overhead
- Programmable Transfer Length — from 8–16 bits inclusive
- Programmable Transfer Delay — from 1 µs to 0.5 ms (at 16.78 MHz)
- Programmable Queue Pointer
- Continuous Transfer Mode — up to 256 bits

5.5.1.1 Programmable Queue

A programmable queue allows the QSPI to perform up to 16 serial transfers without CPU intervention. Each transfer corresponds to a queue entry containing all the information needed by the QSPI to independently complete one serial transfer. This unique feature greatly reduces CPU/QSPI interaction, resulting in increased CPU and system throughput.

5.5.1.2 Programmable Peripheral Chip Selects

Four peripheral chip-select pins allow the QSPI to access up to 16 independent peripherals by decoding the four peripheral chip-select signals. Up to four independent peripherals can be selected by direct connection to a chip-select pin. The peripheral chip selects simplify interfacing to two or more serial peripherals by providing dedicated peripheral chip-select signals, alleviating the need for CPU intervention.

5.5.1.3 Wraparound Transfer Mode

Wraparound transfer mode allows automatic, continuous re-execution of the preprogrammed queue entries. Newly transferred data replaces previously transferred data. Wraparound simplifies interfacing with A/D converters by automatically providing the CPU with the latest A/D conversions in the QSPI RAM. Consequently, serial peripherals appear as memory-mapped parallel devices to the CPU.

5.5.1.4 Programmable Transfer Length

The number of bits in a serial transfer is programmable from 8 to 16 bits, inclusive. For example, 10 bits could be used for communicating with an external 10-bit A/D converter. Likewise, a vacuum fluorescent display driver might require a 12-bit serial transfer. The programmable length simplifies interfacing to serial peripherals that require different data lengths.

5.5.1.5 Programmable Transfer Delay

An inter-transfer delay may be programmed from approximately 1 to 500 μs (using a 16.78-MHz system clock). For example, an A/D converter may require time between transfers to complete a new conversion. The default delay is 1 μs. The programmable length of delay simplifies interfacing to serial peripherals that require delay time between data transfers.

5.5.1.6 Programmable Queue Pointer

The QSPI has a pointer that identifies the queue location containing the data for the next serial transfer. The CPU can switch from one task to another in the QSPI by writing to the queue pointer, changing the location in the queue that is to be transferred next. Otherwise, the pointer increments after each serial transfer. By segmenting the queue, multiple-task support can be provided by the QSPI.

5.5.1.7 Continuous Transfer Mode

The continuous transfer mode allows the user to send and receive an uninterrupted bit stream with a peripheral. A minimum of 8 bits and a maximum of 256 bits may be transferred in a single burst without CPU intervention. Longer transfers are possible; however, minimal CPU intervention is required to prevent loss of data. A 1-μs pause (using a 16.78-MHz system clock) is inserted between each queue entry transfer.

5.5.2 Block Diagram

Figure 5-10 provides a block diagram of the QSPI submodule components.

5
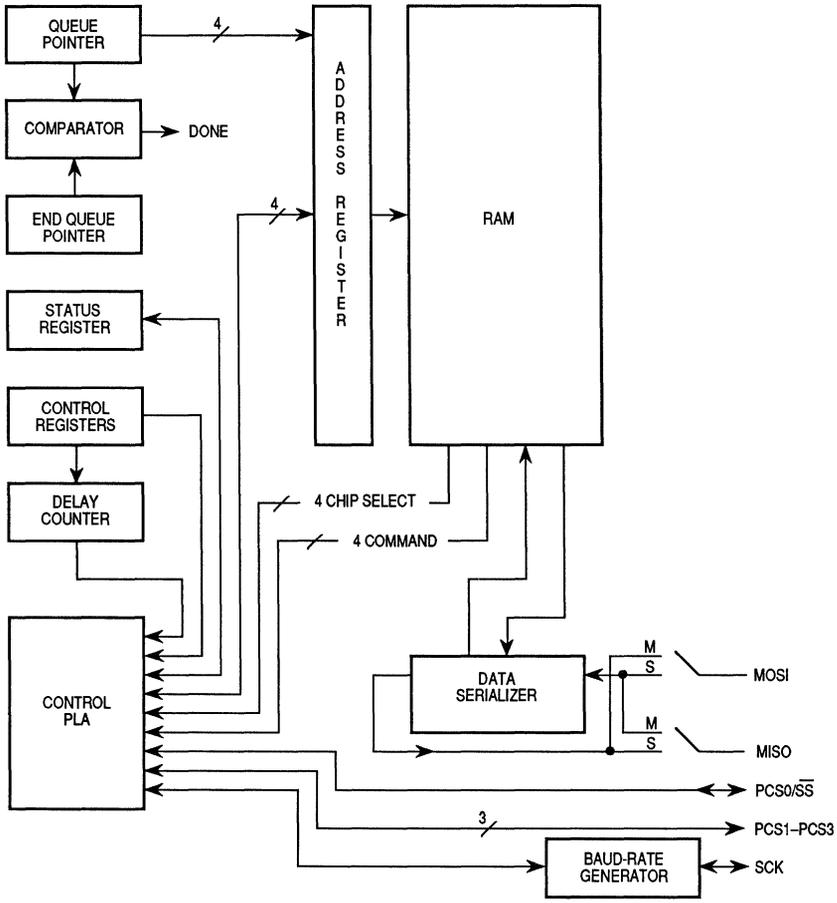
Figure 5-10. QSPI Submodule Diagram

QUEUED SERIAL MODULE

5.5.3 QSPI Pins

Seven pins are associated with the QSPI; however, when not needed for a QSPI application, they may be configured as general-purpose I/O pins. Figure 5-10 identifies the QSPI pins.

Table 5-6 lists the QSPI external input and output pins and their functions. QSM register QDDR determines whether the pins are designated as input or output. The user must initialize QDDR for the QSPI to function correctly.

Table 5-6. External Pin Inputs/Outputs to the QSPI

Pin Names	Mnemonics	Mode	Function
Master In Slave Out	MISO	Master Slave	Serial Data Input to QSPI Serial Data Output from QSPI
Master Out Slave In	MOSI	Master Slave	Serial Data Output from QSPI Serial Data Input to QSPI
Serial Clock	SCK[1]	Master Slave	Clock Output from QSPI Clock Input to QSPI
Peripheral Chip Selects	PCS3–PCS1	Master	Outputs Select Peripheral(s)
Peripheral Chip Select[2] Slave Select[3]	PCS0/ $\overline{SS}$	Master Slave	Output Selects Peripheral(s) Input Selects the QSPI
Slave Select[4]	$\overline{SS}$	Master	May Cause Mode Fault

NOTES:
1. All QSPI pins (except SCK) can be used as general-purpose I/O if they are not used by the QSPI while the QSPI is operating.
2. An output (PCS0) when the QSPI is in master mode.
3. An input ($\overline{SS}$) when the QSPI is in slave mode.
4. An input ($\overline{SS}$) when the QSPI is in master mode; useful in multimaster systems.

5.5.4 Programmer's Model and Registers

The programmer's model (memory map) for the QSPI submodule consists of the QSM global and pin control registers (refer to **5.4.2 QSM Global Registers** and **5.4.3 QSM Pin Control Registers**), four QSPI control registers, one status register, and the 80-byte QSPI RAM. Table 5-7 lists the registers and the QSPI RAM of the programmer's model. All of the registers and RAM can be read and written by the CPU. The four control registers must be initialized, in proper order before the QSPI is enabled to ensure defined operation. Only the control registers must adhere to the order of sequence prescribed in **5.4.1 Overall QSM Configuration Summary**. Write register SPCR1 last when setting up the QSPI, as this register contains the QSPI enable bit (SPE). Asserting this bit starts the QSPI. QSPI control registers are reset to a defined state and may then be changed by the CPU. Reset values are shown below each register.

Table 5-7. QSPI Registers

Address	Name	Usage
$YFFC18, 9	SPCR0	QSPI Control Register 0
$YFFC1A, B	SPCR1	QSPI Control Register 1
$YFFC1C, D	SPCR2	QSPI Control Register 2
$YFFC1E	SPCR3	QSPI Control Register 3
$YFFC1F	SPSR	QSPI Status Register
$YFFD00–1F	RAM	QSPI Receive Data (16 Words)
$YFFD20–3F	RAM	QSPI Transmit Data (16 Words
$YFFD40–4F	RAM	QSPI Command Control (8 Words)

In general, rewriting the same value into a control register does not affect the QSPI operation with the exception of NEWQP (bits 3-0) in SPCR2. Rewriting the same value to these bits causes the RAM queue pointer to restart execution at the designated location.

If control bits are to be changed, the CPU should halt the QSPI first. With the exception of SPCR2, writing a different value into a control register while the QSPI is enabled may disrupt operation. SPCR2 is buffered, preventing any disruption of the current serial transfer. After completion of the current serial transfer, the new SPCR2 values become effective.

5.5.4.1 QSPI Control Register 0 (SPCR0)

SPCR0 (Figure 5-11) contains parameters for configuring the QSPI before it is enabled. Although the CPU can read and write this register, the QSM has read-only access.

SPCR0 — QSPI Control Register 0 $YFFC18

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
MSTR	WOMQ	BITS				CPOL	CPHA	SPBR							

RESET:

| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 |
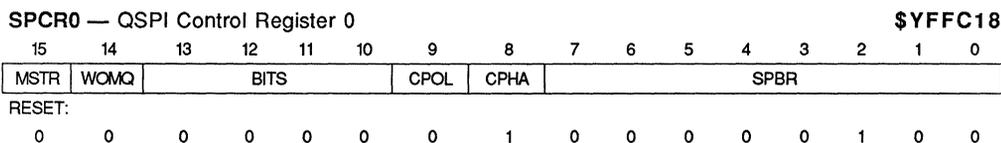
Figure 5-11. QSPI Control Register 0

MSTR — Master/Slave Mode Select

> 1 = QSPI is system master and can initiate transmission to external SPI devices.
>
> 0 = QSPI is a slave device, and only responds to externally generated serial transfers.

MSTR configures the QSPI for either master or slave mode operation. This bit is cleared on reset and may only be written by the CPU, not the QSM.

WOMQ — Wired-OR Mode for QSPI Pins

> 1 = All QSPI port pins designated as output by QDDR function as open-drain outputs and can be wire-ORed to other external lines.
>
> 0 = Output pins have normal outputs instead of open-drain outputs.

WOMQ allows the QSPI pins to be wire-ORed, regardless of whether they are used as general-purpose outputs or as QSPI outputs. WOMQ affects the QSPI pins whether the QSPI is enabled or disabled. This bit does not affect the SCI submodule transmit (TXD) pin, which has its own WOMS bit in an SCI control register.

BITS — Bits Per Transfer

In master mode, BITS determines the number of data bits transferred for each serial transfer in the queue that has the command control bit, BITSE of the QSPI RAM, equal to one. If BITSE equals zero for a command, 8 bits are transferred for that command regardless of the value in BITS. Data transfers from 8 to 16 bits are supported. Illegal (reserved) values all default to 8 bits. BITSE is not used in slave mode. All transfers are of the length specified by BITS. Table 5-8 shows the number of bits per transfer.

5

Table 5-8. Bits per Transfer if
Command Control Bit BITSE = 1

Bit 13	Bit 12	Bit 11	Bit 10	Bits per Transfer
0	0	0	0	16
0	0	0	1	Reserved
0	0	1	0	Reserved
0	0	1	1	Reserved
0	1	0	0	Reserved
0	1	0	1	Reserved
0	1	1	0	Reserved
0	1	1	1	Reserved
1	0	0	0	8
1	0	0	1	9
1	0	1	0	10
1	0	1	1	11
1	1	0	0	12
1	1	0	1	13
1	1	1	0	14
1	1	1	1	15

CPOL — Clock Polarity
   1 = The inactive state value of SCK is high.
   0 = The inactive state value of SCK is low.
CPOL is used to determine the inactive state value of the serial clock (SCK). CPOL is used in conjunction with CPHA to produce the desired clock-data relationship between master and slave device(s). For an understanding of the QSPI clock/data timing relationship, refer to the timing diagrams in Figures 5-24–5-27.

CPHA — Clock Phase
   1 = Data is changed on the leading edge of SCK and captured on the following edge of SCK.
   0 = Data is captured on the leading edge of SCK and changed on the following edge of SCK.
CPHA determines which edge of SCK causes data to change and which edge of SCK causes data to be captured. CPHA is used in conjunction with CPOL to produce the desired clock-data relationship between master and slave device(s). Note that CPHA is set at reset.

SPBR — Serial Clock Baud Rate

The QSPI internally generates the baud rate for SCK, the frequency of which is programmable by the user. The clock signal is derived from the MCU system clock using a modulus counter. At reset, BAUD is initialized to a 2.1-MHz SCK frequency.

The user programs a baud rate for SCK by writing a baud value from 2 to 255. The following equation determines the SCK baud rate:

$$\text{SCK Baud Rate} = \text{System Clock}/(2 \cdot \text{SPBR}) \qquad (5\text{-}1)$$

or

$$\text{SPBR} = \text{System Clock}/(2 \cdot \text{SCK Baud Rate Desired}) \qquad (5\text{-}2)$$

where SPBR equals 2, 3, 4, . . . , 255.

Programming SPBR with the values zero or one disables the QSPI baud rate generator. SCK is disabled and assumes its inactive state value. No serial transfers occur. SPBR has 254 active values. Table 5-9 lists several possible baud values and the corresponding SCK frequency based on a 16.78-MHz system clock.

Table 5-9. Examples of SCK Frequencies

System Clock Frequency	Required Division Ratio	Value of SPBR	Actual SCK Frequency
16.78 MHz	4	2	4.19 MHz
	8	4	2.10 MHz
	6	8	1.05 MHz
	34	17	493 kHz
	168	84	100 kHz
	510	255	33 kHz

5.5.4.2 QSPI Control Register 1 (SPCR1)

SPCR1 (Figure 5-12) contains parameters for configuring the QSPI before it is enabled. Although the CPU can read and write this register, the QSM has read access only, except for SPE. This bit is automatically cleared by the QSPI after completing all serial transfers or when a mode fault occurs.

SPCR1— QSPI Control Register $YFFC1A

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

RESET:

0	0	0	0	0	1	0	0	0	0	0	0	0	1	0	0

Figure 5-12. QSPI Control Register 1

SPE — QSPI Enable

    1 = The QSPI is enabled and the pins allocated by QSM register QPAR are controlled by the QSPI.

    0 = The QSPI is disabled, and the seven QSPI pins can be used as general-purpose I/O pins, regardless of the values in QPAR.

This bit enables or disables the QSPI Submodule. Setting SPE causes the QSPI to begin operation. If the QSPI is a master, setting SPE causes the QSPI to begin initiating serial transfers. If the QSPI is a slave, the QSPI begins monitoring the PCS0/SS pin to respond to the external initiation of a serial transfer.

When the QSPI is disabled, the CPU may use the QSPI RAM. When the QSPI is enabled, both the QSPI and the CPU have access to the QSPI RAM. The CPU has both read and write access capability to all 80 bytes of the QSPI RAM. The QSPI can read only the transmit data segment and the command control segment, and can write only the receive data segment of the QSPI RAM.

By clearing SPE, the QSPI turns itself off automatically when it is finished. An error condition called mode fault (MODF) also clears SPE. This error occurs when PCS0/SS is configured for input, the QSPI is a system master (MSTR = 1), and PCS0/SS is driven low externally.

To stop the QSPI, assert HALT, then wait until HALTA is set. SPE may then be safely cleared to zero, providing an orderly method of quickly shutting down the QSPI after the current serial transfer is completed. The CPU can immediately disable the QSPI by just clearing SPE; however, loss of data from a current serial transfer may result and confuse an external SPI device.

DSCKL — Delay before SCK

This bit determines the length of time the QSPI delays from peripheral chip select (PCS) valid to SCK transition for serial transfers in which the command control bit, DSCK of the QSPI RAM, equals one. PCS may be any of the four peripheral chip-select pins. The following equation determines the actual delay before SCK:

$$\text{PCS to SCK Delay} = [\text{DSCKL/System Clock Frequency}] \qquad (5\text{-}3)$$

where DSCKL equals {1,2,3, . . . 127}.

NOTE

A zero value for DSCKL causes a delay of 128/system clocks, which equals 7.6 μs for a 16.78-MHz system clock. Because of design limits, a DSCKL value of one defaults to the same timing as a value of two.

If a queue entry's DSCK equals zero, then DSCKL is not used. Instead, the PCS valid-to-SCK transition is one-half SCK period.

DTL — Length of Delay after Transfer

These bits determine the length of time that the QSPI delays after each serial transfer in which the command control bit, DT of the QSPI RAM, equals one. The following equation is used to calculate the delay:

$$\text{Delay after transfer} = [(32 \bullet \text{DTL})/\text{system clock frequency}] \qquad (5\text{-}4)$$

where DTL equals {1,2,3, . . . 255}.

NOTE

A zero value for DTL causes a delay-after-transfer value of (32 • 256)/system clock, which equals 488.5 μs with a 16.78-MHz system clock.

If DT equals zero, a standard delay is inserted.

$$\text{Standard Delay-after-Transfer} = [17/\text{System Clock}] \qquad (5\text{-}5)$$
$$= 1 \text{ μs with a 16.78-MHz System Clock}$$

Delay after transfer can be used to ensure that the deselect time requirement (for peripherals having such a requirement) is met. Some peripherals must be deselected for a minimum period of time between consecutive serial transfers. A delay after transfer can be inserted between consecutive transfers to a given

peripheral to ensure that its minimum deselect time requirement is met or to allow serial A/D converters to complete conversion before the next transfer is made.

5.5.4.3 QSPI Control Register 2 (SPCR2)

SPCR2 (Figure 5-13) contains parameters for configuring the QSPI. Although the CPU can read and write this register, the QSM has read access only. Writes to this register are buffered. A write to SPCR2 that changes any of the bit values (while the QSPI is operating) is ineffective on the current serial transfer, but becomes effective on the next serial transfer. Reads of SPCR2 return the actual current value of the register, not the buffer. Refer to **5.5.5 Operating Modes and Flowcharts** for a detailed description of this register.

SPCR2 — QSPI Control Register 2 $YFFC1C

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
SPIFIE	WREN	WRTO	0		ENDQP			0	0	0	0		NEWQP		

RESET:

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 5-13. QSPI Control Register 2

SPIFIE — SPI Finished Interrupt Enable
    1 = QSPI interrupts enabled
    0 = QSPI interrupts disabled
SPIFIE enables the QSPI to generate a CPU interrupt upon assertion of the status flag SFIP. Because of its special buffering, the value written to SPIFIE applies only upon completion of the queue (the transfer of the entry indicated by ENDQP). Thus, if a single sequence of queue entries is to be transferred (i.e., no WRAP), then SPIFIE should be set to the desired state before the first transfer.

If a subqueue (see bit NEWQP) is to be used, the same CPU write that causes a branch to the subqueue may enable or disable the SPIF interrupt for the subqueue. The primary queue retains its own selected interrupt mode, either enabled or disabled.

The SPIF interrupt must be cleared by clearing SPIF. Later interrupts may then be prevented by clearing SPIFIE to zero.

The QSPI has three possible interrupt sources, but only one interrupt vector. These sources are SPIF, MODF, and HALTA. When the CPU responds to a QSPI interrupt, the user must ascertain the exact interrupt cause by reading register

SPSR. Any interrupt that was set may then be cleared by writing to SPSR with a zero in the bit position corresponding to the exact interrupt source. Clearing SPIFIE does not immediately clear an interrupt already caused by SPIF.

WREN — Wrap Enable
    1 = Wraparound mode enabled
    0 = Wraparound mode disabled
WREN enables or disables wraparound mode. If enabled, the QSPI executes commands in the queue through the command contained in ENDQP. Execution continues at either address $0 or at the address found in NEWQP, depending on the state of WRTO. The QSPI continues looping until either WREN is negated, HALT is asserted, or SPE is negated. Once WREN is negated, the QSPI finishes executing commands through the command at the address contained in ENDQP, sets the SPIF flag, and stops. When WREN is set, SPIF is set each time the QSPI transfers the entry indicated by ENDQP.

WRTO — Wrap To
When wraparound mode is enabled and after the end of queue has been reached, WRTO determines which address the QSPI executes next. End of queue is determined by an address match with ENDQP. Execution wraps to address $0 if WRTO is not set, or to the address found in NEWQP if WRTO is set.

Bit 12 — Not Implemented

ENDQP — Ending Queue Pointer
This field determines the last absolute address in the queue to be completed by the QSPI. After completing each command, the QSPI compares the queue pointer value of the just-completed command with the value of ENDQP. If the two values match, the QSPI assumes it has reached the end of the programmed queue and sets the SPIF flag to so indicate.

The QSPI RAM queue has 16 entries: $0–$F. The user may program the NEWQP to start executing commands, beginning at any of the 16 addresses. Similarly, the user may program the ENDQP to stop execution of commands at any of the 16 addresses.

The queue is a circular data structure. If ENDQP is set to a lower address than NEWQP, the QSPI executes commands through address $F, and then continues execution at address $0 and so on until it stops after executing the command at address ENDQP. A maximum of 16 commands are executed before stopping, unless wraparound mode is enabled or unless the user modifies NEWQP and/or ENDQP.

The user may write a NEWQP value at any time, changing the flow of execution. ENDQP may also be written at any time, changing the length of the queue. Wraparound mode may also be enabled, causing continuous execution until the mode is disabled or the QSPI is halted.

Bits 7–4 — Not Implemented

NEWQP — New Queue Pointer Value
   NEWQP determines which queue entry the QSPI transfers first. NEWQP should be initialized before the QSPI is enabled with SPE. NEWQP may also be written while the QSPI is operating. When this happens, the QSPI completes transfer of the queue entry in progress and then immediately begins transferring queue entries starting with the entry indicated by the NEWQP.

   In this way, NEWQP provides additional functionality to the QSPI by providing a mechanism for supporting multiple queues or subqueues within the QSPI RAM. By changing the value in NEWQP, the user can cause the QSPI to execute a sequence of QSPI commands beginning at any location in the queue. Therefore, the user in advance is able to set up separate subqueues for different tasks within the QSPI RAM. By writing to NEWQP, selection between the different subqueues within the QSPI RAM is accomplished.

   If wraparound mode is enabled by setting WREN and WRTO in SPCR2, NEWQP assumes an additional function. When the end of the queue is reached, as determined by ENDQP, the address contained in NEWQP is used by the QSPI to wrap around to the first queue entry. The QSPI then re-executes the queued commands repeatedly until halted.

5.5.4.4 QSPI Control Register 3 (SPCR3)

   SPCR3 (Figure 5-14) contains parameters for configuring the QSPI. The CPU can read and write this register; the QSM has read-only access.

15	14	13	12	11	10	9	8		0
0	0	0	0	0	LOOPQ	HMIE	HALT	XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX	

RESET:

0	0	0	0	0	0	0	0

Figure 5-14. QSPI Control Register 3

Bits 15–11 — Not Implemented

LOOPQ — QSPI Loop Mode
    1 = Feedback path enabled
    0 = Feedback path disabled
LOOPQ enables or disables the feedback path on the data serializer for testing. If enabled, LOOPQ routes serial output data back into the data serializer, instead of received data. If disabled, LOOPQ allows regular received data into the data serializer. LOOPQ does not affect the QSPI output pins.

HMIE — HALTA and MODF Interrupt Enable
    1 = HALTA and MODF interrupts enabled
    0 = HALTA and MODF interrupts disabled
HMIE enables or disables QSPI interrupts to the CPU caused when either the HALTA status flag or the MODF status flag in SPSR is asserted. When HMIE is set, the assertion of either flag causes the QSPI to send a hardware interrupt to the CPU. When HMIE is clear, the asserted flag does not cause an interrupt.

HALT — Halt
    1 = Halt enabled
    0 = Halt not enabled
This bit is used by the CPU to stop the QSPI on a queue boundary. The QSPI halts in a known state from which it can later be restarted. When HALT is asserted by the CPU, the QSPI finishes executing the current serial transfer (up to 16 bits) and then halts. While halted, if the command control bit (CONT of the QSPI RAM) for the last command was asserted, the QSPI continues driving the peripheral chip-select pins with the value designated by the last command before the halt. If CONT was clear, the QSPI drives the peripheral chip-select pins to the value in QSM register QPDR.

If HALT is asserted during the last command in the queue, the QSPI completes the last command, asserts both HALTA and SPIF, and clears SPE. If the last queue command has not been executed, asserting HALT does not set SPIF nor clear SPE. QSPI execution continues when the CPU clears HALT.

5.5.4.5 QSPI Status Register (SPSR)

SPSR (Figure 5-15) contains QSPI status information. Only the QSPI can assert the bits in this register. The CPU reads this register to obtain status information and writes this register to clear status flags. CPU writes to CPTQP have no effect.

SPSR — QSPI Status Register $YFFC1F

15		7	6	5	4	3	2	1	0
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX		SPIF	MODF	HALTA	0		CPTQP		

RESET:

| | | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
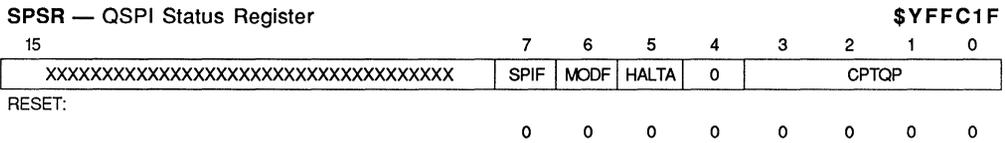
Figure 5-15. QSPI Status Register

SPIF — QSPI Finished Flag
   1 = QSPI finished
   0 = QSPI not finished
SPIF is set when the QSPI finishes executing the last command determined by the address contained in ENDQP in SPCR2. When the address of the command being executed matches the ENDQP, the SPIF flag is set after finishing the serial transfer.

If wraparound mode is enabled (WREN = 1), the SPIF is set, after completion of the command defined by ENDQP, each time the QSPI cycles through the queue. If SPIFIE in SPCR2 is set, an interrupt is generated when SPIF is asserted. Once SPIF is set, the CPU may clear it by reading SPSR followed by writing SPSR with a zero in SPIF.

MODF — Mode Fault Flag
   1 = Another SPI node requested to become the network SPI master while the QSPI was enabled in master mode (MSTR = 1), or the PCS0/SS pin was incorrectly pulled low by external hardware.
   0 = Normal operation
MODF is asserted by the QSPI when the QSPI is the serial master (MSTR = 1) and the slave select (PCS0/SS) input pin is pulled low by an external driver. This is possible only if the PCS0/SS pin is configured as input by QDDR. This low input to SS is not a normal operating condition. It indicates that a multimaster system conflict may exist, that another MCU is requesting to become the SPI network master, or simply that the hardware is incorrectly affecting PCS0/SS. SPE in SPCR1 is cleared, disabling the QSPI. The QSPI pins revert to control by QPDR. If

MODF is set and HMIE in SPCR3 is asserted, the QSPI generates an interrupt to the CPU.

The CPU may clear MODF by reading SPSR with MODF asserted, followed by writing SPSR with a zero in MODF. After correcting the mode fault problem, the QSPI can be re-enabled by asserting SPE.

The PCS0/SS pin may be configured as a general-purpose output instead of input to the QSPI. This inhibits the mode fault checking function. In this case, MODF is not used by the QSPI.

HALTA — Halt Acknowledge Flag
    1 = QSPI halted
    0 = QSPI not halted
HALTA is asserted by the QSPI when it has come to an orderly halt at the request of the CPU, via the assertion of HALT. To prevent undefined operation, the user should not modify any QSPI control registers or RAM while the QSPI is halted.

If HMIE in SPCR3 is set, the QSPI sends interrupt requests to the CPU when HALTA is asserted. The CPU can only clear HALTA by reading SPSR with HALTA set and then writing SPSR with a zero in HALTA.

Bit 4 — Not Implemented

CPTQP — Completed Queue Pointer
CPTQP contains the queue pointer value of the last command in the queue that was completed. The value of CPTQP is not updated until the command has been completed entirely. While the first command in a queue is executing, CPTQP contains either the reset value ($0) or the pointer to the last command completed in the previous queue.

If the QSPI is halted, CPTQP may be used to determine which commands have not been executed. The CPTQP may also be used to determine which locations in the receive data segment of the QSPI RAM contain valid received data.

5.5.4.6 QSPI Ram

The QSPI uses an 80-byte block of dual-access static RAM, which can be accessed by both the QSPI and the CPU. Because of sharing, the length of time taken by the CPU to access the QSPI RAM, when the QSPI is enabled, may be longer than when the QSPI is disabled. From one to four CPU wait states may be inserted by the QSPI in the process of reading or writing.

The size and type of access of the QSPI RAM by the CPU affects the QSPI access time. The QSPI is byte, word, and long-word addressable. Only word accesses of the RAM by the CPU are coherent accesses because these accesses are an indivisible operation. If the CPU makes a coherent access of the QSPI RAM, the QSPI cannot access the QSPI RAM until the CPU is finished. However, a long-word or misaligned word access is not coherent because the CPU must break its access of the QSPI RAM into two parts, which allows the QSPI to access the QSPI RAM in between the two accesses by the CPU.

The RAM is divided into three segments: receive data (REC.RAM), transmit data (TRAN.RAM), and command control (COMD.RAM). Receive data is information received from a serial device external to the MCU. Transmit data is information stored by the CPU for transmission to an external peripheral chip. Command control contains all the information needed by the QSPI to perform the transfer. Figure 5-16 illustrates the organization of the RAM.
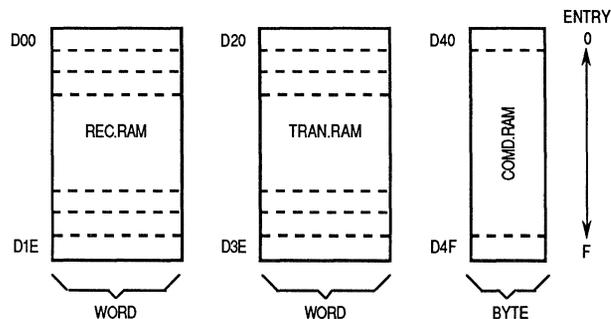
Figure 5-16. Organization of the QSPI RAM

Once the CPU has set up the queue of QSPI commands and enabled the QSPI, the QSPI operates independently of the CPU. The QSPI executes all of the commands in its queue, sets a flag indicating that it is finished, and then either interrupts the CPU or waits for CPU intervention.

5.5.4.6.1 Receive Data Ram (REC.RAM)

This segment of the RAM stores the data that is received by the QSPI from peripherals, SPI bus masters, or other MCUs. The CPU reads this segment of RAM to retrieve the data from the QSPI. Data stored in receive RAM is right-justified, i.e., the least significant bit is always in the right-most bit position within the word (bit 0) regardless of the serial transfer length. Unused bits in a receive

queue entry are set to zero by the QSPI upon completion of the individual queue entry. The CPU can access the data using byte, word, or long-word addressing.

The CPTQP value in SPSR shows which queue entries have been executed. The CPU uses this information to determine which locations in receive RAM contain valid data before reading them.

5.5.4.6.2 Transmit Data Ram (TRAN.RAM)

This segment of the RAM stores the data that is to be transmitted by the QSPI to peripherals. The CPU normally writes one word of data into this segment for each queue command to be executed. If the corresponding peripheral, such as a serial input port, is used solely to input data, then this segment does not need to be initialized.

Information to be transmitted by the QSPI should be written by the CPU to the transmit data segment in a right-justified manner. The information in the transmit data segment of the RAM cannot be modified by the QSPI. The QSPI merely copies the information to its data serializer for transmission to a peripheral. Information in transmit RAM remains there until it is re-written by the CPU.

5.5.4.6.3 Command Ram (COMD.RAM)

The command segment of the QSPI RAM is used only by the QSPI when it is in master mode. The CPU writes one byte of control information to this segment for each QSPI command to be executed. The information in the command RAM cannot be modified by the QSPI. It merely uses the information to perform the serial transfer.

COMD.RAM consists of 16 bytes. Each byte is divided into two fields. The first, the peripheral chip-select field, activates the correct serial peripheral during the transfer. The second, the command control field, provides transfer options specifically for that command/serial transfer. This feature gives the user more control over each transfer, providing the flexibility to interface to external SPI chips with different requirements (refer to Figure 5-17).

A maximum of 16 commands can be in the queue command control bytes. These bytes are assigned an address from $0–$F. Queue execution by the QSPI proceeds from the address contained in NEWQP through the address contained in ENDQP both of these fields are contained in SPCR2.

COMD.RAM — Command Ram   **$YFFD40**

7	6	5	4	3	2	1	0
CONT	BITSE	DT	DSCK	PCS3	PCS2	PCS1	PCS0*
—	—	—	—	—	—	—	—
CONT	BITSE	DT	DSCK	PCS3	PCS2	PCS1	PCS0*

$YFFD4F

COMMAND CONTROL | PERIPHERAL CHIP SELECT
*The PCS0 bit represents the dual-function PCS0/$\overline{\text{SS}}$.
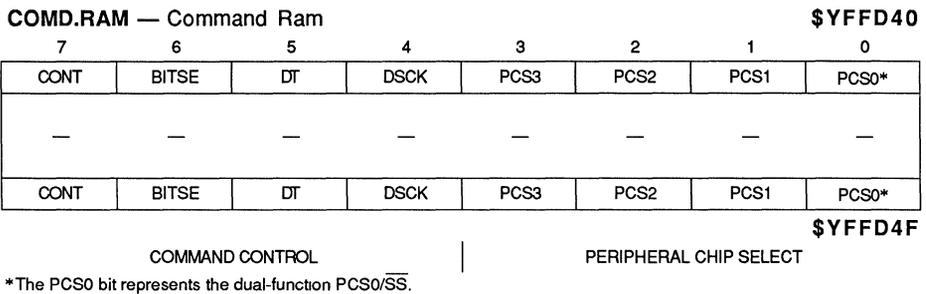
Figure 5-17. Command Ram

$\overline{\text{PCS3–PCS0/SS}}$ — Peripheral Chip Select

The four peripheral chip-select bits can be used directly to select one of four external chips for the serial transfer, or decoded by external hardware to select one of 16 chip-select patterns for the serial transfer. More than one peripheral chip select may be activated at a time, which is useful for broadcast messages in a multinode SPI system. More than one peripheral chip may be connected to each PCS pin. Care must be taken by the system designer not to exceed the maximum drive capability of the pins as defined in **SECTION 10 ELECTRICAL CHARACTERISTICS** for QSM pins.

QSM register QPDR determines the state of the PCS pins when the QSPI is disabled, and also determines the state of PCS pins that are not assigned to the QSPI when the QSPI is enabled. QPDR determines the state of pins assigned to the QSPI between transfers as well.

To use a peripheral chip-select pin, the CPU assigns the pin to the QSPI in QPAR by writing a one to the appropriate bit. The default value of the PCS pin should be written to QPDR. Next, the pin must be defined as an output in QDDR by setting the appropriate bit, which causes the pin to start driving the default value.

The QSPI RAM may then be initialized for a serial transmission, with the peripheral-chip-select bits of the command control byte appropriately configured to activate the desired PCS pin(s) during the serial transfer. When the command is executed, the PCS pin(s) are driven to the values contained in the appropriate control byte. After completing the serial transfer, the QSPI returns control of the peripheral-chip-select signal(s) (if CONT = 0 in the command control byte) to register QPDR.

CONT — Continue
    1 = Keep peripheral chip selects asserted after transfer is complete.
    0 = Return control of peripheral chip selects to QPDR after transfer is complete.
Some peripheral chips must be deselected between every QSPI transfer. Other chips must remain selected between several sequential serial transfers. CONT is designed to provide the flexibility needed to handle both cases.

If CONT = 1 and the peripheral-chip-select pattern for the next command is the same as that of the present command, the QSPI drives the PCS pins to the same value continuously during the two serial transfers. An unlimited number of serial transfers may be sent to the same peripheral(s) without deselecting it (them) by setting CONT = 1.

If CONT = 1 and the peripheral-chip-select pattern for the next command is different from that of the present command, the QSPI drives the PCS pins to the new value for the second serial transfer. Although this case is similar to CONT = 0, a difference remains. When CONT = 1, the QSPI continues to drive the PSC pins using the pattern from the first transfer until it switches to using the pattern for the second transfer.

When CONT = 0, the QSPI drives the PCS pins to the values found in register QPDR between serial transfers.

BITSE — Bits Per Transfer Enable
    1 = Number of bits set in BITS field of SPCR0
    0 = 8 bits

DT — Delay After Transfer
    A/D converters require a known amount of time to perform a conversion. The conversion time for serial CMOS A/D converters may range from 1–100 μs.

    To facilitate interfacing to peripherals with a latency requirement, the QSPI provides a programmable delay at the end of the serial transfer, with the DT field. The user may avoid using this delay option by executing transfers with other peripheral devices in between transfers with the peripheral that requires a delay. This interleaved operation improves the effective serial transfer rate.

    The amount of the delay between transfers is programmable by the user via the DTL field in SPCR1. The range may be set from 1-489 μs at 16.78 MHz.

DSCK — PCS to SCK Delay
    1 = DSCKL field in SPCR1 specifies value of delay from PCS valid to SCK.
    0 = PCS valid to SCK transition is 1/2 SCK.

5.5.5 Operating Modes and Flowcharts

The QSPI utilizes an 80-byte block of dual-access static RAM accessible by both the QSPI and the CPU. The RAM is divided into three segments: 16 command control bytes, 16 transmit data words of information to be transmitted, and 16 receive data words for data to be received. Once the CPU has a) set up a queue of QSPI commands, b) written the transmit data segment with information to be sent, and c) enabled the QSPI, the QSPI operates independently of the CPU. The QSPI executes all of the commands in its queue, sets a flag indicating completion, and then either interrupts the CPU or waits for CPU intervention.

The QSPI operates on a queue data structure contained in the QSPI RAM. Control of the queue is handled by three pointers: the new queue pointer (NEWQP), the completed queue pointer (CPTQP), and the end queue pointer (ENDQP). NEWQP, contained in SPCR2, points to the first command in the queue to be executed by the QSPI. CPTQP, contained in SPSR, points to the command last executed by the QSPI. ENDQP, also contained in SPCR2, points to the last command in the queue to be executed by the QSPI, unless wraparound mode is enabled (WREN = 1).

At reset, NEWQP is initialized to $0, causing QSPI execution to begin at queue address $0 when the QSPI is enabled (SPE = 1). CPTQP is set by the QSPI to the queue address ($0-$F) last executed, but is initialized to $0 at reset. ENDQP is also initialized to $0 at reset, but should be changed by the user to reflect the last queue entry to be transferred before enabling the QSPI. Leaving NEWQP and ENDQP set to $0 causes a single transfer to occur when the QSPI is enabled.

The organization of the QSPI RAM defines that one byte of command control data, one word of transmit data, and one word of receive data all correspond to one queue entry, $0-$F.

After executing the current command, ENDQP is checked against CPTQP for an end-of-queue condition. If a match occurs, the SPIF flag is set and the QSPI stops unless wraparound mode is enabled.

The QSPI operates in one of two modes: master or slave. Master mode is used when the MCU originates all data transfers. Slave mode is used when another MCU or a peripheral to the MCU initiates all serial transfers to the MCU via the QSPI. Switching between the two operating modes is achieved under software control by writing to the master (MSTR) bit in SPCR0.

In master mode, the QSPI executes the queue of commands as defined by the control bits in each entry. Chip-select pins are activated; data is transmitted, received, and placed in the QSPI RAM.

In slave mode, a similar operation occurs in response to the slave select ($\overline{SS}$) pin activated by an external SPI bus master. The primary differences are a) no peripheral chip selects are generated, and b) the number of bits transferred is controlled in a different manner. When the QSPI is selected, it executes the next queue transfer to correctly exchange data with the external device.

The following flowcharts, Figures 5-18–5-23, outline the operation of the QSPI for both master and slave modes. Note that the CPU must initialize the QSM global and pin registers and the QSPI control registers before enabling the QSPI for either master or slave operation. If using master mode, the necessary command control RAM should also be written before enabling the QSPI. Any data to be transmitted should also be written before the QSPI is enabled. When wrap mode is used, data for subsequent transmissions may be written at any time.

5
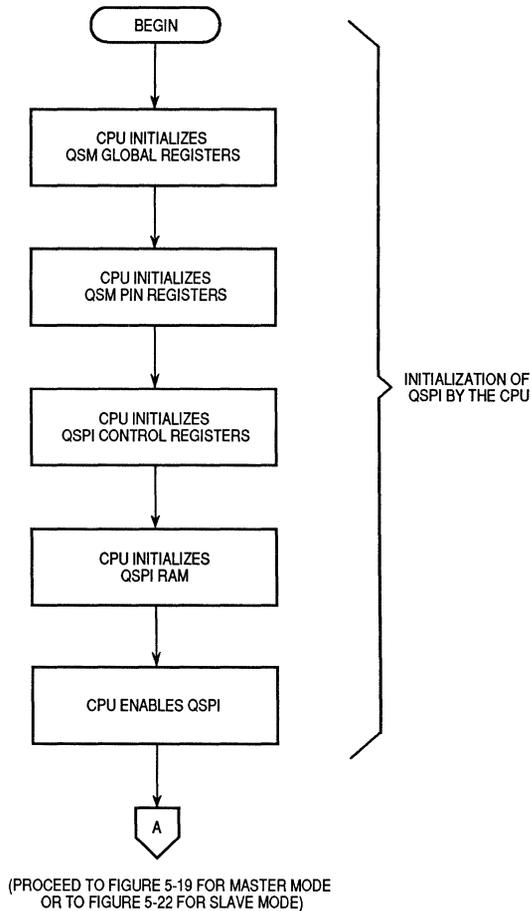
Figure 5-18. Flowchart of QSPI Initialization Operation

QSPI CYCLE BEGINS

```
        ┌───┐
        │ A │◄──────────────────┐
        └─┬─┘                    │
          ▼                      │
      ╱───────╲                  │
     ╱ IS QSPI ╲      Y          │
     ╲ DISABLED ╱──────────────────┘
      ╲───────╱
          │ N
          ▼
      ╱────────────╲     Y    ┌──────────────────────┐
     ╱  HAS NEWQP   ╲─────────►│    QUEUE POINTER     │
     ╲ BEEN WRITTEN ╱          │  CHANGED TO NEWQP    │
      ╲────────────╱           └──────────┬───────────┘
          │ N                             │
          ◄────────────────────────────────┘
          ▼
   ┌──────────────────────┐
   │  READ COMMAND CONTROL │
   │   AND TRANSMIT DATA   │
   │ FROM RAM USING QUEUE  │
   │   POINTER ADDRESS     │
   └──────────┬───────────┘
              ▼
   ┌──────────────────────┐
   │  ASSERT PERIPHERAL    │
   │    CHIP SELECT(S)     │
   └──────────┬───────────┘
              ▼
        ╱────────────╲    Y   ┌──────────────────────────┐
       ╱  IS PCS TO   ╲───────►│ EXECUTE PROGRAMMED DELAY │
       ╲  SCK DELAY   ╱        └────────────┬─────────────┘
       ╲ PROGRAMMED  ╱                      │
        ╲────────────╱                      │
              │ N                           │
              ▼                             │
   ┌──────────────────────┐                 │
   │ EXECUTE STANDARD DELAY│◄────────────────┘
   └──────────┬───────────┘
              ▼
   ┌──────────────────────┐
   │ EXECUTE SERIAL TRANSFER│
   └──────────┬───────────┘
              ▼
   ┌──────────────────────┐
   │  STORE RECEIVED DATA  │
   │  IN RAM USING QUEUE   │
   │   POINTER ADDRESS     │
   └──────────┬───────────┘
              ▼
           ┌───┐
           │ B │
           └───┘
```
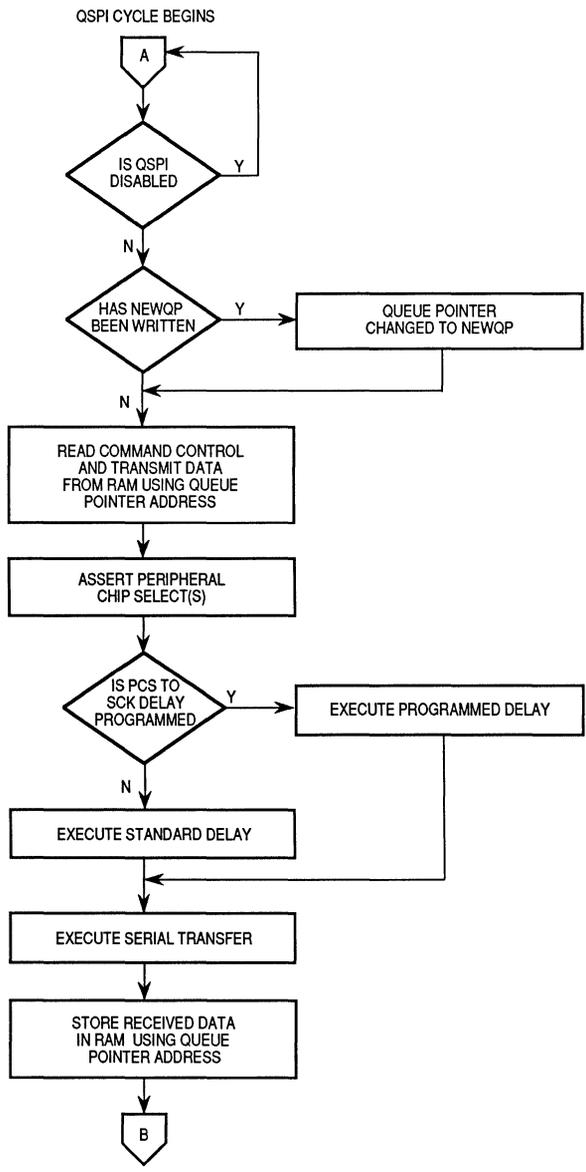
(CONTINUED ON NEXT PAGE)

Figure 5-19. Flowchart of QSPI Master Operation (Part 1)

```
                    ┌───┐
                    │ B │
                    └─┬─┘
                      ▼
        ┌───────────────────────────┐
        │   WRITE QUEUE POINTER      │
        │   TO CPTQP STATUS BITS     │
        └─────────────┬─────────────┘
                      ▼
                  ◇ IS CONTINUE ◇    Y
                  ◇ BIT ASSERTED ◇ ────────┐
                      │ N                   │
                      ▼                     │
        ┌───────────────────────────┐       │
        │   NEGATE PERIPHERAL        │       │
        │   CHIP SELECT(S)           │       │
        └─────────────┬─────────────┘◄──────┘
                      ▼
               ◇ IS DELAY ◇      Y    ┌──────────────────────────┐
               ◇ AFTER TRANSFER ◇ ───►│ EXECUTE PROGRAMMED DELAY │
               ◇ ASSERTED ◇           └──────────────────────────┘
                    │ N                              │
                    ▼                                │
        ┌───────────────────────────┐                │
        │   EXECUTE STANDARD DELAY   │                │
        └─────────────┬─────────────┘◄───────────────┘
                      ▼
                    ┌───┐
                    │ C │
                    └───┘
```
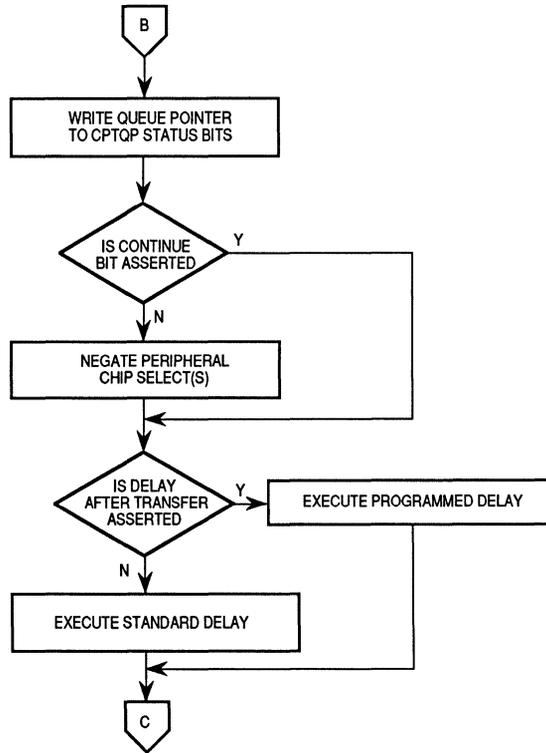
(CONTINUED ON NEXT PAGE)

Figure 5-20. Flowchart of QSPI Master Operation (Part 2)
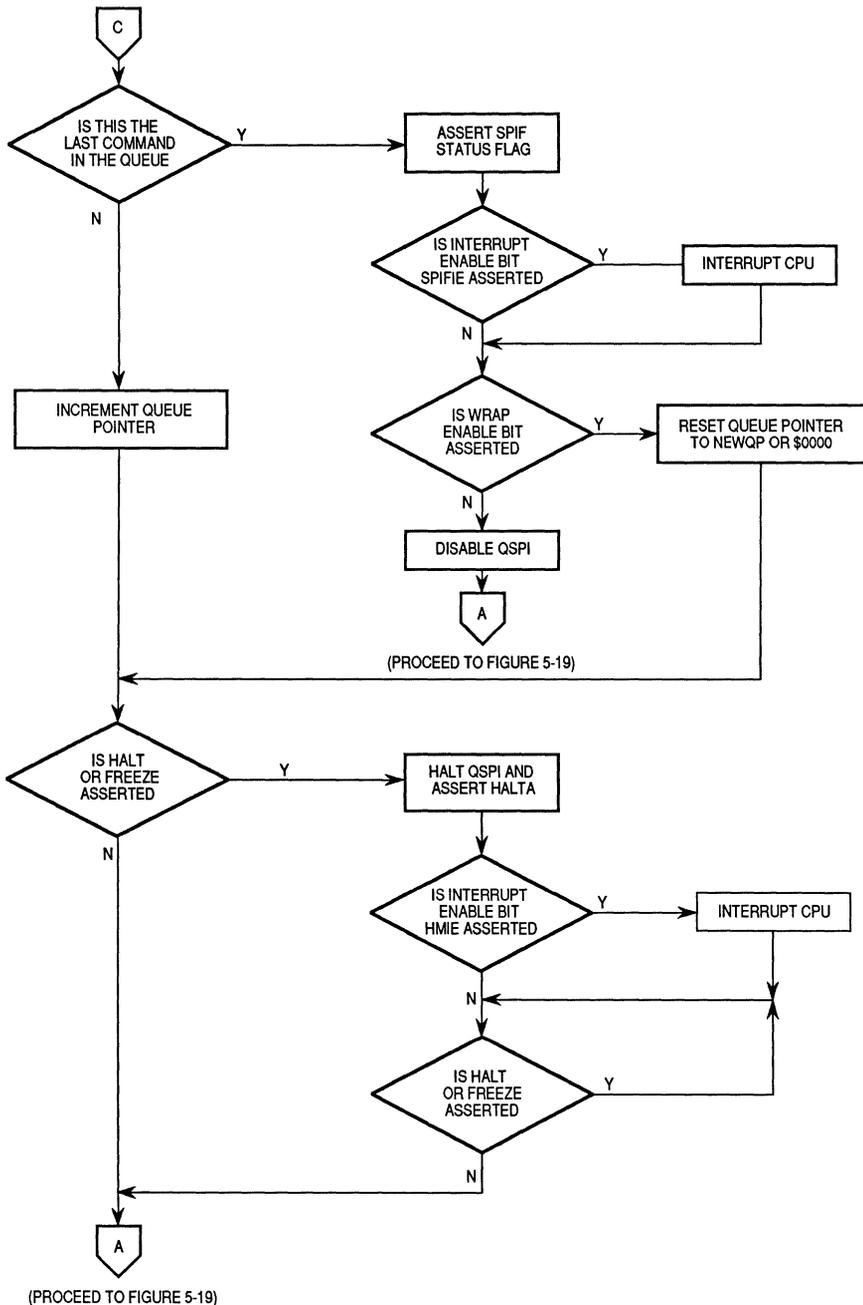
Figure 5-21. Flowchart of QSPI Master Operation (Part 3)
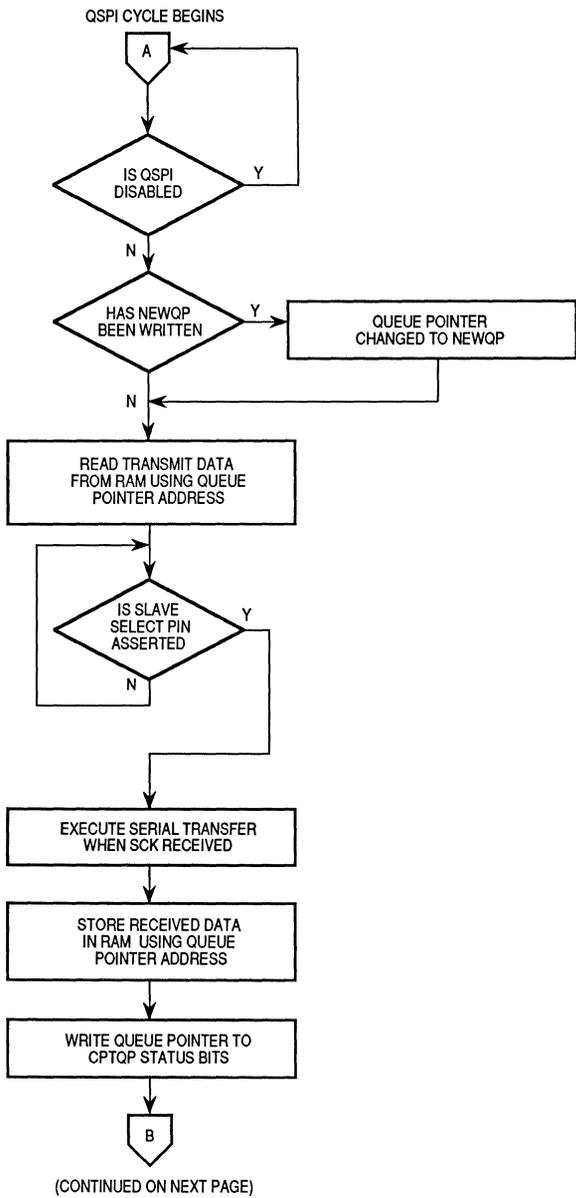
Figure 5-22. Flowchart of QSPI Slave Operation (Part 1)

B

IS THIS THE LAST COMMAND IN THE QUEUE — Y → ASSERT SPIF STATUS FLAG

N

IS INTERRUPT ENABLE BIT SPIFIE ASSERTED — Y → INTERRUPT CPU

N

INCREMENT QUEUE POINTER

IS WRAP ENABLE BIT ASSERTED — Y → RESET QUEUE POINTER TO NEWQP OR $0000

N

DISABLE QSPI

A

(PROCEED TO FIGURE 5-22)

IS HALT OR FREEZE ASSERTED — Y → HALT QSPI AND ASSERT HALTA

N

IS INTERRUPT ENABLE BIT HMIE ASSERTED — Y → INTERRUPT CPU

N

IS HALT OR FREEZE ASSERTED — Y

N

A

Figure 5-23. Flowchart of QSPI Slave Operation (Part 2)

Although the QSPI inherently supports multimaster operation, no special arbitration mechanism is provided. The user is given a mode fault flag (MODF) to indicate a request for SPI master arbitration; however, the system software must implement the arbitration. Note that unlike previous SPI systems, e.g., on the M68HC11 Family, MSTR is not cleared by a mode fault being set nor are the QSPI pin output drivers disabled; however, the QSPI is disabled when software clears SPE in QSPI register SPCR1.

Normally, the SPI bus performs simultaneous bidirectional synchronous transfers. The serial clock on the SPI bus master supplies the clock signal (SCK) to time the transfer of the bits. Four possible combinations of clock phase and polarity may be employed.

Data is transferred with the most significant bit first. The number of bits transferred per command defaults to eight, but may be programmed to a value from 8–16 bits, using the BITSE field.

Typically, outputs used for the SPI bus are not open drain unless multiple SPI masters are in the system. If needed, WOMQ in SPCR0 may be set to provide open-drain outputs. An external pullup resistor should be used on each output bus line. WOMQ affects all QSPI pins regardless of whether they are assigned to the QSPI or used as general-purpose I/O.

5.5.5.1 Master Mode

When operated in master mode, the QSPI may initiate serial transfers. The QSPI is unable to respond to any externally initiated serial transfers. QSM register QDDR should be written to direct the data flow on the QSPI pins used. The SCK pin should be configured as an output. Pins MOSI and PCS3–PCS0/$\overline{SS}$ should be configured as outputs as necessary. MISO should be configured as an input if necessary.

QSM register QPAR should be written to assign the necessary bits to the QSPI. The pins necessary for master mode operation are MISO and/or MOSI, SCK, and one or more of the PCS pins, depending on the number of external peripheral chips to be selected. MISO is used as the data input pin in master mode, and MOSI is used as the data output pin in master mode. Either or both may be necessary, depending on the particular application. SCK is the serial clock output in master mode.

PCS3-PCS0/$\overline{SS}$ are the select pins used to select external SPI peripheral chips for a serial transfer initiated by the QSPI. These pins operate as either active-

high or active-low chip selects. Other considerations for initialization are prescribed in **5.4.1 Overall QSM Configuration Summary**.

5.5.5.1.1 Master Mode Operation

After reset, the QSM registers and the QSPI control registers must be initialized as described above. In addition to the command control segment, the transmit data segment may, depending upon the application, need to be initialized. If meaningful data is to be sent out from the QSPI, the user should write the data to the transmit data segment before enabling the QSPI.

Shortly after SPE is set, the QSPI commences operation at the address indicated by NEWQP. The QSPI transmits the data found in the transmit data segment at the address indicated by NEWQP, and the QSPI stores received data in the receive data segment at the address indicated by NEWQP. Data is transferred synchronously with the internally generated SCK.

Transmit data is loaded into the data serializer (refer to Figure 5-9). The QSPI employs control bits, CPHA and CPOL, to determine which SCK edge the MISO pin uses to latch incoming data and which edge the MOSI pin uses to start driving the outgoing data. SPBR of SPCR0 determines the baud rate of SCK. DSCK and DSCKL determine any peripheral chip selects valid to SCK start delay.

The number of bits transferred is determined by BITSE and BITS fields. Two options are available: the user may use the default value of 8 bits, or the user may program the length from 8–16 bits, inclusive.

Once the proper number of bits are transferred, the QSPI stores the received data in the receive data segment, stores the internal working queue pointer value in CPTQP, increments the internal working queue pointer, and loads the next data required for transfer from the queue. The internal working queue pointer address is the next command executed unless the CPU writes a new value first.

If CONT is set and the peripheral-chip-select pattern does not change between the current and the pending transfer, the PCS pins are continuously driven in their designated state during and between both serial transfers. If the peripheral-chip-select pattern changes, then the first pattern is driven out during execution of the first transfer, followed by the QSPI switching to the next pattern of the second transfer when execution of the second transfer begins. If CONT is clear, the deselected peripheral-chip-select values (found in register QPDR) are driven out between transfers.

DT causes a delay to occur after the specified serial transfer is completed. The length of the delay is determined by DTL. When DT is clear, the standard delay (1 µs at a 16.78-MHz system clock) occurs after the specified serial transfer is completed.

5.5.5.1.2 Master Wraparound Mode

When the QSPI reaches the end of the queue, it always sets the SPIF flag whether wraparound mode is enabled or disabled. An optional interrupt to the CPU is generated when SPIF is asserted. At this point, the QSPI clears SPE and stops unless wraparound mode is enabled. A description of SPIFIE may be found in **5.5.4.3 QSPI Control Register 2 (SPCR2).**

In wraparound mode, the QSPI cycles through the queue continuously. Each time the end of the queue is reached, the SPIF flag is set. If the CPU fails to clear SPIF it remains set, and the QSPI continues to send interrupt requests to the CPU (assuming SPIFIE is set). The user may avoid causing CPU interrupts by clearing SPIFIE. As SPIFIE is buffered, clearing it after the SPIF flag is asserted does not immediately stop the CPU interrupts, but only prevents future interrupts from this source. To clear the current interrupt, the CPU must read QSPI register SPSR with SPIF asserted, followed by a write to SPSR with a zero in SPIF (clear SPIF).

Execution continues in wraparound mode, even while the QSPI is requesting interrupt service from the CPU. The internal working queue pointer increments to the next address, and the commands are executed again. SPE is not cleared by the QSPI. New receive data overwrites previously received data in the receive data segment.

Wraparound mode is properly exited in two ways: a) The CPU may disable wraparound mode by clearing WREN. The next time the end of the queue is reached, the QSPI sets SPIF, clears SPE, and stops; b) The CPU sets HALT. This second method halts the QSPI after the current transfer is completed, allowing the CPU to negate SPE. The CPU can immediately stop the QSPI by clearing SPE; however, this method is not recommended as it causes the QSPI to abort a serial transfer in process.

5.5.5.2 Slave Mode

When operating in slave mode, the QSPI may respond to externally initiated serial transfers. The QSPI is unable to initiate any serial transfers. Slave mode is typically used when multiple MCUs are in an SPI bus network, because only one device can be the SPI master (in master mode) at any given time.

QSM register QDDR should be written to direct data flow on the QSPI pins used. The MISO and MOSI pins, if needed, should be configured as output and input, respectively. Pins SCK and PCS0/$\overline{SS}$ should be configured as inputs.

QSM register QPAR should be written to assign the necessary bits to the QSPI. The pins necessary for slave mode operation are MISO and/or MOSI, SCK, and PCS0/$\overline{SS}$. MISO is the data output pin in slave mode, and MOSI is the data input pin in slave mode. Either or both may be necessary depending on the particular application. The serial clock (SCK) is the slave clock input in slave mode. PCS0/$\overline{SS}$ is the slave select pin used to select the QSPI for a serial transfer by the external SPI bus master when the QSPI is in slave mode. The external bus master selects the QSPI by driving PCS0/$\overline{SS}$ low. The command control segment is not implemented in slave mode; therefore, the CPU does not need to initialize it. This segment of the QSPI RAM and any other unused segments may be employed by the CPU as general-purpose RAM. Other considerations for initialization are prescribed in **5.4.1 Overall QSM Configuration Summary.**

5.5.5.2.1 Description of Slave Operation

After reset, the QSM registers and the QSPI control registers must be initialized as described above. Although the command control segment is not used, the transmit and receive data segments may, depending upon the application, need to be initialized. If meaningful data is to be sent out from the QSPI, the user should write the data to the transmit data segment before enabling the QSPI.

If SPE is set and MSTR is not set, a low state on the slave select (PCS0/$\overline{SS}$) pin commences slave mode operation at the address indicated by NEWQP. The QSPI transmits the data found in the transmit data segment at the address indicated by NEWQP, and the QSPI stores received data in the receive data segment at the address indicated by NEWQP. Data is transferred in response to an external slave clock input at the SCK pin.

Because the command control segment is not used, the command control bits and peripheral-chip-select codes have no effect in slave mode operation. The QSPI does not drive any of the four peripheral chip selects as outputs. PCS0/$\overline{SS}$ is used as an input.

Although CONT cannot be used in slave mode, a provision is made to enable receipt of more than 16 data bits. While keeping the QSPI selected (PCS0/$\overline{SS}$ is held low), the QSPI stores the number of bits, designated by BITS, in the current receive data segment address, increments NEWQP, and continues

storing the remaining bits (up to the BITS value) in the next receive data segment address.

As long as PCS0/$\overline{SS}$ remains low, the QSPI continues to store the incoming bit stream in sequential receive data segment addresses, until either the value in BITS is reached or the end-of-queue address is used with wraparound mode disabled. When the end of the queue is reached, the SPIF flag is asserted, optionally causing an interrupt. If wraparound mode is disabled, any additional incoming bits are ignored. If wraparound mode is enabled, storing continues at either address $0 or the address of NEWQP, depending on the WRTO value.

When using this capability to receive a long incoming data stream, the proper delay between transfers must be used. The QSPI requires time, approximately 1 μs at 16.78-MHz system clock, to prefetch the next transmit RAM entry for the next transfer. Therefore, the user may select a baud rate that provides at least a 1 μs delay between successive transfers to ensure no loss of incoming data. If the system clock is operating at a slower rate, the delay between transfers must be increased proportionately.

Because the BITSE option in the command control segment is no longer available, BITS sets the number of bits to be transferred for all transfers in the queue until the CPU changes the BITS value. As mentioned above, until PCS0/$\overline{SS}$ is negated (brought high), the QSPI continues to shift one bit for each pulse of SCK. If PCS0/$\overline{SS}$ is negated before the proper number of bits (according to BITS) is received, the QSPI, the next time it is selected, resumes storing bits in the same receive data segment address where it left off. If more than 16 bits are transferred before negating the PCS0/$\overline{SS}$, the QSPI stores the number of bits indicated by BITS in the current receive data segment address, then increments the address and continues storing as described above. Note that PCS0/$\overline{SS}$ does not necessarily have to be negated between transfers.

Once the proper number of bits (designated by BITS) are transferred, the QSPI stores the received data in the receive data segment, stores the internal working queue pointer value in CPTQP, increments the internal working queue pointer, and loads the new transmit data from the transmit data segment into the data serializer. The internal working queue pointer address is used the next time PCS0/$\overline{SS}$ is asserted, unless the CPU writes to the NEWQP first.

The DT and DSCK command control bits are not used in slave mode. As a slave, the QSPI does not drive the clock line nor the chip-select lines and, therefore, does not generate a delay.

In slave mode, the QSPI shifts out the data in the transmit data segment. The transmit data is loaded into the data serializer (refer to Figure 5-9) for transmission. This serializer shifts the 16 bits of data out in sequence, most significant bit first, as clocked by the incoming SCK signal. The QSPI uses CPHA and CPOL to determine which incoming SCK edge the MOSI pin uses to latch incoming data, and which edge the MISO pin uses to drive the data out.

The QSPI transmits and receives data until reaching the end of the queue (defined as a match with the address in ENDQP), regardless of whether PCS0/$\overline{SS}$ remains selected or is toggled between serial transfers. Receiving the proper number of bits causes the received data to be stored. The QSPI always transmits as many bits as it receives at each queue address, until the BITS value is reached or PCS0/$\overline{SS}$ is negated.

5.5.5.2.2 Slave Wraparound Mode

When the QSPI reaches the end of the queue, it always sets the SPIF flag, whether wraparound mode is enabled or disabled. An optional interrupt to the CPU is generated when SPIF is asserted. At this point, the QSPI clears SPE and stops unless wraparound mode is enabled. A description of SPIFIE bit can be found in **5.5.4.3 QSPI Control Register 2 (SPCR2)**.

In wraparound mode, the QSPI cycles through the queue continuously. Each time the end of the queue is reached, the SPIF flag is set. If the CPU fails to clear SPIF, it remains set, and the QSPI continues to send interrupt requests to the CPU (assuming SPIFIE is set). The user may avoid causing CPU interrupts by clearing SPIFIE. As SPIFIE is buffered, clearing it after the SPIF flag is asserted does not immediately stop the CPU interrupts, but only prevents future interrupts from this source. To clear the current interrupt, the CPU must read QSPI register SPSR with SPIF asserted, followed by a write to SPSR with zero in SPIF (clear SPIF).

Execution continues in wraparound mode even while the QSPI is requesting interrupt service from the CPU. The internal working queue pointer is incremented to the next address and the commands are executed again. SPE is not cleared by the QSPI. New receive data overwrites previously received data located in the receive-data segment.

Wraparound mode is properly exited in two ways: a) The CPU may disable wraparound mode by clearing WREN. The next time end of the queue is reached, the QSPI sets SPIF, clears SPE, and stops; and, b) The CPU sets HALT. This second method halts the QSPI after the current transfer is completed, allowing the CPU to negate SPE. The CPU can immediately stop

the QSPI by clearing SPE; however, this method is not recommended, as it causes the QSPI to abort a serial transfer in process.

5.5.5.3 QSPI Pin Timing

Table 5-10 and Figures 5-24–5-27 show the timing relationships for the QSPI pins. The figures are separated for master and slave mode timings. Both of these mode timings depend on the clock phase (CPHA) bit used. Although the clock polarity (CPOL) bit has no effect on the timing values, it does determine the inactive state of the serial clock.

Table 5-10. QSPI Pin Timing

Num	Function		Min	Max	Unit
	Operating Frequency				
	Master		DC	1/4	System Clock Frequency
	Slave		DC	1/4	System Clock Frequency
1	Cycle Time				
	Master		4	510	System Clocks
	Slave		TBD	—	System Clocks
2	Enable Lead Time				
	Master		2	128	System Clocks
	Slave		TBD	—	System Clocks
3	Enable Lag Time				
	Master		1/2	1/2	SCK
	Slave		TBD	—	SCK
4	Clock (SCK) High or Low Time				
	Master		2	255	System Clocks
	Slave		2	—	System Clocks
5	Sequential Transfer Delay				
	Master		17	8192	System Clocks
	Slave (Does Not Require Deselect)		13	—	System Clocks
6	Data Setup Time (Inputs)				
	Master	Nominal 50	TBD	—	ns
	Slave	Nominal 50	TBD	—	ns
7	Data Hold Time (Inputs)				
	Master	Nominal 50	TBD	—	ns
	Slave	Nominal 50	TBD	—	ns
8	Access Time				
	Slave		—	1/4	SCK
9	MISO Disable Time				
	Slave		—	1/2	SCK
10	Data Valid (after SCK Edge)* Nominal 50				
	Master		—	TBD	ns
	Slave		—	TBD	ns
11	Data Hold Time (Outputs)				
	Master		0	TBD	ns
	Slave		0	TBD	ns
12	Rise Time*				
	Outputs (SCK, MOSI, MISO, PCS3–PSC0)		—	TBD	ns
	Inputs (SCK, MOSI, MISO, SS)		—	TBD	µs
13	Fall Time*				
	Outputs (SCK, MOSI, PCS3–PCS0, MISO)		—	TBD	ns
	Inputs (SCK, MOSI, MISO, SS)		—	TBD	µs

*Assumes 200 pF load on all QSPI pins
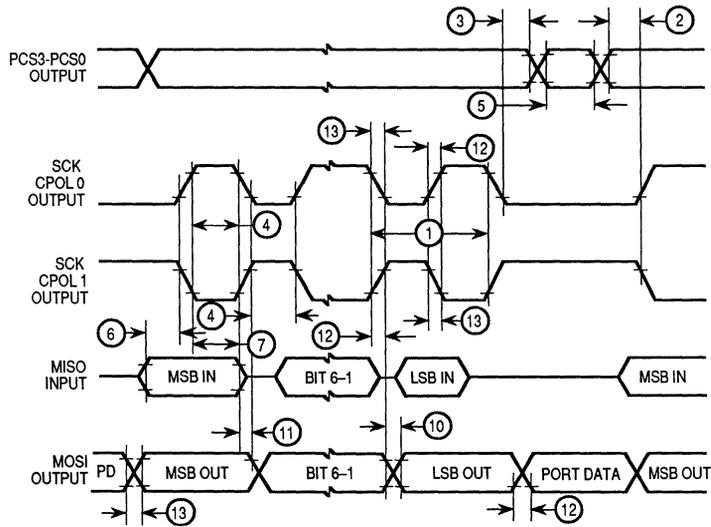
Figure 5-24. QSPI Timing Master, CPHA 0
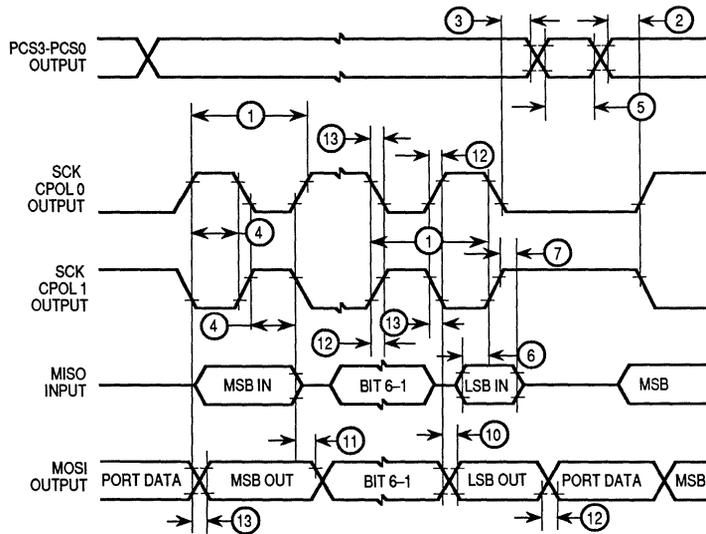
Figure 5-25. QSPI Timing Master, CPHA 1
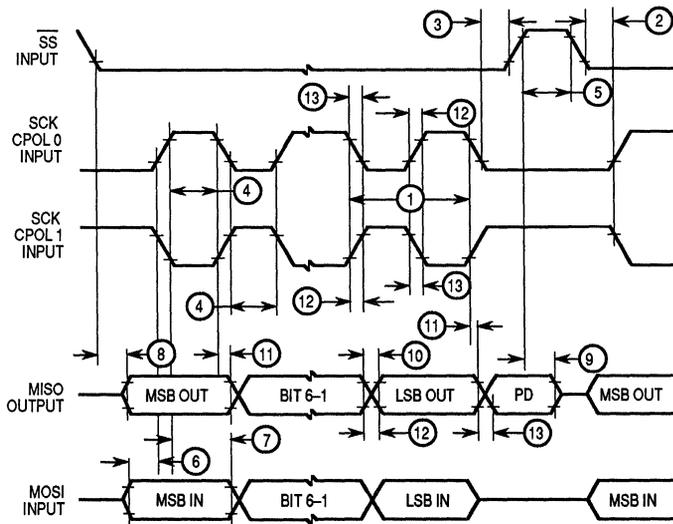
Figure 5-26. QSPI Timing Slave, CPHA 0
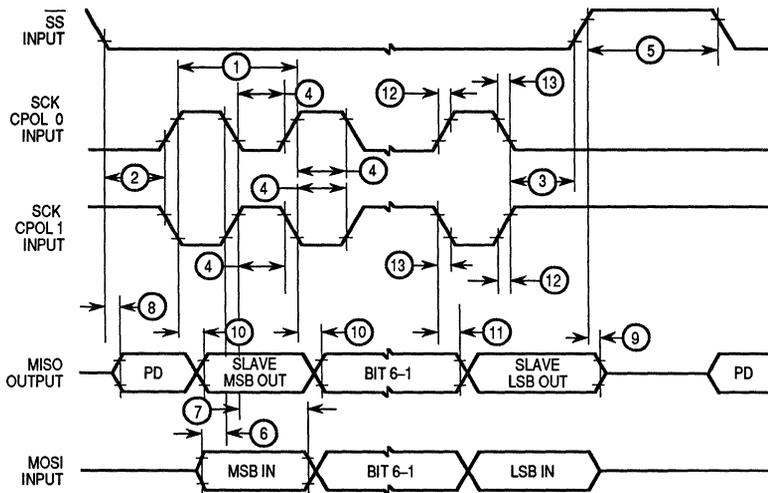
Figure 5-27. QSPI Timing Slave, CPHA 1

5.6 SCI Submodule

The SCI submodule is used to communicate with external devices and other MCUs via an asynchronous serial bus. The SCI is fully compatible with the SCI systems found on other Motorola MCUs such as the M68HC11 and M68HC05 Families. It has all of the capabilities of previous SCI systems as well as several significant new features. The following paragraphs describe the features, pins, programmer's model (memory map), registers, and the transmit and receive operations of the SCI.

5.6.1 Features

Standard SCI features are listed below, followed by a list of additional features offered.

Standard SCI Two-Wire System Features:
- Standard Nonreturn-to-Zero (NRZ) Mark/Space Format
- Advanced Error Detection Mechanism (detects noise duration up to 1/16 of a bit-time)
- Full-Duplex Operation
- Software Selectable Word Length (8- or 9-bit words)
- Separate Transmitter and Receiver Enable Bits
- May be Interrupt Driven
- Four Separate Interrupt Enable Bits

Standard SCI Receiver Features:
- Receiver Wakeup Function (idle or address mark bit)
- Idle-Line Detect
- Framing Error Detect
- Noise Detect
- Overrun Detect
- Receive Data Register Full Flag

Standard SCI Transmitter Features:
- Transmit Data Register Empty Flag
- Transmit Complete Flag
- Send Break

QSM-Enhanced SCI Two-Wire System Features:
- 13-Bit Programmable Baud-Rate Modulus Counter
- Even/Odd Parity Generation and Detection

QSM-Enhanced SCI Receiver Features:
- Two Idle-Line Detect Modes
- Receiver Active Flag

13-Bit Programmable Baud-Rate Modulus Counter
A baud rate modulus counter has been added to provide the user with more flexibility in choosing the crystal frequency for the system clock. The modulus counter allows the SCI baud rate generator to produce standard transmission frequencies for a wide range of system clocks. The user is no longer constrained to select crystal frequencies based on the desired serial baud rate. This counter provides baud rates from 64 baud to 524 kbaud with a 16.78-MHz system clock.

Even/Odd Parity Generation and Detection
The user now has the choice either of seven or eight data bits plus one parity bit, or of eight or nine data bits with no parity bit. Even or odd parity is available. The transmitter automatically generates the parity bit for a transmitted byte. The receiver detects when a parity error has occurred on a received byte and sets a parity error flag.

Two Idle-Line Detect Modes
Standard Motorola SCI systems detect an idle line when 10 or 11 consecutive bit times are all ones. Used with the receiver wakeup mode, the receiver can be awakened prematurely if the message preceding the start of the idle line contained ones in advance of its stop bit. The new (second) idle-line detect mode starts counting idle time only after a valid stop bit is received, which ensures correct idle-line detection.

Receiver Active Flag (RAF)
RAF indicates the status of the receiver. It is set when a possible start bit is detected and is cleared when an idle line is detected. RAF is also cleared if the start bit is determined to be line noise. This flag can be used to prevent collisions in systems with multiple masters.

5.6.2 SCI Pins

There are two unidirectional pins associated with the SCI. The SCI controls the transmit data (TXD) pin when enabled, while the receive data (RXD) pin remains a dedicated input pin to the SCI. TXD is available as a general-purpose I/O pin when the SCI transmitter is disabled; however, when used as a general-purpose I/O, TXD may be configured either as input or output as

determined by QSM register QDDR. Figure 5-1 illustrates these two pins. The SCI pins and their functions are listed in Table 5-11.

Table 5-11. External Pin Inputs/Outputs to the SCI

Pin Names	Mnemonics	Mode	Function
Receive Data	RXD	Receiver Disable	Not Used
		Receiver Enabled	Serial Data Input to SCI
Transmit Data	TXD	Transmitter Disabled	General-Purpose I/O
		Transmitter Enabled	Serial Data Output from SCI

5.6.3 Programmer's Model and Registers

The programmer's model (memory map) for the SCI submodule consists of the QSM global and pin control registers (refer to **5.4.2 QSM Global Registers and 5.4.3 QSM Pin Control Registers**) and the four SCI registers. The SCI registers are listed in Table 5-12 and consist of two control registers, one status register, and one data register. All registers may be read or written at any time by the CPU. Rewriting the same value to any SCI register does not disrupt operation; however, writing a different value into an SCI register when the SCI is running may disrupt operation. To change register values, the receiver and transmitter should be disabled with the transmitter allowed to finish first. The status flags in register SCSR may be cleared at any time.

Table 5-12. SCI Registers

Address	Name	Usage
$YFFC08	SCCR0	SCI Control Register 0
$YFFC0A	SCCR1	SCI Control Register 1
$YFFC0C	SCSR	SCI Status Register
$YFFC0E	SCDR	SCI Data Register Transmit Data Register (TDR)[*] Receive Data Register (RDR)[*]

[*]Reads access the RDR; writes access the TDR.

When initializing the SCI, the SCCR1 has two bits that should be written last, the transmitter enable (TE) and receiver enable (RE) bits, which enable the SCI. Registers SCCR0 and SCCR1 should both be initialized at the same time or before TE and RE are asserted. A single word write to SCCR1 can be used to initialize the SCI and enable the transmitter and receiver.

5.6.3.1 SCI Control Register 0 (SCCR0)

SCCR0 (Figure 5-28) contains the parameter for configuring the SCI baud rate. The baud rate should be set before the SCI is enabled. The CPU can read and write this register at any time.

SCCR0 — SCI Control Register 0   **$YFFC08**

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

RESET:

0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0
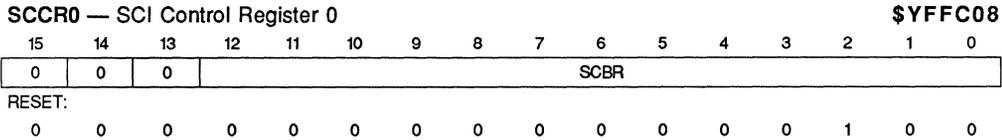
Figure 5-28. SCI Control Register 0

Bits 15-13 — Not Implemented

SCBR — Baud Rate
The SCI baud rate is programmed by writing a 13-bit value to SCBR and is derived from the MCU system clock using a modulus counter.

The SCI receiver operates asynchronously. Therefore, the SCI requires an internal clock to synchronize itself to the incoming data stream. The SCI baud-rate generator produces a receiver sampling clock with a frequency 16 times that of the expected baud rate of the incoming data. From transitions within the received waveform, the SCI determines the most likely position of the bit boundaries and adjusts sampling points to the proper positions within the bit period. The receiver sampling rate is always 16 times the frequency of the SCI baud rate, which is calculated using the following equation:

$$\text{SCI Baud} = \text{System Clock}/(32 \cdot \text{SCBR}) \tag{5-6}$$

where SCBR equals {1, 2, 3, . . . 8191}. Note that zero is a disallowed value for SCBR.

Writing a value of zero to SCBR disables the baud rate generator. There are 8191 different bauds available. The baud value depends on the value for SCBR and the system clock, as used in the above equation. Table 5-13 shows possible baud rates for a 16.78-MHz system clock. The maximum baud rate with this system clock speed is 524 kbaud.

Table 5-13. Examples of SCI Baud Rates

Nominal Baud Rate	Actual Baud Rate	Percent Error	Value of SCBR
500,000.00	524,288.00	4.86	1
38,400.00	37,449.14	–2.48	14
32,768.00	32,768.00	0.00	16
19,200.00	19,418.07	1.14	27
9,600.00	9,532.51	–0.70	55
4,800.00	4,809.98	0.21	109
2,400.00	2,404.99	0.21	218
1,200.00	1,199.74	–0.02	437
600.00	599.87	–0.02	874
300.00	299.94	–0.02	1,748
110.00	110.01	0.01	4,766
64.00	64.00	0.01	8,191

NOTE: These rates are based on a 16.78-MHz system clock.

More accurate baud rates can be obtained by varying the system clock frequency with the VCO synthesizer. Each VCO speed increment adjusts the baud rate up or down by 1/64 or 1.56%.

5.6.3.2 SCI Control Register 1 (SCCR1)

SCCR1 (Figure 5-29) contains parameters for configuration of the SCI. The CPU can read and write this register at any time. The SCI may modify the RWU bit in some circumstances. In general, the interrupts enabled by these control bits are cleared by reading the status register SCSR, followed by reading (for receiver status bits) or by writing (for transmitter status bits) the data register SCDR. For further detail refer to **5.6.4 Transmitter Operation** and **5.6.5 Receiver Operation**, respectively.

SCCR1 — SCI Control Register 1 $YFFC0A

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	LOOPS	WOMS	ILT	PT	PE	M	WAKE	TIE	TCIE	RIE	ILIE	TE	RE	RWU	SBK

RESET:

| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
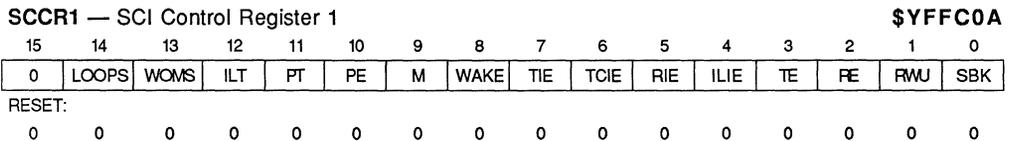
Figure 5-29. SCI Control Register 1

Bit 15 — Not Implemented

LOOPS — LOOP Mode
    1 = Test SCI operation, looping, feedback path enabled
    0 = Normal SCI operation, no looping, feedback path disabled
LOOPS controls a feedback path on the data serial shifter. If enabled, the output of the SCI transmitter is fed back into the receive serial shifter as receiver input, and no data is driven out of the TXD pin nor is data received from the RXD pin. The TXD pin is driven high (idle line). Both the transmitter and receiver must be enabled for loop mode to function.

WOMS — Wired-OR Mode for SCI Pins
    1 = If configured as an output, TXD is an open-drain output.
    0 = If configured as an output, TXD is a normal CMOS output.
WOMS determines whether the TXD pin is an open-drain output or a normal CMOS output. This bit is used only when TXD is an output. If the TXD pin is being used as a general-purpose input pin, WOMS has no effect.

ILT — Idle-Line Detect Type
    1 = Long idle-line detect (starts counting when the first one is received after a
        stop bit(s))
    0 = Short idle-line detect (starts counting when the first one is received)
ILT determines which one of two types of idle-line detection is to be used by the SCI receiver. The short idle-line detection circuitry causes the SCI receiver to start counting ones at any point (even during the frame), which means that the stop bit and any contiguous one data bits at the end of the last byte are counted toward the 10 or 11 ones in an idle frame. Hence, the data content of the last byte transmitted may affect the timing of idle-line detection.

The long idle-line detection circuitry causes the SCI receiver to start counting ones right after a stop bit, which means that the stop bit and any contiguous one data bits in a previous data byte are not counted toward the 10 or 11 ones in an idle line. Hence, the data content of the last byte transmitted does not affect the timing of idle-line detection.

PT — Parity Type
    1 = Odd parity
        If the data contains an even number of ones, then the parity bit equals one.
        If the data contains an odd number of ones, then the parity bit equals zero.
    0 = Even parity
        If the data contains an even number of ones, then the parity bit equals zero.
        If the data contains an odd number of ones, then the parity bit equals one.
When parity is enabled, PT determines whether parity is even or odd for both the receiver and the transmitter.

PE — Parity Enable
    1 = SCI parity enabled; the transmitter generates the parity bit and the receiver checks incoming parity.
    0 = SCI parity disabled
PE determines whether parity is enabled or disabled for both the receiver and the transmitter. If PE is set, the transmitter internally generates the parity bit and appends it to the data bits during transmission. The receiver checks the last bit before a stop bit to determine if the correct parity was received. If the received parity bit is not correct, the SCI sets the PF error flag in SCSR.

When PE is set, the most significant bit (MSB) of the data field is used for the parity function, which results in either seven or eight bits of user data, depending on the condition of M bit. Table 5-14 lists the available choices.

Table 5-14. M and PE Bit Fields

M	PE	Result
0	0	8 Data Bits
0	1	7 Data Bits, 1 Parity Bit
1	0	9 Data Bits
1	1	8 Data Bits, 1 Parity Bit

M — Mode Select
    1 = SCI frame: 1 start bit, 9 data bits, 1 stop bit (11 bits total)
    0 = SCI frame: 1 start bit, 8 data bits, 1 stop bit (10 bits total)
The M bit determines the SCI frame format. If M is clear (its reset value), the frame format is one start bit, eight data bits, one stop bit. If M is set, the frame format is one start bit, nine data bits, one stop bit.

The ninth data bit can be controlled by software to perform a function such as address mark. Frames with the ninth data bit set could be identified as an address

mark. All receivers in a network could be placed in wakeup mode until an address mark is detected, at which time all receivers would wakeup and read the address. All receivers being addressed could continue to receive the following message, while all receivers not being addressed could be put back into wakeup mode.

The ninth data bit could also serve as a second stop bit. By setting this bit permanently to one, communication with other SCIs requiring two stop bits could be accommodated.

Note that only 10 or 11 bits in a frame are allowed. If parity is to be enabled, the last data bit must be used for this purpose. The parity bit may be odd, even, mark, or space. Parity and address (control) bits are mutually exclusive. A choice must be made between one or the other, or neither. Every frame must have one start bit and at least one stop bit. The possible combinations are given in the bit description of PE.

WAKE — Wakeup by Address Mark
　　　　1 = SCI receiver awakened by address mark (eighth or ninth (last) bit set)
　　　　0 = SCI receiver awakened by idle-line detection
WAKE determines which one of two conditions wakes up the SCI receiver when it is in wakeup mode. If WAKE is clear (its reset value), the detection of an idle line (10 or 11 contiguous ones), which clears RWU, causes the SCI receiver to wake up. If WAKE is set, the detection of an address mark (the last data bit of a frame is set), which clears RWU, causes the SCI receiver to wake up.

TIE — Transmit Interrupt Enable
　　　　1 = SCI TDRE interrupts enabled
　　　　0 = SCI TDRE interrupts inhibited
When set, TIE enables an SCI interrupt whenever the TDRE flag in SCSR is set. The interrupt is blocked by negating TIE.

TCIE — Transmit Complete Interrupt Enable
　　　　1 = SCI TC interrupts enabled
　　　　0 = SCI TC interrupts inhibited
When set, TCIE enables an SCI interrupt whenever the TC flag in SCSR is set. The interrupt may be cleared by reading SCSR when TC is set and then by writing the transmit data register (TDR) of SCDR. The interrupt is blocked by negating TCIE.

RIE — Receiver Interrupt Enable

    1 = SCI RDRF interrupts enabled

    0 = SCI RDRF interrupts inhibited

When set, RIE enables an SCI interrupt whenever the RDRF flag in SCSR is set. The interrupt is blocked by negating RIE.

ILIE — Idle-Line Interrupt Enable

    1 = SCI IDLE interrupts enabled

    0 = SCI IDLE interrupts inhibited

When set, ILIE enables an SCI interrupt whenever the IDLE flag in SCSR is set. The interrupt is blocked by negating ILIE.

TE — Transmitter Enable

    1 = SCI transmitter enabled, TXD pin dedicated to the SCI transmitter

    0 = SCI transmitter disabled, TXD pin may be used as general-purpose I/O.

When set, TE enables the SCI transmitter and assigns to it the TXD pin. When TE is clear, the TXD pin may be used for general-purpose I/O. An idle frame, called a preamble, consisting of 10 (or 11) contiguous ones, is automatically transmitted whenever TE is changed from zero to one. Refer to **5.6.4 Transmitter Operation** for a detailed description of TE and the SCI transmit operation.

RE — Receiver Enable

    1 = SCI receiver enabled

    0 = SCI receiver disabled

RE enables the SCI receiver when set. When disabled, the receiver status bits RDRF, IDLE, OR, NF, FE, and PF are inhibited and are not asserted by the SCI. Refer to **5.6.5 Receiver Operation** for a complete description of RE and the SCI receiver operation.

RWU — Receiver Wakeup

    1 = Wakeup mode enabled, all received data ignored until awakened

    0 = Normal receiver operation, all received data recognized

Setting RWU enables the wakeup function, which allows the SCI to ignore received data until awakened by either an idle line or address mark (as determined by WAKE). When in wakeup mode, the receiver status flags are not set, and interrupts are inhibited. This bit is cleared automatically (returned to normal mode) when the receiver is awakened.

SBK — Send Break

    1 = Break frame(s) transmitted after completion of the current frame

    0 = Normal operation

SBK provides the ability to transmit a break code (10 or 11 contiguous zeros) from the SCI. When SBK is set, the SCI completes the current frame transmission (if it is

transmitting) and then begins transmitting continuous frames of 10 (or 11) zeros until SBK is cleared. If SBK is toggled by writing it first to a one and then immediately to a zero (in less than one serial frame interval), the transmitter sends only one or two break frames before reverting to mark (idle line) or before commencing to send data. SBK is normally used to broadcast the termination of a transmission.

5.6.3.3 SCI Status Register (SCSR)

SCSR (Figure 5-30) contains flags that the SCI sets to inform the user of various operational conditions. These flags are automatically cleared either by hardware or by a special acknowledgement sequence consisting of a SCSR read (either the upper byte, the lower byte, or the entire word) with a flag bit set, followed by a read (or write in the case of flags TDRE and TC) of data register SCDR (either the lower byte, or the entire word). An upper byte access of SCDR is only meaningful for reads. Note that a long-word read can consecutively access both registers SCSR and SCDR. This action clears the receive status flag bits that were set at the time of the read, but does not clear the TDRE or TC flags. To clear TDRE or TC, the SCSR read must be followed by a write to register SCDR (either the lower byte or the entire word).

If an internal SCI signal for setting a status bit comes after the CPU has read the asserted status bits but before the CPU has written or read register SCDR, the newly set status bit is not inadvertently cleared. Instead, register SCSR must be read again with the status bit set, and register SCDR must be written or read before the status bit is cleared.

NOTE
None of the status bits are cleared by reading a status bit while it is asserted and then by writing zero to that same bit. The procedure outlined above must be followed. Emphasis is also given to note that reading either byte of register SCSR causes all 16 bits to be accessed, and any status bits already set in either byte are armed to clear on a subsequent read or write of register SCDR.

As mentioned, register SCSR co-functions with register SCDR. SCDR is a combination of two data registers: the TDR and the RDR. Each of these data registers has a serial shifter.

SCSR — SCI Status Register $YFFC0C

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	TDRE	TC	RDRF	RAF	IDLE	OR	NF	FE	PF

RESET:

| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

Figure 5-30. SCI Status Register

Bits 15–9 — Not Implemented

TDRE — Transmit Data Register Empty Flag
    1 = A new character may now be written to register TDR.
    0 = Register TDR still contains data to be sent to the transmit serial shifter.
TDRE is set when the byte in register TDR is transferred to the transmit serial shifter. If this bit is zero, the transfer is yet to occur and a write to TDR will overwrite the previous value. New data is not transmitted if TDR is written without first clearing TDRE, which is accomplished by reading register SCSR with TDRE set, followed by a write to TDR. Reset sets this bit.

TC — Transmit Complete Flag
    1 = SCI transmitter is idle.
    0 = SCI transmitter is busy.
TC is set when the transmitter finishes shifting out all data, queued preambles (mark/idle line), or queued breaks (logic zero). TC is cleared when SCSR is read with TC set, followed by a write to register TDR.

RDRF — Receive Data Register Full Flag
    1 = Register RDR contains new data.
    0 = Register RDR is empty or contains previously read data.
RDRF is set when the content of the receive serial shifter is transferred to register RDR. If one or more errors are detected in the received word, the appropriate receive-related flag(s) NF, FE, and/or PF are set within the same clock cycle. RDRF is cleared when register SCSR is read with RDRF set, followed by a read of register RDR.

RAF — Receiver Active Flag
    1 = SCI receiver is busy.
    0 = SCI receiver is idle.
RAF indicates whether the SCI receiver is busy. This flag is set when the SCI receiver detects a possible start bit and is cleared when the chosen type of idle line is detected. RAF can be used to reduce collisions in systems with multiple masters.

The SCI receiver samples each start bit 16 times (at a rate of 16 times the baud rate). The 16 sample times are called RT1–RT16. RAF is set initially at RT1. The SCI receiver samples RT3, RT5, and RT7. If the receiver line is high during two or three of the three receive time (RT) samples, the start bit is considered invalid, and RAF is subsequently cleared. A more detailed description is found in **5.6.5.1 Receiver Bit Processor.**

IDLE — Idle-Line Detected Flag
    1 = SCI receiver detected an idle-line condition.
    0 = SCI receiver did not detect an idle-line condition.
IDLE is set when the SCI receiver detects an idle-line condition (reception of a minimum of 10 or 11 consecutive ones as specified by ILT in SCCR1). This bit is not set by the idle-line condition when RWU in SCCR1 is set. Once cleared, IDLE is not set again until after RDRF is set (after the line is active and becomes idle again). If a break is received, RDRF is set, allowing a subsequent idle line to be detected again. IDLE is cleared when SCSR is read with IDLE set, followed by a read of register RDR.

OR — Overrun Error Flag
    1 = RDRF is not cleared before new data arrives.
    0 = RDRF is cleared before new data arrives.
OR is set when a new byte is ready to be transferred from the receive serial shifter to register RDR, and RDR is already full (RDRF is still set). Data transfer is inhibited until OR is cleared. Previous data in RDR remains valid, but additional data received during an overrun condition (including the byte that set OR) is lost.

A difference exists between OR and the other receiver status flags. NF, FE, and PF all reflect the status of data already transferred to register RDR. OR reflects an operational condition that resulted in a loss of data to RDR. OR is cleared when SCSR is read with OR set, followed by a read of register RDR.

NF — Noise Error Flag
    1 = Noise occurred on the received data.
    0 = No noise detected on the received data.
NF is set when the SCI receiver detects noise on a valid start bit, on any of the data bits, or on the stop bit(s). It is not set by noise on the idle line or on invalid start bits. Each bit is sampled three times for noise. If the three samples are not at the same logic level, the majority value is used for the received data value, and NF is set. NF is not set until the entire frame is received and RDRF is set. Although an interrupt is not explicitly associated with NF, an interrupt may be generated with RDRF and NF checked in this manner. NF is cleared when SCSR is read with NF set, followed by a read of register RDR.

FE — Framing Error Flag

    1 = Framing error or break occurred on the received data.

    0 = No framing error on the received data.

FE is set when the SCI receiver detects a zero where a stop bit (one) was to occurr. A framing error results when the frame boundaries in the received bit stream are not synchronized with the receiver bit counter. FE is not set until the entire frame is received and RDRF is set. Although an interrupt is not explicitly associated with FE, an interrupt may be generated with RDRF and FE checked in this manner. A break can also cause FE to be set. FE is cleared when SCSR is read with FE set, followed by a read of register RDR.

PF — Parity Error Flag

    1 = Parity error occurred on the received data.

    0 = No parity error occurred on the received data.

PF is set when the SCI receiver detects a parity error. PF is not set until the entire frame is received and RDRF is set. Although an interrupt is not explicitly associated with PF, an interrupt may be generated with RDRF and PF checked in this manner. PF is cleared when SCSR is read with PF set, followed by a read of the register RDR.

5.6.3.4 SCI Data Register (SCDR)

SCDR (Figure 5-31) contains two data registers, both at the same address. The first register is the RDR, which is a read-only register. It contains data received over the SCI serial interface. Initially, data is received into the receive serial shifter and is transferred by the receiver into RDR. The second register is the SCI TDR, which is a write-only register. Data to be transmitted over the SCI serial interface is written to TDR. The transmitter transfers this data to the transmit serial shifter, adding on additional format bits before the data is sent out on the SCI serial interface.

SCDR — SCI Data Register $YFFC0E

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	R8/T8	R7/T7	R6/T6	R5/T5	R4/T4	R3/T3	R2/T2	R1/T1	R0/T0

RESET:

0	0	0	0	0	0	0	U	U	U	U	U	U	U	U	U

Figure 5-31. SCI Data Register

R8/T8 — Receive 8/Transmit 8

This bit is the ninth serial data bit received (R8) when the SCI system is configured for a 9-bit data operation (M = 1). When the SCI system is configured for an 8-bit data operation (M = 0), this bit has no meaning or effect.

This bit is the ninth serial data bit transmitted (T8) when the SCI system is configured for 9-bit data operation (M = 1). When the SCI system is configured for an 8-bit data operation (M = 0), this bit has no meaning or effect.

Accesses to the lower byte of SCDR triggers the mechanism for clearing the status bits or for initiating transmissions whether byte, word, or long-word accesses are used.

R0–R7/T0–T7 — Receive 0–7/Transmit 0–7

The first eight bits (7-0) contain the first eight data bits to be received (R0–R7) when SCDR is read, and also contain the first eight data bits to be transmitted (T0–T7) when SCDR is written.

5.6.4 Transmitter Operation

The transmitter consists of a transmit serial shifter and a parallel transmit data register (TDR) located in SCDR (refer to **5.6.3.4 SCI Data Register (SCDR)**). A character may be loaded into the TDR while another character is being shifted out, a capability called double buffering. The transmit serial shifter cannot be directly accessed by the CPU. The output of the transmit serial shifter is connected to the TXD pin whenever the transmitter is operating (TE = 1, or TE = 0 and transmitter operation not yet complete).

The following definitions apply to the transmitter and receiver operation:

Bit Time — The time required to serially transmit or receive one bit of data, which is equal to one cycle of the baud frequency.

Start Bit — One bit time of logic zero that indicates the beginning of a data frame. A start bit must begin with a one-to-zero transition and be preceded by at least three receive time (RT) samples of logic one.

Stop Bit — One bit time of logic one that indicates the end of a data frame.

Frame — A start bit, followed by a specified number of data or information bits, terminated by a stop bit. The number of data or information bits must agree between the transmitting and receiving devices. The most common frame format is one start bit followed by eight data bits (LSB

first) terminated by one stop bit, for a total of 10 bit times in the frame. The SCI optionally provides a 9-bit data format that results in an 11 bit-time frame.

The M bit in SCCR1 specifies the number of bit times in the frame (10 or 11). The most common format for nonreturn-to-zero (NRZ) serial interface is one start bit (logic zero or space), followed by eight data bits (terminated LSB first), by one stop bit (logic one or mark). In addition to this standard format, the SCI provides hardware support for a 9-bit data format. This format is one start bit, eight data bits (LSB first), a parity or address (control) bit, and one stop bit. Following are all the possible formats:

Start bit, seven data bits, two stop bits
Start bit, seven data bits, address bit, one stop bit
Start bit, seven data bits, address bit, two stop bits
Start bit, seven data bits, parity bit, one stop bit
Start bit, eight data bits, one stop bit
Start bit, eight data bits, two stop bits
Start bit, eight data bits, parity bit, one stop bit
Start bit, eight data bits, address, one stop bit

When the transmitter is enabled by writing a one to TE in SCCR1, a check is made to determine if the transmit serial shifter is empty. If empty (TC = 1), a preamble consisting of all ones (no start bits) is transmitted. If the transmit serial shifter is not empty (TC = 0), then normal shifting continues until the word in progress with stop bit(s) is sent. The preamble (an all ones frame) is then transmitted.

When TE is cleared, the transmitter is disabled only after all pending information is transmitted, including any data in the transmit serial shifter (inclusive of the stop bit), any queued preamble (idle frame), or any queued break (logic zero frame). The TC flag is set, and the TXD pin reverts to control by QPDR and QDDR. This function allows the user to terminate a transmission sequence in the following manner. After loading the last byte into register TDR and receiving the interrupt from TDRE in SCSR, (indicating that the data has transferred into the transmit serial shifter), the user clears TE. The last frame is transmitted normally, and the TXD pin reverts to control by QPDR and QDDR.

To insert a delimiter between two messages and place the nonlistening receivers in wakeup mode or to signal a retransmission (by forcing an idle line), TE is set to zero and then to one before the word in the transmit serial shifter has completed transmission. The transmitter waits until that word is transmitted and then starts transmission of a preamble (10 or 11 contiguous ones). After

the preamble is transmitted, and if TDRE is set (no new data to transmit), the line continues to mark (remain high). Otherwise, normal transmission of the next word begins.

Two SCI messages may be separated with minimum idle time by using a preamble of 10 bit times (11 if a 9-bit data format is specified) of marks (logic ones). The entire process can occur using the following procedure:

a. Write the last byte of the first message to the TDR.
b. Wait for TDRE to go high, indicating that the last byte is transferred to the transmit serial shifter.
c. Clear TE and then set TE back to one. This queues the preamble to follow immediately the stop bit of the current transmission.
d. Write the first byte of the second message to register TDR.

In this sequence, if the first byte of the second message is not transferred to register TDR prior to the finish of the preamble transmission, then the transmit data line (TXD pin) simply marks idle (logic one) until TDR is finally written. Also, if the last byte of the first message finishes shifting out (including the stop bit) and TE is clear, TC will go high and transmission will be considered complete. The TXD pin reverts to being a general-purpose I/O line.

The CPU writes data to be transmitted to register TDR, which automatically loads the data into the transmit serial shifter. Before writing to TDR, the user should check TDRE in SCSR. If TDRE = 0, then data is still waiting to be sent to the transmit serial shifter. Writing to TDR with TDRE clear overwrites previous data to be transferred. If TDRE = 1, then register TDR is empty, and new data may be written to TDR clearing TDRE.

As soon as the data in the transmit serial shifter has shifted out and if a new byte of data is in TDR (TDRE = 0), then the new data is transferred from register TDR to the transmit serial shifter, and TDRE is automatically set. An interrupt may optionally be generated at this point.

The data in the transmit serial shifter is prefixed by a start bit (logic zero) and suffixed by the ninth data bit, if M = 1, and by one stop bit. The ninth data bit can be used as normal data or as an extra stop bit. A parity bit is substituted if PE = 1. This data stream is shifted out over the TXD pin. When the data is completely shifted out and no preamble or send break is requested, then TC is set to one and the TXD pin remains high (logic one or mark).

Parity generation is enabled by setting PE in SCCR1 to a one. The last data bit, bit eight (or bit nine of the data if M = 1), is used as the parity bit, which is inserted between the normal data bits and the stop bit(s).

When TE is cleared, the transmitter yields control of the TXD pin in the following manner. If no information is being shifted out (i.e., if the transmitter is in an idle state, TC = 1), then the TXD pin reverts to being a general-purpose I/O pin. If a transmission is still in progress (TC = 0), the characters in the transmit serial shifter continue to be shifted out normally, followed by any queued break. When finished, TXD reverts to being a general-purpose I/O pin. To avoid terminating the transmitter before all data is transferred, the software should always wait for TDRE to be set before clearing TE.

Transmissions may be purposely aborted by the send break function. By writing SBK in SCCR1 to a one, a nonzero integer multiple of 10 bit times (11 if 9-bit data format is specified) of space (logic zero) is transmitted. If SBK is set while a transmission is in progress, the character in the transmit serial shifter finishes normally (including the stop bit) before the break function begins. Break frames are sent until either SBK or TE is cleared. To guarantee the minimum break time, SBK should be quickly toggled to one and then back to zero. After the break time, at least one bit time of mark idle (logic one) is transmitted to ensure that a subsequent start bit can be recognized.

The TXD pin has several control options to provide flexible operation. WOMS in SCCR1 can select either open-drain output (for wired-OR operation) or normal CMOS output. WOMS controls the function of the TXD pin whether the pin is being used for SCI transmissions (TE = 1) or as a general-purpose I/O pin.

In an SCI system with multiple transmitters, the wired-OR mode should be selected for the TXD pin of all transmitters, allowing multiple output pins to be coupled together. In the wired-OR mode, an external pullup resistor on the TXD pin is necessary.

In some systems, a mark (logic one) signal is desired on the TXD pin, even when the transmitter is disabled. This is accomplished by writing a one to QPDR in the appropriate position and configuring the TXD pin as an output in QDDR. When the transmitter releases control of the TXD pin, it reverts to driving a logic one output, which is the same as mark or idle.

5.6.5 Receiver Operation

The receiver can be divided into two segments. The first is the receiver bit processor logic that synchronizes to the asynchronous receive data and

evaluates the logic sense of each bit in the serial stream. The second receiver segment controls the functional operation and the interface to the CPU including the conversion of the serial data stream to parallel access by the CPU.

5.6.5.1 Receiver Bit Processor

The receiver bit processor contains logic to synchronize the bit time of the incoming data and to evaluate the logic sense of each bit. To accomplish this an RT clock, which is 16 times the baud rate, is used to sample each bit. Each bit time can thus be divided into 16 time periods called RT1–RT16. The receiver looks for a possible start bit by watching for a high-to-low transition on the RXD pin and by assigning the RT time labels appropriately.

When the receiver is enabled by writing RE in SCCR1 to one, the receiver bit processor logic begins an asynchronous search for a start bit. The goal of this search is to gain synchronization with a frame. The bit-time synchronization is done at the beginning of each frame so that small differences in the baud rate of the receiver and transmitter are not cumulative. The SCI also synchronizes on all one-to-zero transitions in the serial data stream, which makes the SCI tolerant to small frequency variations in the received data stream.

The sequence of events used by the receiver to find a start bit is listed below.

a. Sample RXD input during each RT period and maintain these samples in a serial pipeline that is three RT periods deep.
b. If RXD is low during this RT period, go to step 1.
c. If RXD is high during this RT period, store this sample and proceed to step 4.
d. If RXD is low during this RT period, but not high for the previous three RT periods (which is noise only), set an internal working noise flag and go to step 1, since this transition was not a valid start-bit transition.
e. If RXD is low during this RT period and has been high for the previous three RT periods, call this period RT1, set RAF, and proceed to step 6.
f. Skip RT2 but place RT3 in the pipeline and proceed to step 7.
g. Skip RT4 and sample RT5. If both RT3 and RT5 are high (RT1 was noise only), then set an internal working noise flag. Go to step 3 and clear RAF. Otherwise, place RT5 in the pipeline and proceed to step 8.
h. Skip RT6 and sample RT7. If any two of RT3, RT5, or RT7 is high (RT1 was noise only), set an internal working noise flag. Go to step 3 and clear RAF. Otherwise, place RT7 in the pipeline and proceed to step 9.
i. A valid start bit is found and synchronization is achieved. From this point on until the end of the frame, the RT clock will increment starting over again with RT1 on each one-to-zero transition or each RT16. The

beginning of a bit time is thus defined as RT1 and the end of a bit time as RT16.

Upon detection of a valid start bit, synchronization is established and is maintained through the reception of the last stop bit, after which the procedure starts all over again to search for a new valid start bit. During a frame's reception, the SCI resynchronizes the RT clock on any one-to-zero transitions.

Additional logic in the receiver bit processor determines the logic level of the received bit and implements an advanced noise-detection function. During each bit time of a frame (including the start and stop bits), three logic-sense samples are taken at RT8, RT9, and RT10. The logic sense of the bit time is decided by a majority vote of these three samples. This logic level is shifted into register RDR for every bit except the start and stop bits.

If RT8, RT9, and RT10 do not all agree, an internal working noise flag is set. Additionally for the start bit, if RT3, RT5, and RT7 do not all agree, the internal working noise flag is set. If this flag is set for any of the bit times in a frame, the NF flag in SCSR is set concurrently with the RDRF flag in SCSR when the data is transferred to register RDR. The user must determine if the data received with NF set is valid. Noise on the RXD pin does not necessarily corrupt all data.

The operation of the receiver bit processor is shown in the following figures. These examples demonstrate the search for a valid start bit and the synchronization procedure as outlined above. The possibility of noise durations greater than one bit time are not considered in these examples. Figure 5-32 illustrates the ideal case with no noise present.


Figure 5-32. Start Search Example 1

Figure 5-33 shows the start-bit search and resynchronization process being restarted because the first low detected was determined to be noise rather than the beginning of a start bit time. Since the noise occurred before the start bit was found, it will not cause the internal working noise flag to be set.



```
                    |←————————— PERCEIVED START BIT ————————→|
                    |←——————————— ACTUAL START BIT ——————————→|←— LSB —
  _____    __          _____    __/
             |__|  |_____|

  1  1  1  0   1  1  1  0   0   0   0 0 0 0
  ^  ^  ^  ^   ^  ^  ^  ^   ^   ^   ^ ^ ^ ^ ^ ^ ^ ^ ^ ^ ^ ^ ^ ^ ^ ^ ^
  R* R* R* R   R  R  R  R* R* R  R  R R R R R R R R R R R R R* R R R
  T  T  T  T   T  T  T  T  T  T  T  T T T T T T T T T T T T T  T T T
  1  1  1  1   2  3  4  5  1  1  2  3 4 5 6 7 8 9 1 1 1 1 1 1  1 2 3
                                                 0 1 2 3 4 5  6

  * Restart RT Clock
```

Figure 5-33. Start Search Example 2

Figure 5-34 shows that noise is perceived as the beginning of a start bit. Note that the high level sensed at RT3 causes the internal working noise flag to be set. Even though this figure shows improper alignment of the perceived bit-time boundaries to the actual bit-time boundaries, the logic sense samples taken at RT8, RT9, and RT10 fall well within the correct actual bit time. The start bit and all other bits in the frame should be received correctly.



```
                  |←————————— PERCEIVED START BIT ————————→|
                : |←————————————— ACTUAL START BIT ——————————→|←— LSB —
  _____    __          _____    __/
             |__|  |_____|

  1  1  1  1  1:0   1   0   0 0 0 0
  ^  ^  ^  ^  ^  ^  ^  ^   ^  ^  ^  ^ ^ ^ ^ ^ ^ ^ ^ ^ ^:^ ^ ^ ^ ^ ^ ^
  R* R* R* R* R* R  R  R   R  R  R  R R R R R R R R R R R* R R R R R R
  T  T  T  T  T  T  T  T   T  T  T  T T T T T T T T T T T  T T T T T T
  1  1  1  1  1  1  2  3   4  5  6  7 8 9 1 1 1 1 1 1 1 1  2 3 4 5 6 7
                                        0 1 2 3 4 5 6

  * Restart RT Clock
```

Figure 5-34. Start Search Example 3

Figure 5-35 shows how a large burst of noise is perceived as the beginning of a start bit. Note that RT5 is sensed logic high, setting the internal working noise flag. This figure also illustrates a worst-case alignment of the perceived bit-time

boundaries to the actual bit-time boundaries; however, RT8, RT9, and RT10 all fall within the correct actual bit time. The start bit is detected and the incoming data stream is correctly sensed.

```
          |←———————— PERCEIVED START BIT ————————→|
          :          |←————————— ACTUAL START BIT ——————————→|← LSB –

 ___    __    __    _____      __
    |__|  |__|  |__|                                        |____/

  1  1  1 :0    0    1    0  0  0  0                    :
  ^  ^  ^  ^    ^    ^    ^  ^  ^  ^  ^  ^  ^  ^  ^  ^  ^  ^  ^  ^  ^  ^  ^  ^  ^  ^
  R* R* R* R    R    R    R  R  R  R  R  R  R  R  R  R  R* R  R  R  R  R  R  R  R  R
  T  T  T  T    T    T    T  T  T  T  T  T  T  T  T  T  T  T  T  T  T  T  T  T  T  T
  1  1  1  1    2    3    4  5  6  7  8  9  1  1  1  1  1  1  1  2  3  4  5  6  7  8  9
                                           0  1  2  3  4  5  6
  * Restart RT Clock
```

Figure 5-35. Start Search Example 4

Figure 5-36 illustrates the effect of noise early within the start bit time. Although this noise does not affect proper synchronization with the start bit time, it does set the internal working noise flag.

```
              |←————————— PERCEIVED START BIT ———————→|
              |←————————— ACTUAL START  BIT ———————→|← LSB –

 _____      __ _____      __
                        |____|  |                            |____/

  1  1  1  1  1  1  1  1  0    1    0    0  0  0  0
  ^  ^  ^  ^  ^  ^  ^  ^  ^  ^  ^  ^  ^  ^  ^  ^  ^  ^  ^  ^  ^  ^  ^  ^  ^  ^  ^  ^
  R* R* R* R* R* R* R* R* R* R    R    R    R  R  R  R  R  R  R  R  R  R  R  R  R* R  R  R
  T  T  T  T  T  T  T  T  T  T    T    T    T  T  T  T  T  T  T  T  T  T  T  T  T  T  T  T
  1  1  1  1  1  1  1  1  1  1    2    3    4  5  6  7  8  9  1  1  1  1  1  1  1  1  2  3
                                                            0  1  2  3  4  5  6
  * Restart RT Clock
```

Figure 5-36. Start Search Example 5

Figure 5-37 shows a large burst of noise near the beginning of the start bit that causes the start-bit search to be restarted. During RT1 following RT7, a search for a new start bit could not be started as the previous three RT samples are not all high. The receiver bit processor misses this start bit. The frame might be partially received or missed entirely, depending on the data in the frame and when the start-bit search logic synchronized upon what appeared to be a start bit. If a valid stop bit is not detected, an FE flag is set in SCSR.

|←——————— ACTUAL START BIT ———————→|←LSB→

```
———————————————————————\___/‾\_/‾_____/‾
1  1  1  1  1  1  1  1  1  0     0     1  0  1  0  0  0  0  0  0  0  0  0  0  0  0
^  ^  ^  ^  ^  ^  ^  ^  ^  ^  ^  ^  ^  ^  ^  ^  ^  ^  ^  ^  ^  ^  ^  ^  ^  ^  ^
R *R *R *R *R *R *R *R *R  R  R  R  R  R  R *R *R *R *R *R *R *R *R *R *R *R
T  T  T  T  T  T  T  T  T  T  T  T  T  T  T  T  T  T  T  T  T  T  T  T  T  T  T
1  1  1  1  1  1  1  1  1  1  2  3  4  5  6  7  1  1  1  1  1  1  1  1  1  1  1
```

* Restart RT Clock

Figure 5-37. Start Search Example 6

Figure 5-38 explores the case where the majority vote of RT8, RT9, and RT10 returns a logic-high level. However, the start bit is a special case that overrules the majority voting scheme. In review, at least three of the samples taken at RT1, RT3, RT5, and RT7 must be low. The start bit is detected and the RT clock is synchronized; because RT8–RT10 were not unanimous, the NF flag is set.

|←——————— PERCEIVED START BIT ———————→|
|←——————— ACTUAL START BIT ———————→|←LSB→

```
———————————————————————_____/‾\_/‾_____/‾
1  1  1  1  1  1  1  1  1  0     0     0           1  0  1
^  ^  ^  ^  ^  ^  ^  ^  ^  ^  ^  ^  ^  ^  ^  ^  ^  ^  ^  ^  ^  ^  ^  ^  ^  ^  ^
R *R *R *R *R *R *R *R *R  R  R  R  R  R  R  R  R  R  R  R  R  R  R  R *R  R  R
T  T  T  T  T  T  T  T  T  T  T  T  T  T  T  T  T  T  T  T  T  T  T  T  T  T  T
1  1  1  1  1  1  1  1  1  1  2  3  4  5  6  7  8  9  1  1  1  1  1  1  1  2  3
                                                    0  1  2  3  4  5  6
```

* Restart RT Clock

Figure 5-38. Start Search Example 7

5.6.5.2 Receiver Functional Operation

The receiver contains a receive serial shifter and a parallel RDR. While one character is in the process of being shifted in, another character may be held in RDR. This capability is called double buffering. The receive serial shifter cannot be accessed directly by the CPU. The input of the receive serial shifter is connected to the majority sampling logic of the receive bit processor.

The receiver is enabled when RE in SCCR1 is set to one. When RE is zero, the receiver is initialized and most of the receiver bit processor logic is disabled. The receiver bit processor logic drives a state machine (run by the RT clock) that determines the logic level for each bit time. This state machine controls when the bit processor logic is to sample the RXD pin and also controls when data is to be passed to the receive serial shifter. Data is shifted into the receive serial shifter according to the most recent synchronization of the RT clock with the incoming data stream. From this point on, the data is moved synchronously with the MCU system clock.

The first bit shifted in is the start bit, which is always a logic zero. The next eight bits shifted in are the basic data byte (LSB first). The next bit shifted in depends on the mode selected by M in SCCR1. If M = 1, then the bit is the ninth data bit and is placed in R8 of SCDR, concurrent with the transfer of data from the receive serial shifter to register RDR.

The last bit shifted in for each frame is the stop bit, which is always a logic one. If a logic zero is sensed during this bit time, the FE error flag in SCSR is set. A framing error is usually caused by mismatched baud rates between the receiver and transmitter or by a significant burst of noise. Note that a framing error is not always caught; the data in the expected stop bit time may be a logic one regardless.

When the stop bit is received, the frame is considered to be complete, and the received character in the receive serial shifter is transferred in parallel to RDR. If M = 1, the ninth bit is transferred at the same time; however, if the RDRF flag in SCSR is set, transfers are inhibited. Instead, the OR error flag is set, indicating to the user that the CPU needs to service register RDR faster. The data in RDR is preserved, but the data in the receive serial shifter is lost.

All status flags associated with a serially received frame are set simultaneously and at a time that does not interfere with CPU access to the affected registers. When a completed frame is received, either the RDRF or OR flag is always set. If RIE in SCCR1 is set, an interrupt results whenever RDRF is set. The receiver status flags NF, FE, and PF are set simultaneously with RDRF, as appropriate. These receiver flags are never set with OR because the flags only apply to the data in the receive serial shifter. The receiver status flags do not have separate interrupt enables, since they are set simultaneously with RDRF and must be read by the user at the same time as RDRF.

All receiver status flags are cleared by the following sequence. Register SCSR is read first, followed by a read of register SCDR. Reading SCSR not only informs the CPU of the status of the received data, but also arms the clearing

mechanism. Reading SCDR supplies the received data to the CPU and clears all of the status flags: RDRF, IDLE, OR, NF, FE, and PF.

5.6.5.2.1 Idle-Line Detect

The receiver hardware includes the ability to detect an idle line. This function can be used to indicate when a group of serial transmissions is finished. An idle line is defined as a minimum of 10 bit times (or 11 if a 9-bit data format is selected) of contiguous ones on the RXD pin. During a typical serial transmission, frames are transmitted isochronously, that is, no idle time occurs between frames. Even if all data bits in a frame are logic ones, the start bit ensures that at least one logic zero bit time occurs for each frame.

Motorola MCUs from the M68HC11 and M68HC05 Families have SCIs with only one type of idle-line detect circuitry. On these MCUs, the receiver bit processor starts counting logic one bit times at any point (even within a frame). This method allows the earliest recognition of an idle line because the stop bit and any contiguous ones preceding the stop bit are counted with the logic ones in the idle line following the stop bit. In some applications, the CPU overhead prevents the servicing of interrupts as soon as possible to ensure that no bit time of an idle line occurs between frames. Although this idle line causes no deterioration of the message content, if one bit time should occur after a data byte of all ones, the combination is seen as an idle line and causes sleeping SCIs to wake up.

The SCI on the QSM module contains this same idle-line detect logic called short idle-line detect as well long idle-line detect. In long idle-line detect mode, the SCI begins counting logic ones after the stop bit is received. The data content of a byte, therefore, does not affect how quickly the idle line is detected. When RXD goes idle for the minimum required time, the IDLE flag in SCSR is set. ILT in SCCR1 is used to choose between short and long idle-line detection.

If ILIE in SCCR1 is set, a hardware interrupt request is generated when the IDLE flag is set. This flag is cleared by reading SCSR with IDLE set, followed by reading register RDR. The IDLE flag is not set again until after at least one frame has been received (RDRF = 1), which prevents an extended idle interval from causing more than one interrupt.

5.6.5.2.2 Receiver Wakeup

The SCI receiver hardware provides a receiver wakeup function to support multinode networks containing more than one receiver. This function allows the transmitting device to direct a message to an individual receiver or group of receivers by sending an address frame at the start of a message. All receivers

not addressed for the current message invoke the receiver wakeup function, which effectively allows them to sleep through the rest of the message. Therefore, the CPU is alleviated from servicing register RDR, resulting in increased system performance.

The SCI receiver is placed in wakeup mode by writing a one to RWU in SCCR1. While RWU is set, all receiver status flag bits are inhibited from being set. Note that the IDLE flag cannot be used when RWU is set. Although the CPU can clear RWU by writing a zero to SCCR1, it is normally left alone by software and is cleared automatically by hardware in one of two methods: idle-line wakeup or address-mark wakeup.

WAKE in SCCR1 determines which method of wakeup is to be employed. If WAKE = 0, idle-line wakeup is selected. This method is compatible with the method originally used on the MC6801. If WAKE = 1, address-mark wakeup is selected, which uses a one in the MSB of data to denote an address frame and uses a zero to denote a normal data frame. Each method has its particular advantages and disadvantages.

Both wakeup methods require a software device addressing and recognition scheme and, therefore, can conform to all transmitters and receivers. The addressing information is usually the first frame(s) of the message. Receivers for which the message is not intended may set RWU and go back to sleep for the remainder of the message.

Idle-line wakeup allows a receiver to sleep until an idle line is detected, causing RWU to be cleared by the receiver and causing the receiver to wake up. The receiver waits through the idle times for the first frame of the next message. If the receiver is not the intended addressee, RWU may be set to put the receiver back to sleep. This method of receiver wakeup requires that a minimum of one frame of idle line be imposed between messages. As previously stated, no idle time is allowed between frames within a message.

Address-mark wakeup uses a special frame format to wake up the receiver. All frames consist of seven (or eight) data bits plus an MSB that indicates an address frame when set to a one. The first frame of each message should be an address frame. All receivers in the system must use a software scheme to determine which messages address them. If the message is not intended for a particular receiver, the CPU sets RWU so that the receiver goes back to sleep, thereby eliminating additional CPU overhead for servicing the rest of the message.

When the first frame of a new message is received with the MSB set, denoting an address frame, RWU is cleared. The byte is received normally, transferred to register RDR, and the RDRF flag is set. Address-mark wakeup allows messages to include idle times between frames and eliminates idle time between messages; however, an efficiency loss results from the extra bit time (address bit) that is required on all frames.

SECTION 6
STANDBY RAM (WITH TPU EMULATION)

The MC68332 contains 2K bytes of standby RAM memory. This section describes the operation and control of the RAM module.

6.1 Overview

The RAM module contains 2048 bytes of fully static complementary metal-oxide semiconductor (CMOS) RAM, powered by V_{DD} in normal operation. The entire array may be used as standby RAM if standby power is supplied to the V_{STBY} pin. Switching between V_{DD} and V_{STBY} occurs automatically.

The RAM may be used as general-purpose memory for the microcontroller unit (MCU), providing fast, two-clock accesses to the CPU. Typically, the RAM is used for program control stacks and frequently modified data variables. The CPU may read or write byte, word, or long-word data.

The RAM may also be used as microcode control memory for the time processor unit (TPU). The TPU must be placed in emulation mode to use the RAM in this manner. The CPU is not able to access the RAM in this mode.

6.2 Programmer's Model

The RAM module consists of two separately addressable sections — a 2K-byte RAM array and three registers for control and testing. The programmer's model is shown in Figure 6-1.

6.2.1 RAM Array Addressing

The RAM array can be placed anywhere in the address map of the MCU by means of the array base address register (RAMBAR), provided that it is placed on a 2K-byte boundary and does not overlap the RAM module control registers. RAMBAR can be written only once after reset. This prevents the RAM array from being accidentally remapped by software.

CONTROL REGISTERS

A23					A0	15	8	7	0	
Y	1111	1111	1011	1100	000X	MODULE CONFIGURATION REGISTER (RAMMCR)				$YFFB00
Y	1111	1111	1011	1100	001X	TEST REGISTER (RAMTST)				$YFFB02
Y	1111	1111	1011	1100	010X	BASE ADDRESS/STATUS REGISTER (RAMBAR)				$YFFB04
Y	1111	1111	1011	1100	011X	NOT IMPLEMENTED				
						•	•	•		
						•	•	•		
						•	•	•		
Y	1111	1111	1111	1100	111X	NOT IMPLEMENTED				

RAM ARRAY

A23	A11	A0	15	8	7	0
*——	—*000 0000	000X				
*——	—*000 0000	001X				
*——	—*000 0000	010X				
*——	—*000 0000	011X				
			•		•	•
			•		•	•
*——	—*000 0000	111X				

NOTE: All address locations are given in binary.
* — Base address defined in array base address register
X — Zero or one depending on byte/word address
Y — m111 where m is the modmap bit in the SIM MCR (Y = $7 or $F)

Figure 6-1. RAM Module Programmer's Model

The array may be placed in either supervisor space or in unrestricted space. In supervisor space, only the supervisor has access to the RAM. If a user attempts to access the RAM array when it has been placed in supervisor space, the access is ignored and the address may be decoded externally. In the unrestricted space, both the supervisor and user have RAM access.

The RAM responds to program and data space accesses. No checks of function code signals one or zero (FC1–FC0) are made during RAM array accesses. This permits the use of program counter relative addressing modes for operand fetches from the array, and allows program code to be executed from RAM.

6.2.2 RAM Module Register Block

There are three registers associated with the RAM: the RAM module configuration register (RAMMCR), the RAM test register (RAMTST), and a register that combines some status information with the array base address (RAMBAR). To protect these registers from accidental modification, they are always mapped to supervisor data space. When the RAM array is in use by the TPU for microcode emulation, these registers have no meaning or effect on the RAM array.

6.2.2.1 Register Block Addressing

Even though there are only three registers associated with the RAM module, logic design restrictions stipulate a 64-byte minimum register block size for the module. Unimplemented register addresses are read as zeros, and writes have no effect.

The actual base address of the RAM module register block is defined by three parameters. First, the most significant bit (MSB) of the base address (A23) is determined by the module mapping (MM) bit of the system integration module's (SIM's) configuration register. The MM bit defaults to one at reset, but may be written by software to a zero.

Address bits A22–A6 are determined by hardware for each module's register block. The last six address bits A5–A0 define the individual registers in the block. For the RAM module, the registers start at binary x111 1111 1111 1011 0000 0000. Depending on the MM bit, the address for the RAMMCR is either $7FFB00 or $FFFB00.

6.2.2.2 RAM Module Configuration Register (RAMMCR)

The RAMMCR (Figure 6-2) is used to determine in which space, supervisor or unrestricted, the 2K RAM array resides. It also selects whether the RAM is in STOP mode or normal mode. RAMMCR always resides in supervisor data space. Reads of unimplemented bits always return a zero. Writes do not affect unimplemented bits.

RAMMCR — RAM Module Configuration Register $YFFB00

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
STOP	NOT USED						RASP	NOT USED							
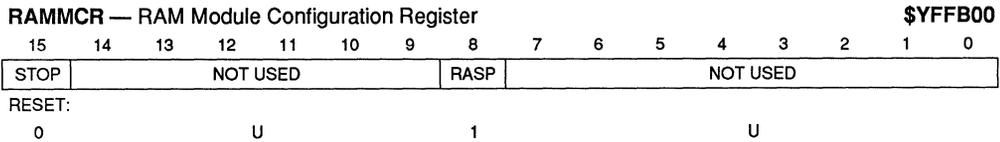
RESET:
| | | | | | | | | | | | | | | | |
| 0 | | | U | | | | 1 | | | | U | | | | |

Figure 6-2. RAM Module Configuration Register

STOP — Stop Control

This bit controls whether the 2K-byte RAM array is in stop mode or normal operation. The reset state is zero, which leaves the 2K-byte RAM array configured for normal operation. If placed in stop mode, the RAM array retains its contents, but cannot be read or written by the CPU. This bit may be read or written at any time when in the supervisor mode.

0 = 2K-byte RAM array operates normally.
1 = 2K-byte RAM array enters low power stop mode.

RASP — RAM Array Space

This bit controls the space placement of the 2K-byte RAM array. The reset or default state is one, placing the RAM in the supervisor space. It may be read or written at any time when in supervisor mode.

0 = 2K-byte RAM array is placed in unrestricted space.
1 = 2K-byte RAM array is placed in supervisor space.

6.2.2.3 RAM Test Register (RAMTST)

The RAMTST (Figure 6-3) provides a means of functionally testing the array. This register may be written only while the MCU is in test mode. Otherwise, reads of this register return zeros, and writes have no effect.

RAMTST — RAM Test Register $YFFB02

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
SDTEST							RTBA	NOT USED							

RESET:
| | | | | | | | | | | | | | | | |
| | | | 0 | | | | 0 | | | | 0 | | | | |
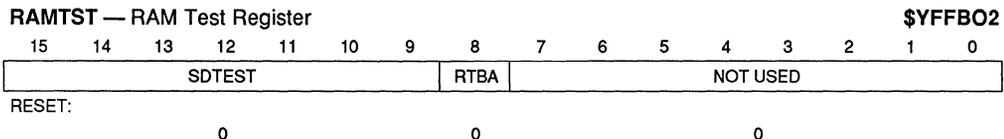
Figure 6-3. RAM Test Register

SDTEST — Soft Defect Test Function
SDTEST is used to select a particular RAM array test, when in test mode. Tests are initiated by writing a value to SDTEST, and the result is either output on data bit zero (D0) of the intermodule bus (IMB) or is determined by checking the array itself. Section **6.3.4 Test Mode Operation** explains the individual tests that can be run.

RTBA — Base Address Register R/$\overline{W}$
0 = RAMBAR may be written only once.
1 = RAMBAR may be written as desired.
RTBA controls the write-once lock on the array base address and status register (RAMBAR).

6.2.2.4 Base Address and Status Register (RAMBAR)
RAMBAR (Figure 6-4) is used to specify the 13 MSBs of the starting RAM array location (lowest address value) in the memory map. It also contains a status flag that shows whether the RAM is enabled or disabled. The unimplemented bits are read as zero and are unaffected by writes. RAMBAR is unaffected by a system reset.

RAMBAR can be written only once after a master reset. This prevents runaway software from accidentally re-mapping the array. Because the locking mechanism is activated by the first write after a master reset, the base address field should be written in a single word operation. Writing only one-half of the register prevents the other half from being written. Note that in test mode the locking mechanism for RAMBAR can be disabled by the RTBA bit in the RAMTST register.

RAMBAR — RAM Base Address and Status Register $YFFB04

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
A23	A22	A21	A20	A19	A18	A17	A16	A15	A14	A13	A12	A11	NOT USED		RAMDS

RESET.

| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |

Figure 6-4. RAM Base Address and Status Register

BIT 15–3 — RAM Array Base Address
These bits specify address lines A23–A11 of the base address of the RAM array when enabled. This allows the array to be placed on a 2K-byte boundary anywhere in the memory map, provided it does not overlap the RAM array

register block. If the register block is overlapped, the RAM array is disabled by internal logic and the RAMDS flag bit set to one until the next reset.

RAMDS — RAM Array Disabled
    0 = RAM array is enabled.
    1 = RAM array is disabled.
This bit indicates whether or not the RAM array is active or disabled. The RAM array is disabled by internal logic after a master reset. Writing a valid base address to the RAM array base address field (bits 15–3) automatically clears RAMDS, enabling the RAM array.

6.3 Operation

The RAM has six modes of operation: normal, standby, reset, test, stop, and TPU microcode emulation mode.

6.3.1 Normal Operation

The RAM module is in normal operation when powered by V_{DD}. The memory array may be read or written as byte, word, or long-word accesses. A byte or aligned word (high-order byte is at an even address) access only takes one bus cycle or two system clocks. A long word or misaligned word access requires two bus cycles or four system clocks.

6.3.2 Standby Operation

The contents of the static RAM array are maintained when the chip is powered down by supplying voltage to the V_{STBY} pin. The minimum voltage guaranteed to save the RAM contents is 3.0 V. Circuitry within the RAM module automatically switches between V_{DD} and V_{STBY}, whichever is higher, with no loss of data.

When the RAM is powered by V_{STBY} ($V_{STBY} > V_{DD}$), accesses to the RAM array are ignored. Reads may be inaccurate and writes may be corrupted if the voltage source is switching during the operation. Standby mode provides a guaranteed method of preserving RAM contents whenever V_{DD} is removed. If V_{STBY} operation is not desired, the V_{STBY} pin should be connected to V_{SS}.

6.3.3 Reset Operation

When a synchronous reset occurs, the CPU completes its current bus cycle before resetting. If a byte or word read or write was in process, this operation will be completed. If a long-word access was in progress, it is possible that only half of the operation will be completed, i.e., only one word of the long word might get written. When an asynchronous reset occurs, such as the loss of the clock, a

software watchdog timeout, or a master reset during a read or write, data read from or written to the RAM may be corrupted.

6.3.4 Test Mode Operation

The RAM module has special test functions that are accessible only in test mode. These tests are initiated by writing to the RAMTST register. The RAM array may be used normally in test mode, except when running tests; however, data left in the RAM during the test will be corrupted. Test mode provides a means for the user to test the RAM, even after the MCU has been installed in the system.

Soft defects are detected by running two tests: the open-circuit and array-current tests. The open-circuit tests check for most faults, whereas the array-current tests check for floating-gate defects not found by the open-circuit tests. To completely check for soft defects, it is recommended that all tests be run.

6.3.4.1 Open Tests

The open-circuit tests check each cell of the array for an open circuit. The tests check the left and right inverter circuits of the cell with both one and zero data input to the cell.

This test is divided into four parts. In each part, the entire array must be read and data checked for the proper value. Each location returns zeros for test 1 and test 3, and ones for test 2 and test 4. Any other value indicates an open-circuit condition within the cell.

Open-Circuit Test 1 — This checks the left half of each cell with an input value of zero.

    a. Write SDTEST to 0100111.
    b. Read the entire array, checking that each location returns all zeros (no open circuits).

Open-Circuit Test 2 — This checks the left half of each cell with an input value of one.

    a. Write SDTEST to 0110111.
    b. Read the entire array, checking that each location returns all ones (no open circuits).

Open-Circuit Test 3 — This checks the right half of each cell with an input value of zero.

    a. Write SDTEST to 0101111.
    b. Read the entire array, checking that each location returns all zeros (no open circuits).

Open-Circuit Test 4 — This checks the right half of each cell with an input value of one.

    a. Write SDTEST to 0111111.
    b. Read the entire array, checking that each location returns all ones (no open circuits).

Terminate the open-circuit test by setting SDTEST to 0000000.

6.3.4.2 Array-Current Test

The array-current test is invoked in a two-step procedure, beginning with a preconditioning test, or pretest, followed by the actual current test. The following steps describe the test procedure and result interpretation. The array-current test must be executed twice, once with the array initialized to all ones, and again with the array initialized with all zeros.

Array-Current Pretest — This pretest preconditions the RAM array for the actual array-current test. It must be completed immediately prior to performing the main array-current test.

    a. While in test mode, write SDTEST in the RAMTST register to 1100100 test mode.
    b. Read any address in the RAM array.
    c. Continuously check the least significant bit (LSB) of the data read (D0). If D0 = 0, preconditioning is still occurring. If D0 = 1, the pretest is complete.

Array-Current Test — This procedure must be started immediately after performing the pretest. The test determines whether the cell array current is higher or lower than an internal reference value.

    a. Write SDTEST to 1000100.
    b. Wait approximately 200 μs.
    c. Read any address of the RAM array.

d. Check the LSB (D0). If D0 = 0, the array-current test is successful. If D0=1, the array current is too high, indicating the existence of one or more soft defects (floating gates) in the array.

e. Write SDTEST to 0000000 to terminate the tests and to return the RAM array to normal operation.

6.3.5 Stop Operation

Writing the STOP bit of the RAMMCR causes the RAM module to enter stop mode. In this mode, the RAM array cannot be read or written. However, all data is retained. If V_{DD} falls below V_{STBY}, internal logic automatically switches to draw power from the V_{STBY} pin as in standby mode. In stop mode, the RAM array is disabled and does not respond to addresses within the array, allowing external logic to decode these addresses.

Stop mode differs from standby mode in that stop mode is entered under software control and is independent of the power source. Stop mode is exited by clearing the STOP bit of RAMMCR.

6.3.6 TPU Emulation Mode Operation

The RAM array may be used as the microcode control store for the TPU module. This mode of operation is selected from within the TPU. Refer to development support in the TPU manual for a complete description.

The TPU is connected to the RAM via a dedicated bus. While in emulation mode, the access timing of the RAM module matches the timing of the TPU microinstruction ROM to ensure accurate emulation. Normal accesses via the IMB are inhibited and the control registers have no effect, allowing external RAM to emulate the 2K RAM array at the same addresses.

6

6

SECTION 7
CPU32 OVERVIEW

This section is an overview of the CPU32. Refer to the CPU32 reference manual for a complete description of the capabilities and functions of this module.

The CPU32, the instruction processing module of the M68300 Family, is based on the industry-standard MC68000 core processor with many features of the MC68010 and MC68020 as well as unique features suited for high-performance controller applications. The CPU32 is designed to provide a significant increase in performance over the MC68HC11 CPU to meet the demand for higher performance requirements for the 1990's, while maintaining source code and binary code compatibility with the M68000 Family.

One major goal of the CPU32 is to increase system throughput. This increase could not be achieved by simply increasing the clock/bus frequency or by adding a few new instructions to an existing 8-bit, MC6800-type CPU. A faster, more powerful CPU, capable of processing data sizes up to 32 bits, has been included on the chip as a first step in realizing such a performance increase. As controller applications become more complex and control programs become larger, high-level languages (HLLs) will become the system designer's choice in programming languages. HLLs allow users to develop complex algorithms faster, with few errors, and they provide easier portability. The CPU32 has an instruction set based on the M68000 Family, which can efficiently support HLLs.

Ease of programming is an important consideration in using a microcontroller. An instruction format implementing a register-memory interaction philosophy predominates in the design, and all data resources are available to all operations requiring those resources. All eight multifunction data registers are available as data resources, and all seven general-purpose addressing registers are available for addressing data. Although the program counter (PC) and stack pointers (SP) are special-purpose registers, they are also available for most data addressing activities. The eight general-purpose data registers readily 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. Ease of program checking and diagnosis is further enhanced by trace and trap capabilities at the instruction level.

7

7.1 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 for Controller Applications
- Enhanced Addressing Modes
  - Scaled Index
  - Address Register Indirect with Base Displacement and Index
  - Expanded PC Relative Modes
  - 32-Bit Branch Displacements
- Instruction Set Enhancements
  - 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 Instruction
- Low Power Stop Instruction
- Hardware Breakpoint Signal, Background Mode
- 16.78-MHz Operating Frequency at −40 to 125°C
- Fully Static Implementation

7.2 Architecture Summary

The CPU32 architecture includes several important features that provide both power and versatility to the user. The CPU32 is source and object code compatible with the MC68000 and MC68010. All user-state programs can be executed unchanged. The major CPU32 features are as follows:

- 32-Bit Internal Data Path and Arithmetic Hardware
- 32-Bit Internal Address Bus — 24-Bit External Address Bus
- Rich Instruction Set
- Eight 32-Bit General-Purpose Data Registers
- Seven 32-Bit General-Purpose Address Registers
- Separate User and Supervisor Stack Pointers and Address Spaces
- Separate Program and Data Address Spaces
- Flexible Addressing Modes
- Full Interrupt Processing

7.2.1 Programmer's Model

The programming model of the CPU32 consists of two groups of registers: user model and supervisor model, which correspond to the user and supervisor privilege levels. Executing at the user privilege level, user programs can use only the registers of the user model. Executing at the supervisor level, system software uses the control registers of the supervisor level to perform supervisor functions.

As shown in the programming models (refer to Figures 7-1 and 7-2), the CPU32 has sixteen 32-bit general-purpose registers, a 32-bit program counter, one 32-bit supervisor stack pointer, a 16-bit status register, two alternate function code registers, and a 32-bit vector base register. The user programming model remains unchanged from previous M68000 Family microprocessors. The supervisor programming model, which supplements the user programming model, is used exclusively by the CPU32 system programmers who utilize the supervisor privilege level to implement sensitive operating system functions. The supervisor programming model contains all the controls to access and enable the special features of the CPU32. All application software, written to run at the nonprivileged user level, migrates without modification to the CPU32 from any M68000 platform.

7
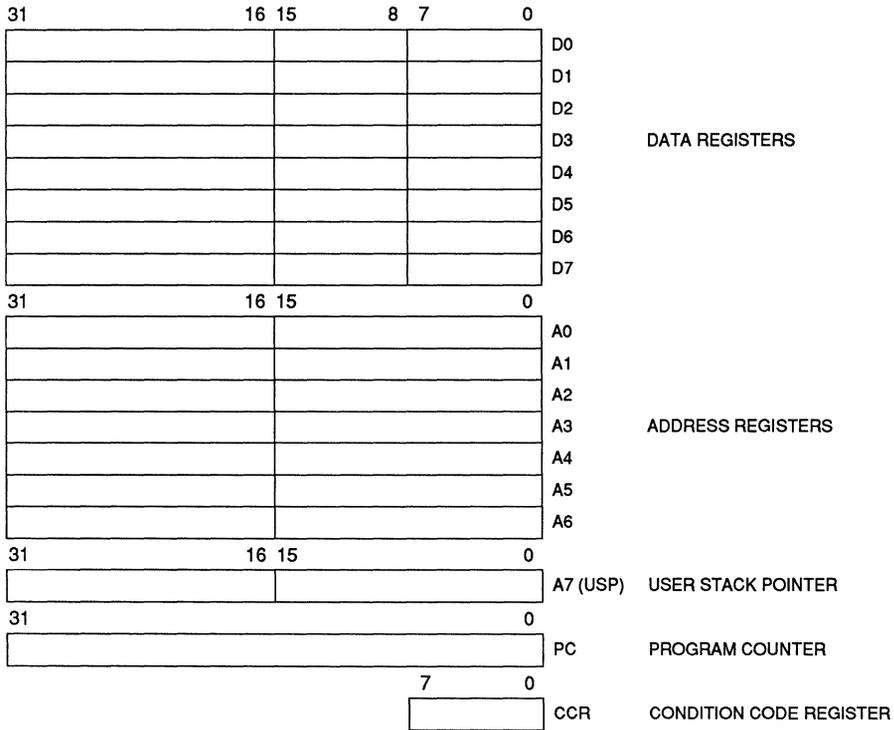
Figure 7-1. User Programming Model
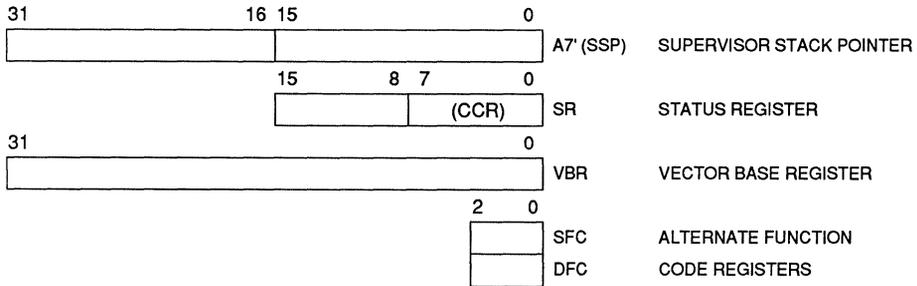
Figure 7-2. Supervisor Programming Model Supplement

7.2.2 Registers

Registers D7–D0 are used as data registers for bit (1–32 bits), byte (8-bit), word (16-bit), long-word (32-bit), and quad-word (64-bit) operations. Registers A6–A0 and the user and supervisor stack pointers are address registers that can be used as software stack pointers or base address registers. Register A7 is a register designation that applies to the user stack pointer in the user privilege level and to the supervisor stack pointer in the supervisor privilege level. In addition, the address registers may be used for word and long-word operations. All of the 16 general-purpose registers (D7–D0, A7–A0) can 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 status register (SR) (Figure 7-3) stores the processor status. It contains the condition codes that reflect the results of a previous operation and can be used for conditional instruction execution in a program. The condition codes are extend (X), negative (N), zero (Z), overflow (V), and carry (C). The user byte containing the condition codes is the only portion of the SR information available in the user privilege level; it is referenced as the condition code register (CCR) in user programs. In the supervisor privilege level, software can access the full status register, including the interrupt priority mask (three bits), as well as additional control bits. These bits put the processor in one of two trace modes (T1, T0) and in user or supervisor privilege level.

SR — Status Register

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
T1	T0	S	0	0	I2	I1	I0	0	0	0	X	N	Z	V	C

Figure 7-3. Status Register

System Byte

T1–T0 — Trace Enable

S — Supervisor/User State

Bits 12–11 — Unimplemented

I2–I0 — Interrupt Priority Mask

User Byte (Condition Code Register)

Bits 7–5 — Unimplemented

X — Extend

N — Negative

Z — Zero

V — Overflow

C — Carry

The vector base register (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 function code registers (SFC and DFC) contain 3-bit function codes. Function codes can be considered extensions of the 24-bit linear address that optionally provide as many as eight 16M-byte address spaces. Function codes are automatically generated by the processor to select address spaces for data and program at the user and supervisor privilege levels and to select a CPU address space used for processor functions (such as breakpoint and interrupt acknowledge cycles). Registers SFC and DFC are used by the MOVES instructions to specify explicitly the function codes of the memory address.

7.2.3 Data Types

Six basic data types are supported:

- Bits
- Packed Binary-Coded Decimal Digits
- Byte Integers (8 bits)
- Word Integers (16 bits)
- Long-Word Integers (32 bits)
- Quad-Word Integers (64 bits)

7.2.3.1 Organization in Registers

The eight data registers can store data operands of 1, 8, 16, 32, and 64 bits and addresses of 16 or 32 bits. The seven address registers and the two stack pointers are used for address operands of 16 or 32 bits. The PC is 32 bits wide.

7.2.3.1.1 Data Registers

Each data register is 32 bits wide. Byte operands occupy the low-order 8 bits, word operands, the low-order 16 bits, and long-word operands, the entire 32 bits. When a data register is used as either a source or destination operand, only the appropriate low-order byte or word (in byte or word operations, respectively) is used or changed; the remaining high-order portion is neither used nor changed. The least significant bit (LSB) of a long-word integer is addressed as bit zero, and the most significant bit (MSB) is addressed as bit 31. Figure 7-4 shows the organization of various types of data in the data registers.

BIT 0 (≤ MODULO (OFFSET) < 31, OFFSET OF 0 = MSB)

31	30		1	0
MSB				LSB

BYTE

31	24	23	16	15	8	7	0
HIGH-ORDER BYTE		MIDDLE HIGH BYTE		MIDDLE LOW BITE		LOW-ORDER BYTE	

16-BIT WORD

31	16	15	0
HIGH-ORDER WORD		LOW-ORDER WORD	

LONG WORD

31	0
LONG WORD	

QUAD WORD

63	62	32
MSB	ANY Dx	

31	1	0
ANY Dy		LSB
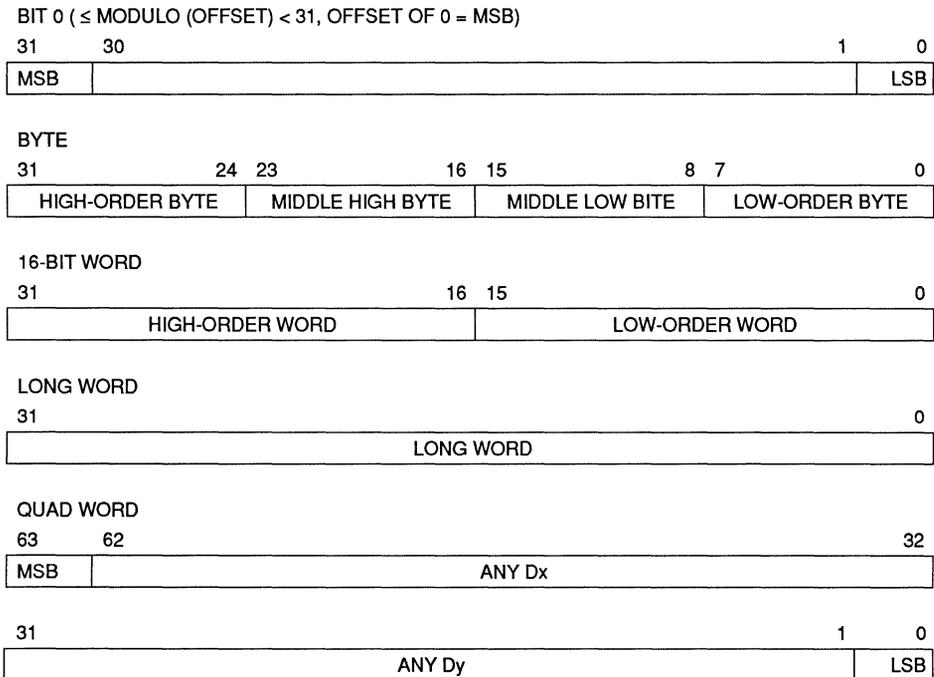
Figure 7-4. Data Organization in Data Registers

Quad-word data consists of two long words, used only by the product of 32-bit multiply or the dividend of 32-bit divide operations (signed and unsigned). Quad words may be organized in any two data registers without restrictions on order or pairing. There are no explicit instructions for the management of this data type, although the MOVEM instruction can be used to move a quad word into or out of the registers.

Binary-coded decimal (BCD) data represents decimal numbers in binary form. Although many BCD codes have been devised, the BCD instructions of the M68000 Family support formats in which the four LSBs consist of a binary number having the numeric value of the corresponding decimal number. In this BCD format, a byte contains two digits; the four LSBs contain the least significant digit, and the four MSBs contain the most significant digit. ABCD, SBCD, and NBCD operate on two BCD digits packed into a single byte.

7

7.2.3.1.2 Address Registers

Each address register and stack pointer is 32 bits wide and holds a 32-bit address. Address registers cannot be used for byte-sized operands. Therefore, when an address register is used as a source operand, either the low-order word or the entire long-word operand is used, depending upon the operation size. When an address register is used as the destination operand, the entire register is affected, regardless of the operation size. If the source operand is a word size, it is first sign-extended to 32 bits, and then used in the operation to an address register destination. Address registers are used primarily for addresses and to support address computation. The instruction set includes instructions that add to, subtract from, compare, and move the contents of address registers. Figure 7-5 shows the organization of addresses in address registers.

31	16	15	0
SIGN EXTENDED		16-BIT ADDRESS OPERAND	

31	0
FULL 32-BIT ADDRESS OPERAND	
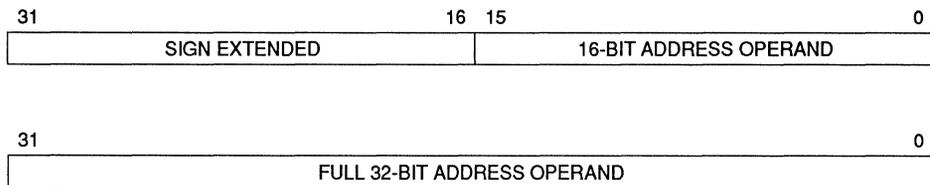
Figure 7-5. Address Organization in Address Registers

7.2.3.1.3 Control Registers

The control registers described in this section contain control information for supervisor functions and vary in size. With the exception of the user portion of the SR (the CCR), they are accessed only by instructions at the supervisor privilege level.

The SR shown in Figure 7-3 is 16 bits wide. Only 11 bits of the SR are defined; all undefined values are reserved by Motorola for future definition. The undefined bits are read as zeros and should be written as zeros for future compatibility. The lower byte of the SR is the CCR. Operations to the CCR can be performed at the supervisor or user privilege level. All operations to the SR and CCR are word-size operations, but for all CCR operations, the upper byte is read as all zeros and is ignored when written, regardless of privilege level.

The SFC and DFC are 32-bit registers with only bits 2-0 implemented that contain the address space values (FC2–FC0) for the read or write operand of the MOVES instruction. The MOVEC instruction is used to transfer values to and from the alternate function code registers. These are long-word transfers; the upper 29 bits are read as zeros and are ignored when written.

7.2.3.2 Organization in Memory

Memory is organized on a byte-addressable basis in which lower addresses correspond to higher order bytes. The address N of a long-word data item corresponds to the address of the most significant byte of the highest order word. The lower order word is located at address N + 2, leaving the least significant byte at address N + 3. The CPU32 requires long-word and word data as well as instruction words to be aligned on word boundaries (refer to Figure 7-6). Data misalignment is not supported.

7

BIT DATA
1 BYTE = 8 BITS

7	6	5	4	3	2	1	0

INTEGER DATA
1 BYTE = 8 BITS

15   8 7   0

MSB	BYTE 0	LSB	BYTE 1
	BYTE 2		BYTE 3

WORD = 16 BITS

15   0

MSB	WORD 0	LSB
	WORD 1	
	WORD 2	

LONG WORD = 32 BITS

15   0

MSB	·········· LONG WORD 0 ··········	HIGH ORDER	
		LOW ORDER	LSB
	·········· LONG WORD 1 ··········		
	·········· LONG WORD 2 ··········		

ADDRESS 1
ADDRESS = 32 BITS

15   0

MSB	·········· ADDRESS 0 ··········	HIGH ORDER	
		LOW ORDER	LSB
	·········· ADDRESS 1 ··········		
	·········· ADDRESS 2 ··········		

MSB = Most Significant Bit
LSB = Least Significant Bit

DECIMAL DATA
BCD DIGITS = 1 BYTE

15   12 11   8 7   4 3   0

BCD 0 MSD	BCD 1 LSD	BCD 2	BCD 3
BCD 4	BCD 5	BCD6	BCD 7

MSD = Most Significant Digit
LSD = Least Significant Digit

Figure 7-6. Memory Operand Addressing

7.3 System Features

The CPU32 includes a number of features that aid system implementation. These features include a privilege mechanism, separation of address spaces, multilevel priority interrupts, trap instructions, and a trace facility.

The privilege mechanism provides user and supervisor privilege states, privileged instructions, and external distinction of user and supervisor state references. The processor separates references between program and data space. This permits sharing of code segments that access separate data segments.

The CPU32 supports seven priority levels for 199 memory vectored interrupts. For each interrupt, the vector location can be provided externally or generated internally. The seventh level provides a nonmaskable interrupt capability.

To simplify system development, instructions are provided to check internal processor conditions and allow software traps. The trace facility allows instruction by instruction tracing of program execution without alteration of the program or special hardware. The following subsections describe the actions of the processor that are outside the normal processing associated with the execution of instructions.

7.3.1 Virtual Memory

The full addressing range of the CPU32 on the MC68332 is 16M byte in each of eight address spaces. Even though most systems implement a smaller physical memory, the system can be made to appear to have a full 16M byte of memory available to each user program by using virtual memory techniques.

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 suspended access is then restarted or continued.

The CPU32 uses instruction restart, which requires that only a small portion of the internal machine state be saved. After correcting the fault, the machine state is restored, and the instruction is fetched and started again. This process is completely transparent to the application program.

7.3.2 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 7-7 shows the required form of an instruction loop for the processor to enter loop mode.
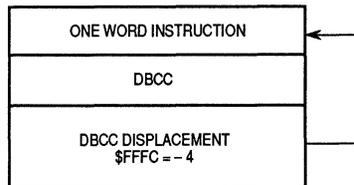
Figure 7-7. Loop Mode Instruction Sequence

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 on interrupts or other exceptions. Loopable instructions consist of all single word instructions that do not cause a change of flow.

7.3.3 Vector Base Register (VBR)

The VBR (Figure 7-8) contains the base address of the 1024-byte exception vector table, consisting 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 consists of one long word, except for the reset vector. The reset vector consists of two long words: the address used to initialize the supervisor stack pointer, and the address used to initialize the program counter.

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 four 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. Details of exception processing are provided in the section on exception processing in the CPU manual.

```
31                                                                              0
+-------------------------------------------------------------------------------+
|                                    VBR                                        |
+-------------------------------------------------------------------------------+
```
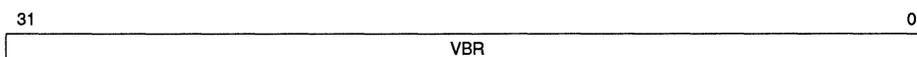
Figure 7-8. Vector Base Register

7.3.4 Exception Processing

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 is made, and the status register 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 status register and PC content 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 can be properly restored.

7.3.5 Processing States

The processor is always in one of four processing states: normal, exception, halted, or background. The normal processing state is associated with instruction execution; the bus is used to fetch instructions and operands and to store results. The exception processing state is associated with interrupts, trap

instructions, tracing, and other exception conditions. The exception may be internally generated explicitly by an instruction or by an unusual condition arising during the execution of an instruction. Externally, exception processing can be forced by an interrupt, a bus error, or a reset. The halted processing state is an indication of catastrophic hardware failure. For example, if during the exception processing of a bus error another bus error occurs, the processor assumes that the system is unusable and halts. The background processing state is initiated by breakpoints, execution of special instructions, or a double bus fault; it must be enabled by pulling $\overline{\text{BKPT}}$ low during $\overline{\text{RESET}}$. Background processing allows interactive debugging of the system via a simple serial interface.

7.3.6 Privilege States

The processor operates at one of two levels of privilege — user or supervisor. The supervisor level has higher privileges than the user level. Not all instructions are permitted to execute in the lower privileged user level, but all instructions are available at the supervisor level. This scheme allows a separation of supervisor and user levels, and so the supervisor can protect system resources from uncontrolled access. The processor uses the privilege level indicated by the S bit in the status register to select either the user or supervisor privilege level and either the USP or a SSP for stack operations.

7.3.7 Block Diagram

A block diagram of the CPU32 is shown in Figure 7-9. The major blocks depicted operate in a highly independent fashion that maximizes concurrency 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, managing the internal buses, registers, and functions of the execution unit.
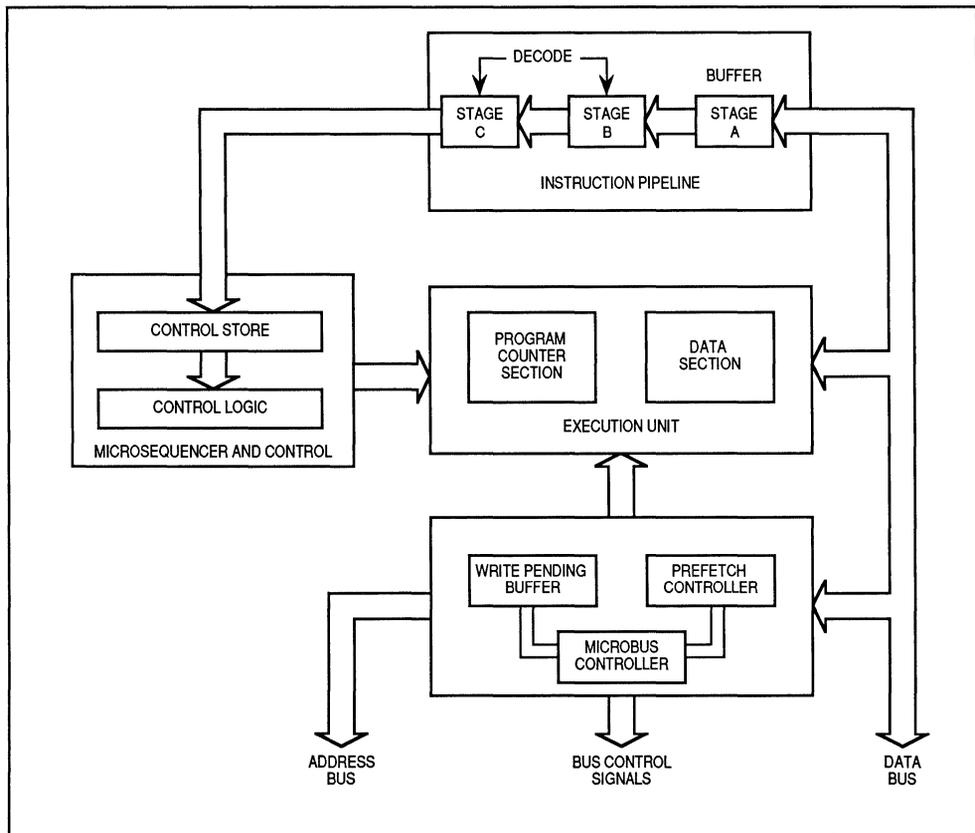
Figure 7-9. CPU32 Block Diagram

7.4 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 program counter relative mode also has index and offset capabilities. In addition to these addressing modes, many instructions implicitly specify the use of the status register, stack pointer, and/or program counter.

7.5 Instructions

The instruction set of the CPU32 is very similar to that of the MC68020 (refer to Table 7-1). Two new instructions have been added to facilitate controller applications — low power stop (LPSTOP) and table lookup and interpolate (TBLS, TBLSN, TBLU, TBLUN). The following MC68020 instructions are not implemented on the CPU32:

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

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.

7.5.1 M68000 Family Compatibility

It is the philosophy of the M68000 Family that all user-mode programs can execute unchanged on a more advanced processor, and 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. Refer to the CPU32 reference manual for a detailed comparison of the CPU32 and MC68020 instruction set.

Table 7-1. Instruction Set Summary

Mnemonic	Description	Mnemonic	Description
ABCD	Add Decimal with Extend	MOVE	Move
ADD	Add	MOVE CCR	Move Condition Code Register
ADDA	Add Address	MOVE SR	Move Status Register
ADDI	Add Immediate	MOVE USP	Move User Stack Pointer
ADDQ	Add Quick	MOVEA	Move Address
ADDX	Add with Extend	MOVEC	Move Control Register
AND	Logical AND	MOVEM	Move Multiple Registers
ANDI	Logical AND Immediate	MOVEP	Move Peripheral
ASL, ASR	Arithmetic Shift Left and Right	MOVEQ	Move Quick
		MOVES	Move Alternate Address Space
Bcc	Branch Conditionally	MULS, MULS.L	Signed Multiply
BCHG	Test Bit and Change	MULU, MULU.L	Unsigned Multiply
BCLR	Test Bit and Clear	NBCD	Negate Decimal with Extend
BGND	Background	NEG	Negate
BKPT	Breakpoint	NEGX	Negate with Extend
BRA	Branch	NOP	No Operation
BSET	Test Bit and Set	OR	Logical Inclusive OR
BSR	Branch to Subroutine	ORI	Logical Inclusive OR Immediate
BTST	Test Bit	PEA	Push Effective Address
CHK, CHK2	Check Register Against Upper and Lower Bounds	RESET	Reset External Devices
CLR	Clear	ROL, ROR	Rotate Left and Right
CMP	Compare	ROXL, ROXR	Rotate with Extend Left and Right
CMPA	Compare Address	RTD	Return and De-allocate
CMPI	Compare Immediate	RTE	Return from Exception
CMPM	Compare Memory to Memory	RTR	Return and Restore Codes
CMP2	Compare Register Against Upper and Lower Bounds	RTS	Return from Subroutine
DBcc	Test Condition, Decrement and Branch	SBCD	Subtract Decimal with Extend
DIVS, DIVSL	Signed Divide	Scc	Set Conditionally
DIVU, DIVUL	Unsigned Divide	STOP	Stop
EOR	Logical Exclusive OR	SUB	Subtract
EORI	Logical Exclusive OR Immediate	SUBA	Subtract Address
EXG	Exchange Registers	SUBI	Subtract Immediate
EXT, EXTB	Sign Extend	SUBQ	Subtract Quick
LEA	Load Effective Address	SUBX	Subtract with Extend
LINK	Link and Allocate	SWAP	Swap Register Words
LPSTOP	Low Power Stop	TBLS,TBLSN	Table Lookup and Interpolate (Signed)
LSL, LSR	Logical Shift Left and Right	TBLU, TBLUN	Table Lookup and Interpolate (Unsigned)
ILLEGAL	Take Illegal Instruction Trap	TAS	Test Operand and Set
JMP	Jump	TRAP	Trap
JSR	Jump to Subroutine	TRAPcc	Trap Conditionally
		TRAPV	Trap on Overflow
		TST	Test Operand
		UNLK	Unlink

7

7.5.2 New Instructions

Two new instructions have been added to the MC68000 instruction set for use in controller applications. They are low power stop (LPSTOP) and table lookup and interpolate (TBL).

7.5.2.1 Low Power Stop (LPSTOP)

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

7.5.2.2 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 each data point could require an inordinate amount of memory. The table instruction requires only a sample of data points stored in the array, reducing memory requirements. Intermediate values are recovered with this instruction via linear interpolation. The results are optionally rounded with the round-to-nearest algorithm.

7.6 Development Support

The following features have been implemented on the CPU32 to enhance the instrumentation and development environment:

- M68000 Family Development Support
- Background Debug Mode
- Deterministic Opcode Tracking
- Hardware Breakpoints

7.6.1 M68000 Family Development Support

All M68000 Family members include features to facilitate applications development. These features include the following:

Trace on Instruction Execution — M68000 Family processors include an instruction by instruction tracing facility as an aid to program development. The MC68020, MC68030, MC68040, and CPU32 also allow tracing only those instructions causing a change in program flow. In the trace mode, a trace exception is generated after an instruction is

executed, allowing a debugger program to monitor the execution of a program under test.

Breakpoint Instruction — An emulator may insert software breakpoints into the target code to indicate when a breakpoint has occurred. On the MC68010, MC68020, MC68030, and CPU32, this function is provided via illegal instructions, $4848–$484F, to serve as breakpoint instructions.

Unimplemented Instruction Emulation — During instruction execution, 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.

7.6.2 Background Debug Mode

Microcomputer systems generally provide a debugger, implemented in software, for system analysis at the lowest level. The background debug mode on the CPU32 is unique in that the debugger has been implemented in CPU microcode. Registers can be viewed and/or altered, memory can be read or written to, and test features can be invoked. Incorporating these capabilities on-chip simplifies the environment in which the in-circuit emulator operates. With an integrated debugger, the traditional emulator configuration, as shown in Figure 7-10, may be replaced by a bus state analyzer (BSA), shown in Figure 7-11. The BSA simply monitors the traffic on the internal bus, whereas the in- circuit emulator replaces the processor in the target system with hardware in the emulator, and all external bus traffic must traverse the cable between the emulator and the target system.
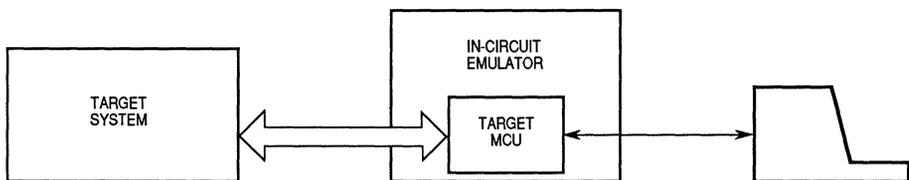
Figure 7-10. Traditional In-Circuit Emulator Diagram
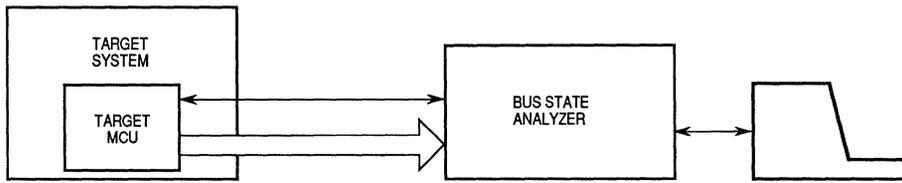
Figure 7-11. Bus State Analyzer Configuration

As each command is accumulated in the serial shifter, a microaddress is generated that points to the microcode routine corresponding to that command. If the command can complete without additional serial traffic, it does; however, if addresses or operands are required, the microcode waits as each word is assembled. Result operands are loaded into the output shift register to be shifted out as the next command is read. Table 7-2 summarizes the command set available in the background debug mode.

Table 7-2. Background Mode Command Summary

Command	Mnemonic	Description
Read D/A Register	RDREG/RAREG	Read the selected address or data register and return the results via the serial interface.
Write D/A Register	WDREG/WAREG	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 background mode.
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 source function code register (SFC) determines the address space accessed.
Write Memory Location	WRITE	Write the operand data to the memory location specified by the long-word address. The destination function code (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 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 supply the first operand. Subsequent operands are written with the FILL command.
Resume Execution	GO	The pipe is flushed and re-filled before resuming instruction execution at the current PC.
Patch User Code	CALL	Current program counter is stacked at the location of the current stack pointer. 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.

7

7.6.3 Deterministic Opcode Tracking

CPU32 function code outputs are augmented by two supplementary signals to monitor the instruction pipeline. The instruction pipe (IPIPE) output indicates the start of each new instruction and each midinstruction pipeline advance. The instruction fetch (IFETCH) output identifies the bus cycles in which the operand is loaded into the instruction pipeline. Pipeline flushes are also signalled with IFETCH. Monitoring these two signals allows a bus analyzer to synchronize itself to the instruction stream and monitor its activity.

7.6.4 On-Chip Breakpoint Hardware

An external breakpoint input and on-chip breakpoint hardware allow a breakpoint trap on any memory access. Off-chip address comparators preclude breakpoints unless show cycles are enabled. Breakpoints on instruction prefetches that are ultimately flushed from the instruction pipeline are not acknowledged; operand breakpoints are always acknowledged. Acknowledged breakpoints initiate exception processing at the address in exception vector number 12, or alternately enter background mode.

SECTION 8
TIME PROCESSOR UNIT OVERVIEW

The time processor unit (TPU) performs both simple and complex timing tasks, making it the latest advance in timer systems. Viewed as a special-purpose microcomputer, this processor performs two operations, match and capture, on one operand: TIME. Every occurrence of either action is called an event. The servicing of these events by the TPU replaces the servicing of interrupts by the host central processing unit (CPU). The timing functions currently synthesized are the following:

- Discrete Input/Output
- Input Capture/Input Transition Counter
- Output Compare
- Pulse-Width Modulation
- Synchronized Pulse-Width Modulation
- Period Measurement with Additional Transition Detect
- Period Measurement with Missing Transition Detect
- Position-Synchronized Pulse Generator
- Stepper Motor
- Period/Pulse-Width Accumulator

The advanced TPU affords for the first time high-resolution timing and multiple time function capability (flexibility) in the timer system pins.

8.1 High-Resolution Timing

High-resolution timing is limited by CPU overhead required for servicing timing tasks such as period measurement, pulse measurement, pulse-width modulated waveform generation, etc. On the TPU, high-resolution timing is achieved by two main capabilities: a) reduced latency, and b) reduced service time, which free the CPU to focus on other responsibilities. The TPU provides a higher resolution than the CPU could achieve, and creates no CPU overhead for servicing timing tasks.

8.1.1 Latency

Latency is the interval of time from an event to the start of event servicing. The ability of the TPU to service its own interrupts or events reduces latency, and the

CPU is not required to service each input transition that occurs on a pin, or to determine each match time required for waveform synthesis. Once configured by the host CPU, the self-contained TPU performs complex time functions requiring high resolution with little or no CPU intervention.

8.1.2 Service Time

Service time is the time expended servicing an event. In older microcontroller unit (MCU) timer functions, service time is constrained because the MCU instruction set is not optimized for time function synthesis. The TPU instruction set is optimized, and time functions are synthesized with fewer instructions than the CPU. Instructions execute faster and service time is reduced. Instructions executed by the TPU are not user software, but firmware, special-purpose microcode written by Motorola to perform a set of time functions. Microcode is placed into the control store (ROM) when the device is manufactured.

8.2 Flexibility

The TPU has the flexibility to be configured to solve the user's timer requirements directly. This flexibility is attained through five capabilities:

- Channel Orthogonality
- Interchannel Communication
- Programmable Channel Service Priority
- Selection of Timing Functions
- Emulation Capability

8.2.1 Channel Orthogonality

Traditionally, timer systems have been limited by the specific functions of channel pins dedicated to perform time functions such as input capture, output compare, or pulse accumulation. All channels of the TPU contain identical hardware and are functionally equivalent in operation, such that any channel can be configured to perform any time function. The user controls the combination of time functions; the only constraint is the number of pins available for timing functions.

8.2.2 Interchannel Communication

The TPU's ability to service itself requires a continuous flow of direct and indirect communication. Direct communication is accomplished through a change channel feature in which any channel of the TPU can operate on another channel to affect its state. Indirect communication is provided by a link feature in which any channel can link to one or more channels, including itself, to signal a need for future service. As a result, the user can reference the operation of one channel to the occurrence of a specific action on another channel. Use of either or both

of these features is contained within the firmware and occurs when the time function executes on a given channel. Both types of interchannel communication can use TPU parameter RAM to transfer data between channels.

8.2.3 Programmable Channel Service Priority

Applications may require different priorities of event service. The channel service priority may be programmed to one of three levels: high, middle, and low. The scheduler allows calculation of worst-case latency for event servicing and ensures servicing of all channels by preventing permanent blockage.

8.2.4 Selection of Timing Functions

In addition to functions previously mentioned, the TPU contains other controller functions, which can be programmed to operate on any channel. Parameter registers associated with each channel are used as general-purpose time operands. A description of each time function follows in the section.

8.2.5 Emulation Capability

The TPU cannot resolve all timer problems using predefined time functions alone; therefore, development of user-defined time functions is allowed in emulation mode. Using the RAM module of the MCU as a writable control store provides TPU emulation. In TPU emulation mode, an auxiliary bus connection is made between the RAM module and the TPU module, and access to the RAM module via the intermodule bus is disabled. A 9-bit address bus, a 32-bit data bus, and control lines transfer information between the modules. To ensure exact emulation, the access timing of the RAM module remains consistent with the TPU ROM control store.

8.3 Features

- 16 channels; each channel is associated with a pin.
- Each channel can perform any time function.
- Each time function may be assigned to more than one channel at a given time.
- Each channel has an event register comprised of the following:
  — 16-Bit Capture Register
  — 16-Bit Compare/Match Register
  — 16-Bit Greater-Than or Equal-To Comparator
- Each channel can be synchronized to one or both of the two 16-bit free-running timer count registers (TCR1 and TCR2).
- TCR1 is clocked from the output of a prescaler. The prescaler's input is the internal TPU system clock divided by either 4 or 32. The four settings of the

8

prescaler are divide by 1, 2, 4, and 8. Channels using TCR1 have the capability to resolve down to the TPU system clock divided by 4.

- TCR2 is clocked from the output of a prescaler. The prescaler's input is the external T2CLK pin. The four settings of the prescaler are divide by 1, 2, 4, and 8. Channels using TCR2 have the capability to resolve down to the TPU system clock divided by 8.
- TCR2 may be used as a hardware pulse accumulator clocked from the external TCR2 pin, or as a gated pulse accumulator of the clock that increments TCR1.
- All channels have at least six 16-bit parameter registers. Channels 14 and 15 each have eight 16-bit parameter registers. All parameter registers are contained in a dual-access RAM, accessible from both the TPU and CPU.
- A scheduler with three priority levels segregates high-, middle-, and low-priority time functions. Any channel may be assigned to one of these three priority levels.
- All time functions are microcoded.
- Emulation and development support are provided for all time function features such as breakpoint, freeze, and single step, giving internal register accessibility.
- Coherent transfer capability for two parameters is provided in hardware.
- Coherent transfer capability for N parameters may be performed as a TPU microcode function. (Refer to development support in the TPU reference manual for further details on this feature.)

8.4 Block Diagram

The TPU is a single module of a microcontroller integrated circuit, and communicates with the rest of the MCU via an intermodule bus (IMB). The TPU is a microcoded controller and consists of five main units, as shown in Figure 8-1: the host interface, 16 timer channels, a scheduler, a microengine, and the execution unit. (Refer to development support in the TPU reference manual for further details of the internal architecture.)
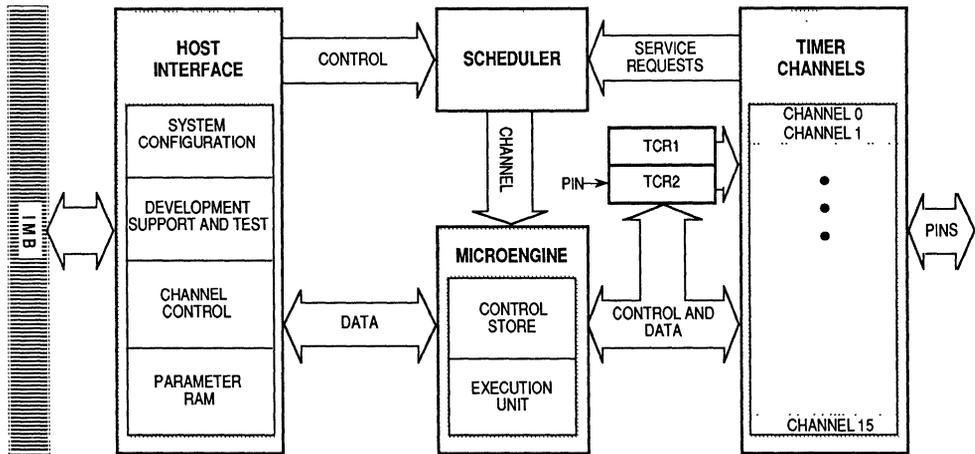
Figure 8-1. TPU Simplified Block Diagram

8.4.1 Host Interface

The host interface registers allow host (CPU or another bus master) communication to the TPU both before and during the execution of a time function. The host interface registers are those accessible from the IMB through the TPU bus interface unit (BIU):

- System Configuration Registers
- Channel Control and Status Registers
- Development Support and Test Verification Registers
- Channel Parameter Registers (RAM)

The parameter registers constitute a shared work space for communication between the bus master and the TPU. The bus master (CPU) specifies to the TPU attributes about the time function being executed on a channel by writing the associated parameter registers for that channel. This data is read by the TPU at the appropriate time to affect channel operation. Likewise, the TPU sends the bus master data by storing the information in the parameter registers to be read by the bus master. (Refer to time functions in the TPU reference manual for detailed information contained in the parameter registers.) The TPU has no capability to access address space outside the TPU module; consequently, the host CPU must provide all such access.

In addition to their normal use as time function operands, channel parameter registers may also contain channel control fields. A channel parameter register

used to configure channel attributes contains three fields: pin state control (PSC), pin action control (PAC), and time base/directionality control (TBS). These fields perform the following functions:

PSC — Forces the output level of the pin.

PAC — On input channels, specifies the transition edge to be detected. On output channels, specifies the logic level to be output to the pin due to a match: a greater-than or equal-to comparison.

TBS — Specifies both the channel direction (input or output) and the time base (TCR1 or TCR2) associated with the input capture and output compare function of each channel.

8.4.2 Timer Channels

All 16 orthogonal TPU channels can perform any of the time functions, because identical hardware exists in all channels. Each channel has a dedicated pin with control hardware for that pin. The channel control hardware responds to an event. This response can be either a) to output a specific level to the pin upon recognition of a match event, or b) to initiate an input capture event upon detection of the specified input transition received on the pin. Either event prompts the channel to issue a service request to the scheduler.

A TPU channel is capable of the following:

- Input Capture — Transfers the content of either of the two TCRs to a capture register whenever an input transition occurs.
- Output Compare — Performs a greater-than or equal-to comparison of the content of the compare/match register with either of the two TCRs. A match event occurs when the comparison is greater-than or equal-to the content value. Either TCR may be captured in the capture register when the event occurs.

The latter feature aids the user when using TCR2 to track a physical device when a match event occurs. The time function may be written to capture from TCR1 the time of the event occurrence or, alternately, to capture the current value of TCR2 corresponding to the physical position when a match with TCR1 occurs at a predetermined time.

8.4.3 Scheduler

Once a service request is received, the scheduler designates the channel for service by the microengine. A service request has four origins: match detect,

input transition detect, link request, or host request. The scheduler provides three priority levels to which a channel may be assigned: high, middle, and low. A channel is granted service based on the assigned priority level, implemented by a priority-scheduling mechanism.

8.4.4 Microengine

The microengine services a channel by executing microinstructions during the service time for each service request from the scheduler. Instead of the CPU, the microengine directs operations of the execution unit and the timer channels in the synthesis of the time functions.

The control store houses special-purpose microcode, which is executed for each phase of a time function. The predefined timing functions described in this manual are contained in an unalterable ROM. For emulation or for development of new timing functions, microcode may be executed from the MCU RAM.

8.4.5 Execution Unit

The execution unit moves data and computes information. It is composed of registers, functional units, and data paths which, along with the microengine, evaluate and control the resources associated with a channel to synthesize time functions.

8.5 General Concept

The TPU is an intelligent, semi-autonomous microcontroller dedicated to timing control. Its intelligence enables the servicing of timing events without CPU intervention. This device uses a private microengine for a processor, a scheduler, input/output channels, ROM instructions, and shared-access data RAM to operate independently and simultaneously with the CPU. Consequently, the setup and service time for each timer event is minimized.

A time-of-day approach is used where all time functions are related to one of two 16-bit free-running TCRs. Time functions are synthesized by combining the two time primitives, match and capture events. By performing these time primitives in hardware, the TPU can precisely determine the time when a match event is to occur and then specify the state of the output pin accordingly. The TPU can also accurately record the time at which an input transition occurs and can perform calculations based on the time of the occurrence. An event register for each channel provides for simultaneity of match/capture-event occurrences on all channels.

When a match or input-capture event requiring service occurs on a channel, the channel generates a service request to the scheduler. The scheduler prioritizes the request with other pending service requests. When the microengine is idle, the scheduler causes the microengine to execute a microcode sequence. When the microengine is busy, the new sequence begins when the code being executed ends. The microengine performs the function, which is defined by the content of the control store, using parameters from the parameter RAM and from the event registers, etc., as needed. Following is an example.

Channel X is generating a periodic waveform and presently the output is high. When the value of the TCR used by that channel increments to match the value of the event register of channel X, a match event occurs. The event switches the output to low and generates a new service request to the scheduler. The scheduler then schedules and initiates service of channel X by the microengine.

When execution of the sequence begins, the microengine uses the execution unit a) to obtain (from the parameter RAM) the value representing the duration of counts for which channel X should remain low, and b) to add to this value the value from the content of the event register of channel X. The content of the event register is then replaced by this sum, the channel control is set for a match event on the same TCR, and the pin control is set to cause the output pin for channel X to switch high when the event occurs. A channel interrupt, which signals the end of service to the CPU, may be asserted (if the time function provides for it and the interrupt is enabled). The microengine is then free to service the next event determined by the scheduler.

8.6 User's Model/Memory Map

The TPU memory map is made up of four functional types of memory:

System Configuration Registers
Channel Control and Status Registers
Development Support and Test Verification Registers
Channel Parameter Registers (RAM)

All registers are 16 bits in length and are accessible through word or long-word transfers with one exception: the interrupt status register can also be accessed on a byte basis. This exception allows the host CPU to perform bit manipulation instructions on the register.

The user's model is comprised of registers grouped into one of two types of data space. Grouping is determined by the amount of system protection needed and is categorized as follows:

Supervisor Data Space (More Protection)
User Data Space (Less Protection)

Registers requiring more protection fall into the supervisor data space group and can be accessed only when the CPU is in supervisor mode. These registers configure the overall TPU and the interface to the rest of the MCU. (CPU status bit S = 1) These registers are indicated by an "S" before the address in Figure 8-2.

8

	ADDRESS	15 BYTE n 8 | 7 BYTE n + 1 0
S	$YFFE00	MODULE CONFIGURATION REGISTER
S	$YFFE02	CONFIGURATION REGISTER
S	$YFFE04	DEVELOPMENT SUPPORT CONTROL REGISTER
S	$YFFE06	DEVELOPMENT SUPPORT STATUS REGISTER
S	$YFFE08	INTERRUPT CONFIGURATION REGISTER
S	$YFFE0A	INTERRUPT ENABLE REGISTER
S	$YFFE0C	CHANNEL FUNCTION SELECT REGISTER 0
S	$YFFE0E	CHANNEL FUNCTION SELECT REGISTER 1
S	$YFFE10	CHANNEL FUNCTION SELECT REGISTER 2
S	$YFFE12	CHANNEL FUNCTION SELECT REGISTER 3
X	$YFFE14	HOST SEQUENCE REGISTER 0
X	$YFFE16	HOST SEQUENCE REGISTER 1
X	$YFFE18	HOST SERVICE REQUEST REGISTER 0
X	$YFFE1A	HOST SERVICE REQUEST REGISTER 1
S	$YFFE1C	CHANNEL PRIORITY REGISTER 0
S	$YFFE1E	CHANNEL PRIORITY REGISTER 1
S	$YFFE20	INTERRUPT STATUS REGISTER
S	$YFFE22	LINK REGISTER
S	$YFFE24	SERVICE GRANT LATCH REGISTER
S	$YFFE26	DECODED CHANNEL NUMBER REGISTER

S = Supervisor accessible only
X = Assignable as supervisor accessible only (if SUPV = 1) or
    unrestricted (if SUPV = 0). Unrestricted allows both user and
    supervisor access.
Y = m111, where m is the state of the modmap bit in the module
    configuration register of the system integration module (Y = $7 or
    $F).

Figure 8-2. TPU Register Map

The assignable space can be either restricted to supervisor-only access or unrestricted to both user and supervisor accesses. The assignable addresses are indicated by an "X" before the address in Figures 8-2 and 8-3. The supervisor bit (SUPV) in the TPU module configuration register (MCR) designates the assignable data space as either supervisor or unrestricted. If SUPV is set, the space is designated as supervisor only. Access is then permitted only when the CPU is operating in supervisor mode. Attempts to access the registers with user software (user data space) cause the bus cycle to be transferred externally. Results can then be unpredictable depending on the hardware environment. If SUPV is clear, then both user and supervisor accesses are permitted. Attempting to access a supervisor-only register, i.e., ones denoted with an "S" from user software, cause the TPU to respond as if an

unimplemented register had been accessed. Writes have no effect and reads return zeros. (Refer to processing states in the CPU32 reference manual for more information on supervisor mode.)

The address space of the TPU memory map occupies 512 bytes. Unused registers within the 512-byte address space return zeros when read. The memory map registers are shown in Figures 8-2 and 8-3.

Channel	Parameter							
	0	1	2	3	4	5	6	7
0	X $YFFF00	02	04	06	08	0A	—	—
1	X $YFFF10	12	14	16	18	1A	—	—
2	X $YFFF20	22	24	26	28	2A	—	—
3	X $YFFF30	32	34	36	38	3A	—	—
4	X $YFFF40	42	44	46	48	4A	—	—
5	X $YFFF50	52	54	56	58	5A	—	—
6	X $YFFF60	62	64	66	68	6A	—	—
7	X $YFFF70	72	74	76	78	7A	—	—
8	X $YFFF80	82	84	86	88	8A	—	—
9	X $YFFF90	92	94	96	98	9A	—	—
10	X $YFFFA0	A2	A4	A6	A8	AA	—	—
11	X $YFFFB0	B2	B4	B6	B8	BA	—	—
12	X $YFFFC0	C2	C4	C6	C8	CA	—	—
13	X $YFFFD0	D2	D4	D6	D8	DA	—	—
14	X $YFFFE0	E2	E4	E6	E8	EA	EC	EE
15	X $YFFFF0	F2	F4	F6	F8	FA	FC	FE

— Not Implemented
X = Assignable as supervisor accessible only (if SUPV = 1) or unrestricted (if SUPV = 0). Unrestricted allows both user and supervisor access.
Y = m111, where m is the modmap bit in the module configuration register of the system integration module (Y = $7 or $F).

Figure 8-3. TPU Parameter RAM Map

8.7 Coherency

Coherency is the reading or writing of data identical in age or logically related. Specifically, when written coherently, all data must be written before any portion can be accessed. When read coherently, all data must be updated before any portion may be read. A condition that would allow data to be read, while some but not all has been updated, is prohibited by the TPU design.

Predefined time functions require two parameters to be written coherently for proper operation. For example, the pulse width modulation (PWM) function requires the two coherent parameters: period and high time. User-defined time functions may require coherency of more than two parameters. A time function that enables the coherent reading/writing of three or four parameters is described in development support in the TPU reference manual.

The RAM arbitration scheme provides for the coherent reading/writing of two adjacent parameter registers, with respect to logical address. The TPU can also access two parameters coherently via microcode (refer to RAM accesses in the TPU reference manual).

8.8 Getting Started — TPU Configuration Summary

After initial power-on reset, the TPU remains in an idle state, requiring initialization of several registers before any time function can begin execution.

A general sequence guide and a visual representation of the attributes needed for the TPU module are given in Figure 8-4. Further details for each time function can be found in the time function section of the TPU reference manual.

- The module configuration register must be initialized to properly configure the following:
  — Prescalers for both TCR1 and TCR2
  — TCR2 selection of either clock or gate function
  — Interrupt arbitration identification number for the entire TPU module
  — User/supervisor bit
- The interrupt configuration register is written to choose the base vector number and interrupt level for the TPU module.
- The channel interrupt enable register must be written if interrupts are to be enabled from the appropriate channels.
- The channel function select registers are written to choose the time function to be performed by each channel.
- The host sequence registers are written to choose the variations possible within the time function flow.
- Parameter 0 (and any other pertinent parameters) for each configured channel must be written.
- The host service request registers are written to initialize the active channels.
- The channel priority is usually written last to enable each channel by assigning it a high, middle, or low priority.
- The host service request registers (or a channel interrupt) should be monitored for completion of initialization.
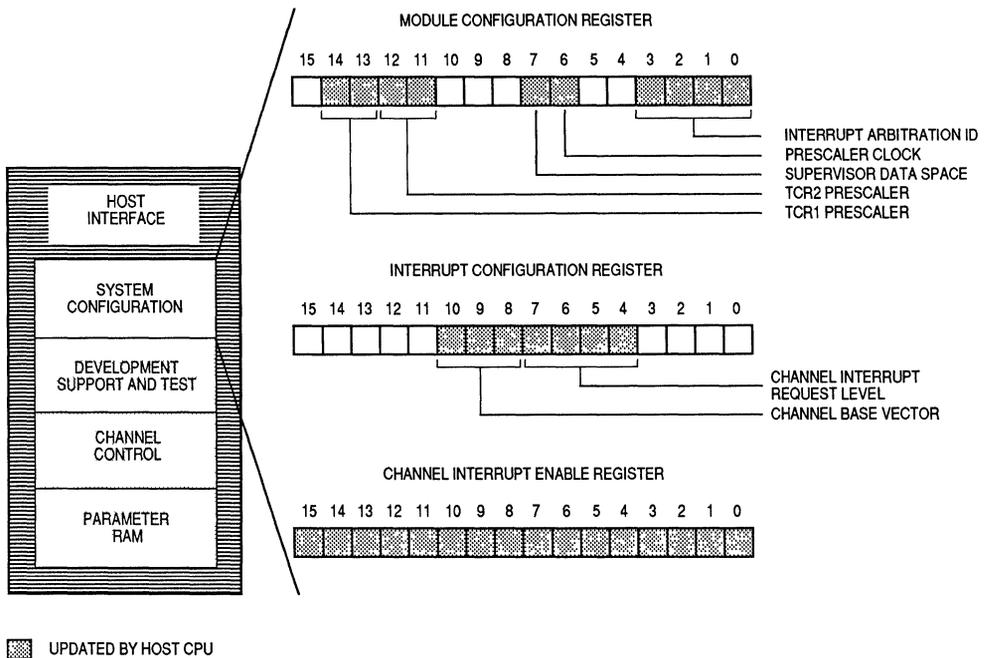
Figure 8-4. TPU Module Configuration

8.9 Time Functions Implemented

The following sections describe the time functions implemented in the TPU microcode ROM.

8.9.1 Discrete Input/Output (DIO)

When a pin is used as a discrete input, a parameter indicates the current input level and the previous 15 levels of a pin. Bit 15, the most significant bit of the parameter, indicates the most recent state. Bit 14 indicates the next most recent state. This sequence continues for each of the 16 bit positions in the parameter. The programmer may choose one of the three following conditions to update the parameter: a) when a transition occurs, b) when the host CPU makes a request, or c) when a rate specified in another parameter is matched. When a pin is used as a discrete output, it is set high or low only upon request by the host CPU.

8.9.2 Input Capture/Input Transition Counter (ITC)

Any channel of the TPU can perform an input capture, i.e., capture the value of a specified TCR upon the occurrence of each transition, and then generate an interrupt request to notify the bus master. A channel can perform input captures continually, or a channel can perform a single input capture, ceasing channel activity until re-initialization. After each input capture, the channel can generate a link to a sequential block of up to eight channels. The user specifies a starting channel of the block and the number of channels within the block. The generation of links depends on the mode of operation. In addition, after each input capture, one byte of the parameter RAM (at an address specified by channel parameter) may be incremented and used as a flag to notify another channel of a transition.

Any channel of the TPU can count a programmable number of transitions, capture the specified TCR upon the occurrence of each transition, capture the specified TCR when the final transition occurs (using the different parameters), and then generate an interrupt request to notify the bus master. A channel can count the programmed number of transitions continually, or count the programmed number of transitions once, ceasing channel activity until re-initialization. After the programmed number of transitions have been counted, the channel can generate a link to a sequential block of up to eight channels. The user specifies a starting channel of the block and the number of channels within the block. The generation of links depends on the mode of operation.

8.9.3 Output Compare (OC)

The time function generates a rising edge, falling edge, or a toggle of the previous edge in one of two ways:

a. OC generates the specified edge at a programmable delay time for a user-specified time. The CPU may also force an immediate output, thereby generating a pulse with a length of the programmable delay time; or

b. Upon receiving a link from a channel, OC references, without CPU interaction, a specifiable period or parameter of another channel and adds an offset to that parameter specified by

$$\text{OFFSET} = \text{PERIOD} \bullet \text{RATIO} \qquad (8\text{-}1)$$

This algorithm generates edges continuously at a rate equal to the OFFSET value until a new link is received, at which time a new offset is calculated.

8.9.4 Pulse-Width Modulation

The TPU can generate a pulse-width modulated waveform in two ways, asynchronous and synchronized to a second channel.

8.9.4.1 Pulse-Width Modulation (PWM)

The TPU can generate a PWM waveform with any duty cycle from zero to 100% (within the resolution and latency capability of the TPU). To define the PWM, the CPU provides one parameter that indicates the period and another parameter that indicates the high time. Updates to one or both of these parameters can direct the waveform change to take effect immediately or coherently, beginning at the next low-to-high transition of the pin.

8.9.4.2 Synchronized Pulse-Width Modulation (SPWM)

The TPU generates a pulse-width modulated (PWM) waveform in which the CPU can change the period and/or high time at any time. When synchronized to a time function on a second channel, the SPWM low-to-high transitions have a time relationship to transitions on the second channel.

8

8.9.5 Period Measurement with Additional Transition Detect (PMA)

PMA allows for a special-purpose 16-bit period measurement. It detects the occurrence of an additional transition as indicated by the current period measurement being less than a programmable ratio of the previous period measurement. Once detected, this condition can be counted and compared to a programmable number of additional transitions, which is to be detected before TCR2 is reset to $FFFF. Also, one byte, at the address specified by a channel parameter, can be incremented and used as a flag. The flag indicates that TCR2 is to be reset to $FFFF once the next additional transition is detected.

8.9.6 Period Measurement with Missing Transition Detect (PMM)

PMM allows for a special-purpose 16-bit period measurement. It detects the occurrence of a missing transition as indicated by the current period measurement being more than a programmable ratio times the previous period measurement. Once detected, this condition can be counted and compared to a programmable number of transitions, which is to be detected before TCR2 is reset to $0000. Also, one byte, at the address specified by a channel parameter, can be incremented and used as a flag. The flag indicates that TCR2 is to be reset to $0000 once the next missing transition is detected.

8.9.7 Position-Synchronized Pulse Generator (PSP)

Any channel of the TPU can generate an output transition or pulse, which is a projection in time based on a reference period previously calculated on another channel. Both TCRs are used in this algorithm: TCR1 is internally clocked, whereas TCR2 is clocked by a position indicator in the user's device. An example of a TCR2 clock source is a sensor that detects special teeth on the flywheel of an automobile. The teeth are placed at known degrees of engine rotation; hence, TCR2 is a coarse representation of engine degrees, i.e., each count represents some number of degrees.

Up to 15 PSP function channels may operate with a single input reference channel executing a PMA or PMM function. The input channel measures and stores the time period between the flywheel teeth, and resets TCR2 when the engine reaches top-dead center. The output channel uses the period calculated by the input channel to project output transitions at specific engine degrees. Since the flywheel teeth might be 30 or more degrees apart, a fractional multiply resolves down to the desired degrees. Two modes of operation allow pulse length to be determined either by angular position or by time.

8.9.8 Stepper Motor (SM)

The stepper motor control algorithm provides for linear acceleration and deceleration control of a stepper motor with a programmable number of step rates of up to 14. The time period between steps (P) is defined as

$$P(r) = K1 - K2 \bullet r \tag{8-2}$$

where r is the current step rate (1–14), and K1 and K2 are programmed via the parameters.

Providing the desired step position in a 16-bit parameter, the CPU then issues a step request. Next, the TPU steps the motor to the desired position through an acceleration/deceleration profile. The parameter indicating the desired position can be changed by the CPU while the TPU is stepping the motor. This algorithm changes the control strategy every time a new step command is received. Results indicate that the algorithm provides the equivalent linearity and response of a closed-loop dc servo-motor system.
A 16-bit parameter initialized by the CPU for each channel defines the output state of the pin. The bit pattern written by the CPU defines the method of stepping, such as full-stepping or half-stepping. With each transition, the 16-bit parameter rotates one bit. The period of each transition is defined by the programmed step rate.

Any group of channels, up to eight, can be programmed to generate the control logic necessary to drive a stepper motor. Nominally, only two or four channels are used for a two-phase motor.

8.9.9 Period/Pulse-Width Accumulator (PPWA)

The PPWA algorithm continuously accumulates the high time or the total elapsed interval of a waveform over a programmable number of input periods. It continually tracks the current as well as most recent accumulated times. After an accumulation period, the algorithm can generate a link to a sequential block of up to eight channels. The user specifies a starting channel of the block and number of channels within the block. The generation of links depends on the mode of operation.

8.9.9.1 Period Measurement

Any channel can be used to measure an accumulated number of periods of an input signal. A maximum of 24 bits may be used for the accumulation parameter. From 1 to 255 period measurements can be made and summed with the previous measurement(s) before the TPU interrupts the CPU, providing instantaneous or average frequency measurement capability, and the latest complete accumulation (over the programmed number of periods).

8.9.9.2 Pulse-Width Measurement

The pulse width (high-time portion) for an input signal can be measured (up to 24 bits) and added to a previous measurement over a programmable number of periods (1 to 255). This ability provides instantaneous or average pulse-width measurement capability, and the latest complete accumulation (over the specified number of periods) is always available in a parameter.

8.9.9.3 Frequency Divide/Multiply

Using the output compare function, an output signal can be generated that is proportional to a specified input signal obtained by using PPW. The ratio of the input and output frequency is programmable. One or more output signals with different frequencies, yet proportional and synchronized to a single input signal, can be generated on separate channels.

8.10 Applications

The TPU's high speed, versatile architecture, and time functions facilitate its use in many control applications, such as stepper motors and angle-based engine control. Control of a stepper motor or an angle-based automotive engine usually

requires high CPU overhead. These applications show how the SM, PMA/PMM, and PSP time functions minimize the overhead associated with these applications, and provide sophistication and flexibility for a wide variety of applications.

8.10.1 Stepper Motor Control

The stepper motor converts electrical pulses into discrete mechanical rotational movements. Each step of the rotor can be controlled by a pulse input to a drive circuit and, therefore, provides accurate speed and position control along with long life and reliability. The stepper motor used in the present example is a two-phase permanent magnet motor that provides discrete angular movement, or steps, every time the polarity of a winding changes.

In a typical stepper motor, electrical power is applied to two coils. Two stator cups formed around each of these coils, with pole pairs mechanically displaced by one-half of a pole pitch, become alternately energized north and south magnetic poles. Between the two stator-coil pairs, the displacement is one-fourth of a pole pitch. A permanent magnet rotor inside the two stator cups is magnetized with the same number of pole pairs as contained by one stator-coil section. Interaction between the rotor and stator (opposite poles attracting and like poles repelling) causes the rotor to move one-fourth of a pole pitch per winding polarity change.

The normal electrical input is a four-step switching sequence, otherwise known as full step. Continuing the sequence rotates the rotor forward, and reversing the sequence reverses the direction of rotation. When operating at a fixed frequency, the electrical input to the motor is a two-phase 90 degrees shifted square wave as shown in Figure 8-5. Acceleration/deceleration is accomplished by controlling the frequency of the waveforms. For example, if the frequency starts at one-fourth, it accelerates to one-half, three-fourths, then to full rate. The motor can be stepped to an eight-step sequence, otherwise known as half step, i.e., a 3.75-degree step from a 7.5-degree motor.
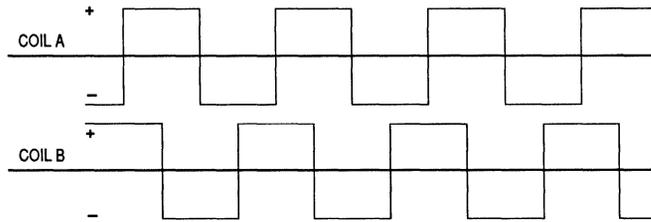
**Figure 8-5. Motor Voltage Waveform
— Fixed Frequency — Four-Step Sequence**

The stator flux in a stepper motor with a bipolar winding is reversed by reversing the current in the winding. This change requires a push-pull bipolar or metal-oxide semiconductor (MOS) H-bridge drive. Figure 8-6 illustrates the half-step and full-step drive methods of such a drive. The rate at which the drive circuitry is sequenced determines motor speed. When designing the circuit, care must be taken to avoid shorting the power supply by preventing the transistors in series from turning on simultaneously, as well as providing the flux-reversal diode and motor protection. (A unipolar winding has two coils wound on the same bobbin per stator half. Flux is reversed by energizing one coil or the other coil from a single power supply.)

FULL-STEP SEQUENCE

STEP	Q1–Q4	Q2–Q3 TP0	Q5–Q8 TP1	Q6–Q7
1	ON	OFF	ON	OFF
2	ON	OFF	OFF	ON
3	OFF	ON	OFF	ON
4	OFF	ON	ON	OFF
1	ON	OFF	ON	OFF

HALF-STEP SEQUENCE

STEP	Q1–Q4 TP0	Q2–Q3 TP1	Q5–Q8 TP2	Q6–Q7 TP3
1	ON	OFF	ON	OFF
2	ON	OFF	OFF	OFF
3	ON	OFF	OFF	ON
4	OFF	OFF	OFF	ON
5	OFF	ON	OFF	ON
6	OFF	ON	OFF	OFF
7	OFF	ON	ON	OFF
8	OFF	OFF	ON	OFF
1	ON	OFF	ON	OFF
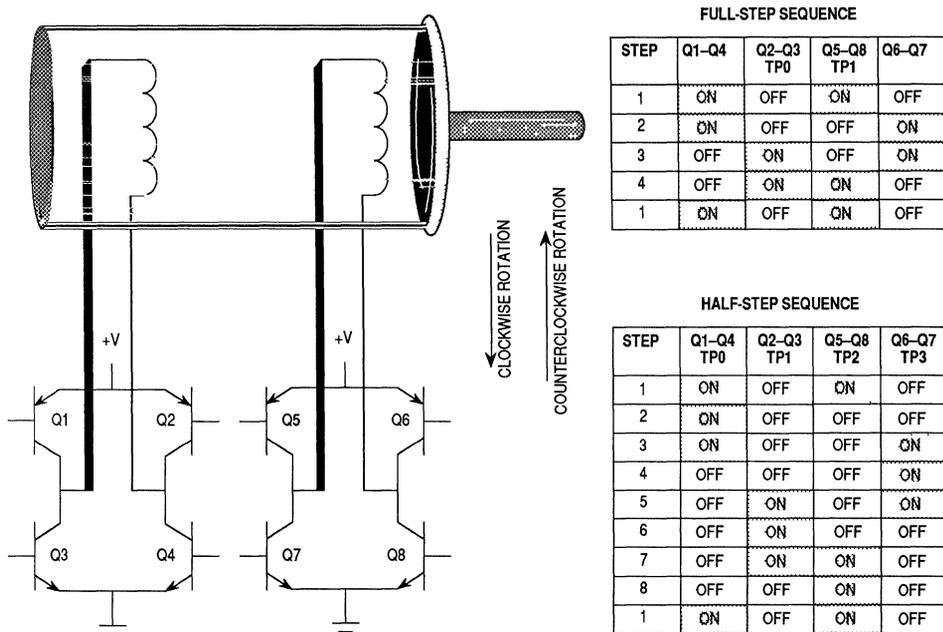
**Figure 8-6. Two-Phase Stepper Motor with
H-Bridge Drive and Step Sequences**

The SM time function provides linear acceleration/deceleration control with a programmable number of step rates up to 14. Any group of up to eight TPU pins can be used for stepper motor control. In this example, the permanent magnet motor uses two pins, TP0 and TP1, for the full-step sequence as shown in Figure 8-7, and four pins, TP3–TP0, for the half-step sequence as shown in Figure 8-8. Figure 8-9 illustrates the acceleration, full speed, and deceleration waveforms for full step, and Figure 8-10 illustrates the same for half steps.
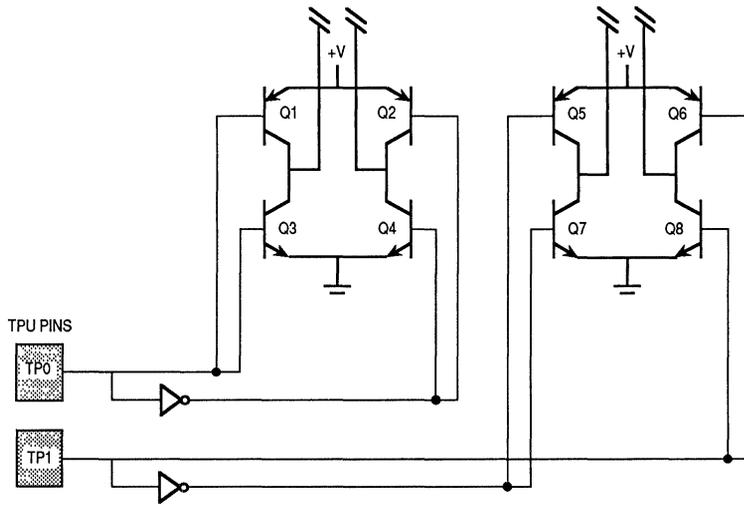
Figure 8-7. Full-Step Control
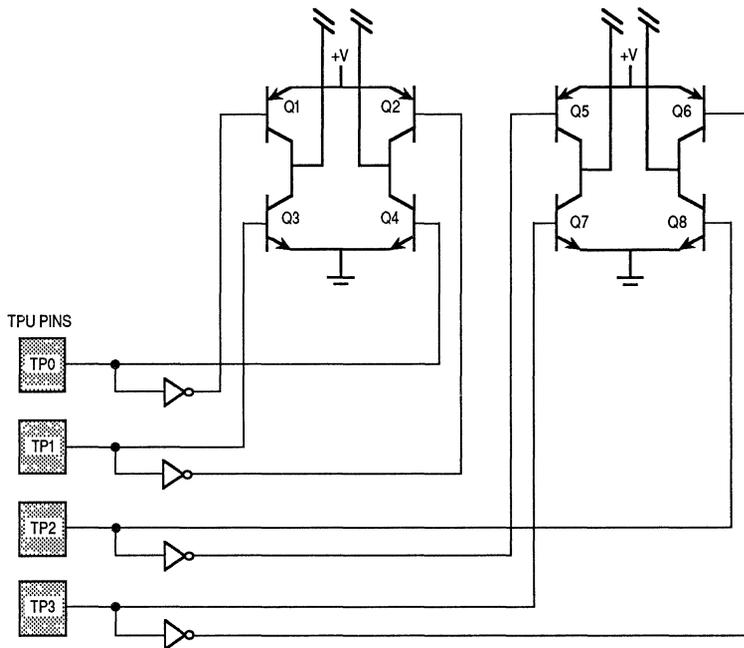
Figure 8-8. Half-Step Control
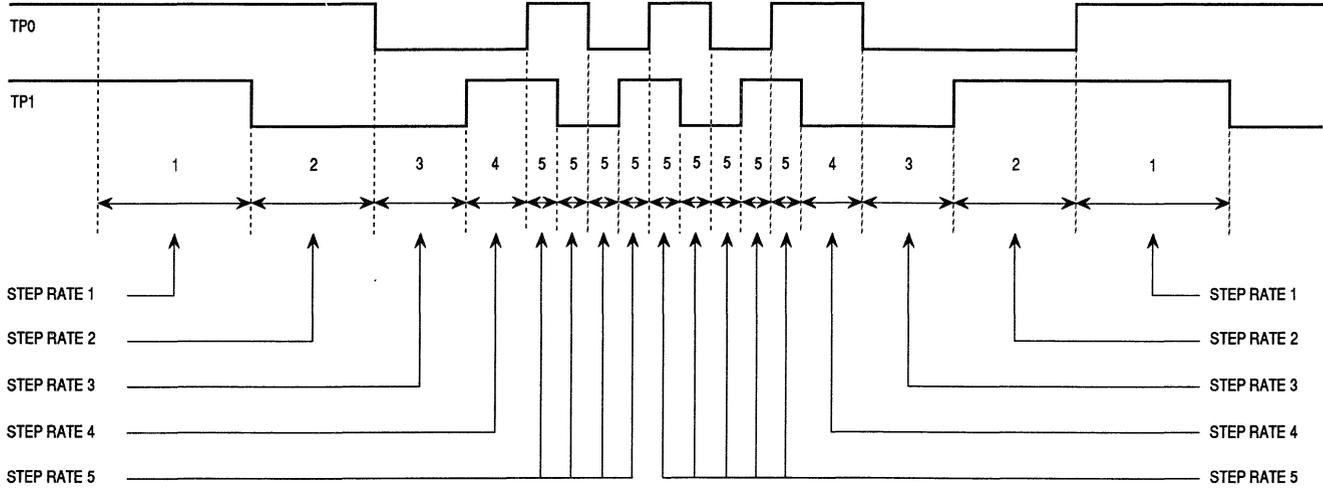
Figure 8-9. Full-Stepping Waveforms, Counterclockwise Rotation
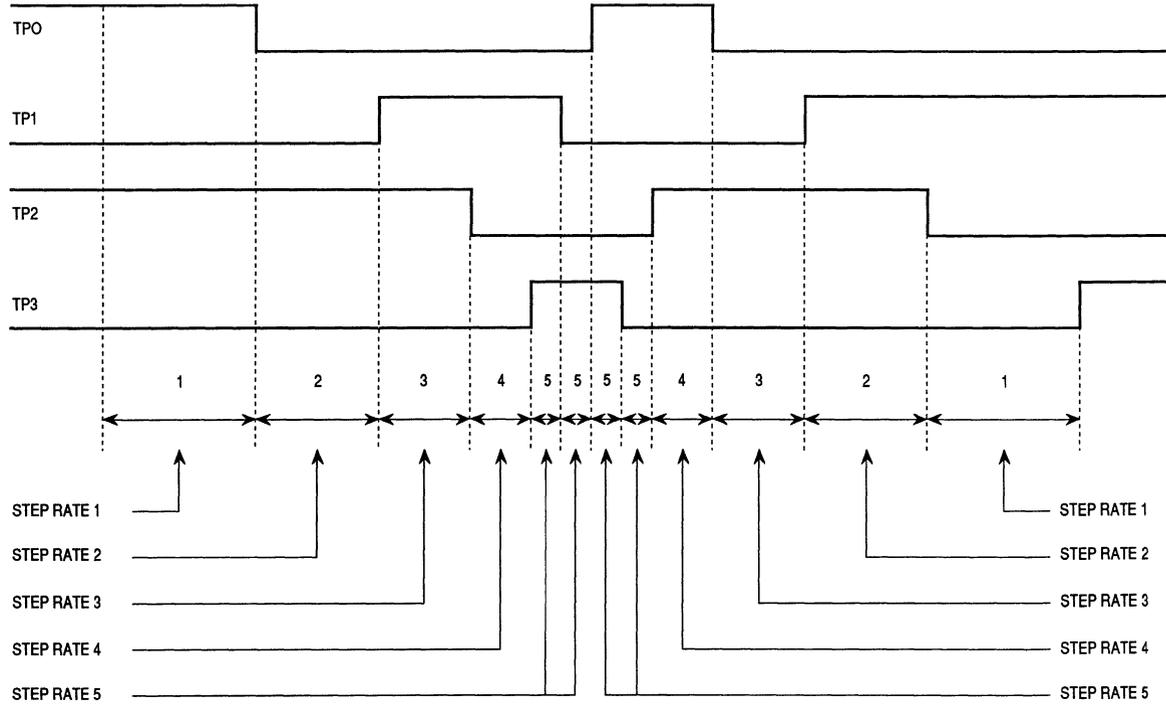
Figure 8-10. Half-Stepping Waveforms, Counterclockwise Rotation

8

A 16-bit parameter initialized by the CPU for each pin used for stepper motor control defines the method of stepping. The bit pattern of the parameter indicates the sequence of output states of the pin. The periodic rate of pin sequencing is set by the step rate(s). To facilitate full stepping, the parameter associated with TP0 should be $3333 and $9999 for TP1. For the half-step sequence, parameters $E0E0, $0E0E, $8383, and $3838 should be programmed for pins TP0, TP1, TP2, and TP3, respectively. These parameters are derived from the tables in Figure 8-6, with the MSBs defining step 1, the next MSB defining step 2, etc.

To initiate stepper motor control, the CPU provides the desired step position in a 16-bit parameter and then issues a step request. The TPU steps the motor to the desired position through an acceleration/deceleration profile stored in ROM microcode. The parameter indicating the desired position can be changed by the CPU while the TPU is stepping the motor. The control strategy of this algorithm changes every time a new step command is received. For example, if the new desired position was passed by the stepper motor stepping at full speed, the TPU decelerates the motor, reverses the rotation, and accelerates the motor to the new desired position.

8.10.2 Angle-Based Engine Control

Angle-based engine control systems generally use one or two variable-reluctance transducers (sensors) to obtain engine-cycle, position, and speed information for determining the required ignition firing points and fuel-injection pulses. (Refer to Figure 8-11 for a description of these functions.) Information is obtained from the camshaft and flywheel crankshaft, which have reference points. The reference points determine the timing for the ignition firing points and fuel-injection pulses, and are identified by special tooth patterns on the camshaft and flywheel crankshaft. Two of its predefined TPU functions, PMA and PMM, are implemented to detect special teeth.

PMA — Period Measurement with Additional Transition Detection
PMM — Period Measurement with Missing Transition Detection

With two sensors, one sensor uses the teeth (markers) on a camshaft as the reference point. The camshaft is interlocked with, and rotates at half the rate of, the flywheel, which has the second sensor, and both sensors work in conjunction to produce information. The sensor on the flywheel obtains crankshaft position information from the special tooth patterns, which contributes to engine-cycle information obtained by the sensor on the camshaft. Speed information is obtained by calculation.

A – AUTO SPARK COIL
B – FUEL VALVE
C – SIGNAL CONDITIONING AND BUFFERING ELECTRONICS
D – HIGH-CURRENT DRIVER ELECTRONICS

* Flywheel crankshaft will have either a missing tooth or an additional tooth.

Figure 8-11. Engine Control Example

The sensor on the camshaft delivers a transition to a TPU channel configured to execute an ITC function. When the proper input pattern is delivered, the ITC function increments a BANK parameter in the channel connected to the flywheel/crankshaft sensor. The BANK switch signal indicates to the BANK SIGNAL parameter in the PMA/PMM function that the next special-tooth reference point is the main reference for the entire engine and that TCR2, which is tracking the teeth on the flywheel, should be reset. Beginning at the reference point, the TPU accomplishes timing by counting the number of teeth and extrapolating from the nearest tooth. Because the teeth alone generally do not give adequate resolution, extrapolation based on previous velocity data is used to determine a more precise angular position. The TPU implements its predefined function for performing these operations: one of two functions, PMA or PMM, performs position tracking and another function, PSP, performs the extrapolation by using the period measured between the last two input transitions by the PMA/PMM function, which represent the last two teeth sensed on the crankshaft flywheel.

The PMA and PMM functions detect special teeth, which identify the beginning reference point mentioned earlier: the PMA detects additional teeth, and the PMM detects missing teeth. These functions perform the timing operation by accepting digital inputs from one or more position sensors and by performing engine position and speed tracking. Engine tracking can be programmed for a wide variety of angle-based systems, with any number of missing or additional special teeth. The condition for special tooth detection can be programmed dynamically. When the CPU updates a parameter, RATIO, one of the following conditions is tested:

a. The missing tooth condition is met if the current measured tooth period exceeds the last measured tooth period times RATIO.
b. The additional tooth condition is met if the current measured tooth period is less than the last measured tooth period times RATIO.

The PSP function performs the necessary extrapolating operation for outputs at the programmed ignition firing points or fuel-injection pulses. A rising output transition can be programmed for a precise engine angle, extrapolating from the nearest tooth. The falling output transition also can be programmed for a precise engine angle, such as that needed by the direct-fire ignition, or it may be programmed to occur at a time period following the rising transition, as required by port fuel injection. In general, the TPU can provide less than 0.1-degree resolution for systems with a large number of flywheel teeth.

The TPU provides a degree of noise immunity through two checks:

a. When each special tooth is detected, an exact, user-defined number of regular teeth must have occurred since the last special tooth, and
b. When each regular tooth is detected, the number of regular teeth between the occurrence of special teeth must not exceed the programmable number.

If an error condition is detected using these checks, the TPU generates an interrupt request to notify the CPU, which responds to the error based on the application. Error checking affords immunity to the occurrence of noise that appears to be either a normal tooth or a special tooth.

8

8

SECTION 9
EMULATION OVERVIEW

Motorola provides both low-cost and high-performance development tools to support the MC68332. The M68332EVS evaluation system provides low-cost emulation for the MC68332. The CDS32 workstation provides a cost-effective high-performance emulation environment. A free assembler (part number M68HXBASM2) and a C compiler (part number M68NXBCC300) on the MCU Bulletin Board (512-891-FREE) are available for product evaluation. Powerful, integrated software tools for MS-DOS computers are provided to support high-level-language development and debugging. Many development support tools from major third-party vendors are also available.

The background debug mode (BDM) feature on the MC68332 MCU significantly reduces the external logic necessary to provide emulation. Additionally, the use of the M68000 Family architecture for the CPU and external bus (based on the MC68020) permits the use of many existing or slightly modified existing development tools. Motorola and third-party development support companies take advantage of these features to provide new emulation and debugging options. These new features can be used at minimum system cost when provided with an interface connector on their target system to enable BDM communication.

9.1 M68332EVS

The M68332EVS low cost-development solution provides a cost-effective solution for initial MCU evaluation. The common evaluation module (EVM) monitor/debugger functions, such as memory and register modification and display, single stepping and tracing, and operand fetch breakpoints are supported. A one-line assembler/disassembler provides a quick code modification capability. Additionally, breakpoints on the access of any address are provided. The EVS is shown in Figure 9-1.

The M68332EVS is composed of three separate boards. These boards all plug together to form a total system, but can be separated for different modes of use. The M68332BCC and M68332BCCDI may be placed directly in the user's target application. Different modes of operation are possible, as shown in Figure 9-2.
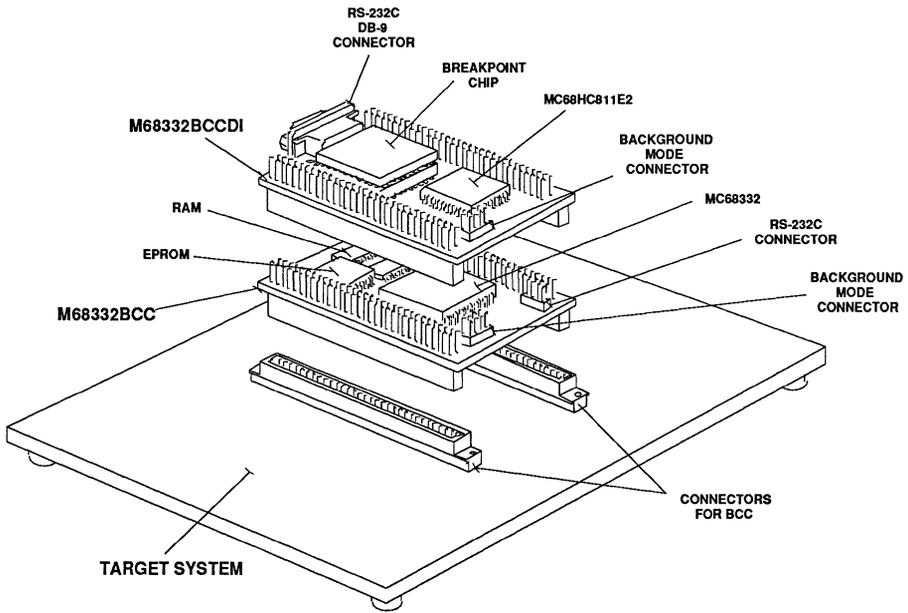
Figure 9-1. M68332EVS
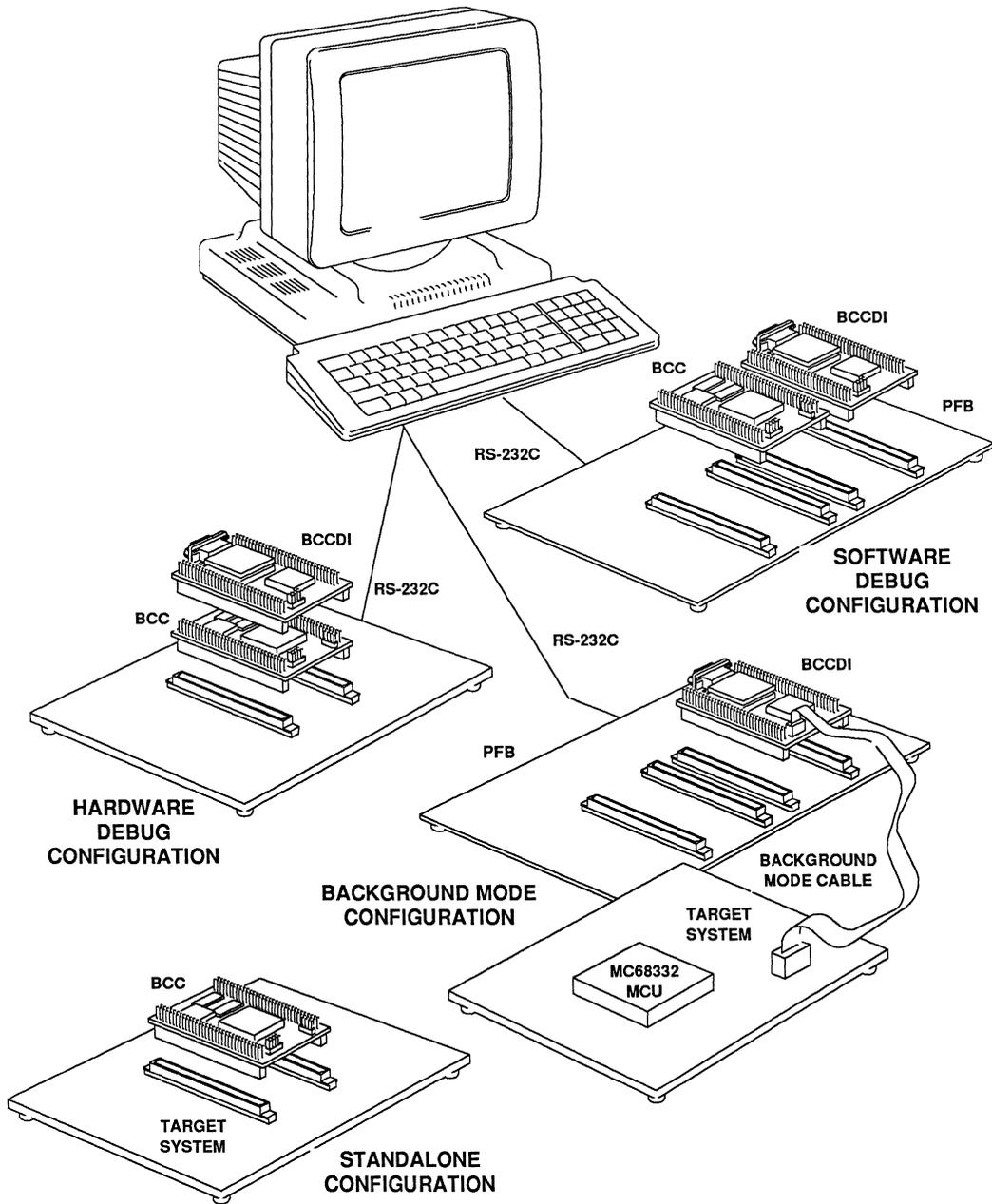
Figure 9-2. M68332 EVS Operating Modes

9.1.1 M68332BCC

The M68332BCC is the heart of the emulation system. It contains the MC68332 MCU, 128K bytes of erasable programmable read-only memory (EPROM), 64K bytes of static RAM, an RS-232C level converter chip, and a 32.768 kHz crystal, all on a printed circuit board approximately the size of a standard business card. The Business Card Computer (BCC) is sold separately. All signals from the MCU are provided by two 64-pin double-row headers that plug directly into any standard high-density breadboard.

The BCC's EPROM is a 64K × 16 array. It contains a powerful monitor/debugger for standalone operation, called 332Bug. The RAM is accessed as 32K × 16, and is used for variable storage by 332Bug as well as user emulation memory. This board runs independently of the other boards when provided with a 5-V supply and a terminal for the monitor interface. User's programs can be downloaded into RAM and executed under monitor control.

9.1.2 M68332BCCDI

The M68332BCCDI provides a development interface for the BCC. On board is an M68HC11 MCU that interfaces with the BCC via background mode. Serial communication via RS-232C to a host computer (IBM PC or clone) provides the user with EVSbug, an enhanced monitor/debugger based on the CDS32 emulator human interface, that runs under MS-DOS. A hardware breakpoint chip provides hardware breakpoints on any address access, including addresses internal to the MCU.

The BCCDI provides additional features. It executes code running on the host computer to control the BCC (hardware debug configuration), or any MC68332 via an 8-pin connector for the background mode configuration. The BCCDI is capabile of programing the EPROM on the BCC with code downloaded from the host computer when operating in the software debug mode.

9.1.3 M68332PFB

The M68332PFB provides a platform board for the EVS. The BCC and/or the BCCDI can plug directly onto the platform board (PFB). Four sockets allow the addition of 32K × 8 static RAMs, 32K × 8 EPROMS, or 64K × 8 EPROMS to provide additional emulation memory for user code. Jumper options allow the MC68332 to boot from EPROMS on the PFB. The EVS is powered by a 5-V (±10%) supply via a convenient connector. Two DB9 connectors provide RS-232C interfaces to a host computer for EVS control. One DB9 connector

interfaces the BCC to a terminal for 332BUG. The other DB9 interfaces the BCCDI to a MSDOS computer for EVSBUG operation. Other connectors on the PFB bring out the MCU's pins for connection a logic analyzer. The PFB and the BCC can be used together to perform software benchmarking and verification.

9.2 CDS32 High-Performance Emulation Solution

Based on the MC68332 MCU, this hardware development tool provides an environment that can operate with any MS-DOS host computer via RS-232C. CDS32 support of the MC68332 consists of emulation and bus-state analysis. The emulator provides full-featured, real-time emulation capability for the MC68332. Object code, created on a personal computer, can be downloaded to the CDS32 for execution. Over 1M byte of emulation memory is provided, alleviating the need for target memory during prototyping. An 8K × 160 bit analyzer buffer with four trigger event terms provides adequate capability to trace real-time events. Flexible breakpoint operation simplifies the process of hardware debug. A one-line assembler/disassembler allows quick changes to be made to the user's code. Emulation speeds up to 33 MHz are supported.

The CDS32 is integrated with the user's target system via a smart probe. This probe consists of a small PC board that contains an MC68332 and some buffer logic for bus analysis support on the CDS32. By inserting the probe into a socket on the target system in place of a MC68332, all MCU functions, clocking, and emulation memory can be provided by the CDS32. An adapter to connect a MC68332 soldered on the user's target application board is in development. Control signals and data are sent by the probe to the CDS32 station via a flat ribbon cable. This interface to the CDS32 is shown in Figure 9-3.

9

9.3 Freeware

An MS-DOS based assembler and C compiler are available free from Motorola for initial product evaluation and development. The assembler does not contain many of the advanced features found in commercial assemblers and is an unsupported product. The C compiler supports a subset of the Kernighan and Ritchie C compiler implementation and supports only the 68000 instruction set. The free software is available through the Motorola MCU Bulletin Board at (512) 891-FREE.

9.4 MS-DOS Software

Motorola software tools for the MC68332 provide a user-friendly development environment for creating and debugging code. The tools include a cross C compiler, macro-assembler/linker, and source level debugger (SLD) for MS-DOS computers. The C compiler allows code to be written in either C or

assembly language. Optimized for execution speed, the C compiler supports modular programming features and promotes software maintainability. The SLD shrinks prototyping time by providing debug capabilities at the C language level. The SLD is part of the CDS32 emulator package and is not sold seperately. The software tools operate on the IBM PC/XT, PC/AT, PS/2, and compatibles.

Third party hardware and software development tools already exist. Assemblers and compilers ar available on a wide range of computer hosts. Emulator and logic analyser support for th MC68332 is available from a variety of third party vendors as well. Contact Motorola for additional information.

Development support for the MC68332's time processor unit (TPU) is available. The 2K-byte RAM on the MC68332 can be used to emulate the TPU's ROM microcode. An assembler is provided to allow the implementation of custom timing algorithms. These may be downloaded to the RAM for TPU microcode emulation. A TPU debugger provides control and analysis of the custom algorithms to ensure correct operation. This TPU tool is available on the Motorola MCU bulletin board at (512) 891-FREE.

9.5 User Requirements

The Motorola and third-party development tools provide development support for the user, even when the MCU is positioned in the final system. This can be accomplished if the target system provides a small 2 × 4 "Berg" (or double-row header) connector (male preferred) for a connection to the development hardware. This connector should be provided by every user who wishes to test or debug application boards in final configuration. Refer to Figure 9-4 for the connector pinout.
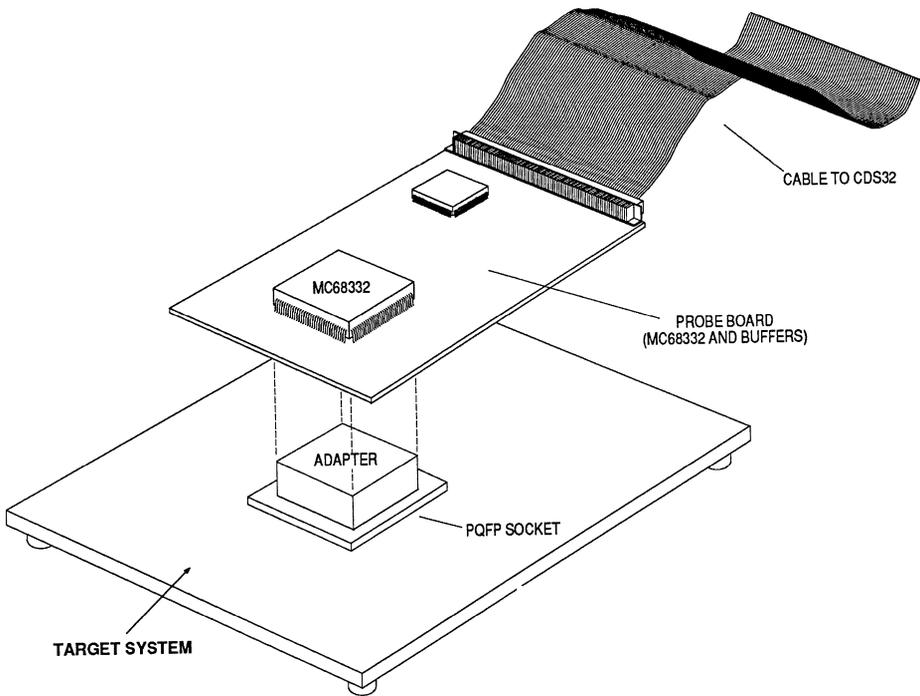
Figure 9-3. CDS32 Interface

9

```
GND 1    O  O    2 BKPT

GND 3    O  O    4 FREEZE

RESET 5  O  O    6 IFETCH

V_DD  7  O  O    8 IPIPE
```
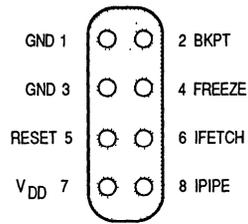
Figure 9.4. Berg Connector Pinout

The user of the MC68332 is strongly urged by Motorola to provide these signals from the MCU to the connector as outlined above. This connector may be placed anywhere on the user's target board, as shown in Figure 9-4. Both the M68332EVS and the CDS32 emulator support this interface, providing the user with both an extremely powerful debug tool, at a minimum system cost, and a means of testing the user's boards after the MCU has been soldered in place.

9

SECTION 10
ELECTRICAL CHARACTERISTICS

This section contains electrical characteristics and associated timing information for the MC68332.

10.1 Maximum Ratings

The following ratings define the conditions under which the device operates without damage. Sections of the device may not operate normally while being exposed to the electrical extremes and contains circuitry to protect against damage from high static voltages or electrical fields. It is advised, however, 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 V_{SS} or V_{DD}).

Rating	Symbol	Value	Unit
Supply Voltage	V_{DD}	−0.3 to +7.0	V
Input Voltage	V_{in}	−0.3 to +7.0	V
Operating Temperature Range MC68332C	T_A	TL to TH −40 to 85	°C
Storage Temperature Range	T_{stg}	−55 to 150	°C

10.2 Thermal Characteristics

Characteristic	Symbol	Value	Unit
Thermal Resistance Plastic 132–Pin Surface Mount	øJA	44	°C/W

10.3 Power Considerations

The average chip-junction temperature, TJ, in C can be obtained from:
$$T_J = T_A + (P_D \, ø_{JA}) \qquad\qquad (10\text{-}1)$$

where
T_A = Ambient Temperature, °C

$ø_{JA}$ = Package Thermal Resistance, Junction-to-Ambient, °C/W

P_D = P_{INT}+$P_{I/O}$

P_{INT} = $I_{DD} \times V_{DD}$, Watts — Chip Internal Power

PI/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:
$$P_D = K \div (T_J + 273°C) \qquad\qquad (10\text{-}2)$$

Solving equations 1 and 2 for K gives:
$$K = P_D + (T_A + 273°C) + ø_{JA} \times P_D{}^2 \qquad\qquad (10\text{-}3)$$

where K is a constant pertaining to the particular part. K can be determined from equation (3) by measuring P_D (at equilibrium) for a known T_A. Using this value of K, the values of PD and T_J can be obtained by solving equations (1) and (2) iteratively for any value of T_A.

10

10.4 Control Timing (V_{DD} = 5.0 Vdc 10%, V_{SS} = 0 Vdc, T_A = T_L to T_H)

Characteristic	Symbol	Min	Max	Unit
System Frequency (See Note)	f_{sys}	dc	16.78	MHz
Crystal Frequency	f_{XTAL}	25	50	kHz
On-Chip VCO System Frequency	f_{sys}	0.13	16.78	MHz
On-Chip VCO Frequency Range	f_{VCO}	0.1	35	MHz
External Clock Operation	f_{sys}	dc	16.78	MHz
PLL Startup Time (C = 0.1 μF, Stable V_{DD} and Crystal)	t_{rc}	—	10	ms

NOTE: All internal registers retain data at 0 Hz.

10.5 DC Characteristics (V_{DD} = 5.0 Vdc ±10%, V_{SS} = 0 Vdc, $T_A = T_L$ to T_H)

Characteristic	Symbol	Min	Max	Unit
Input High Voltage	V_{IH}	$0.7 \times V_{DD}$	V_{DD} +0.3	V
Input Low Voltage	V_{IL}	V_{SS} −0.3	$0.2 \times V_{DD}$	V
Input Leakage Current V_{in}=V_{DD} or V_{SS} TSTME/TSC, BKPT, T2CLK, RXD	I_{in}	−2.5	2.5	µA
Hi-Z (Off-State) Leakage Current (See Note 1) V_{in}=V_{DD} or V_{SS} Group 1 and Group 2 I/O Pins V_{in}=V_{DD} or V_{SS} HALT, RESET	I_{OZ}	 −2.5 −20	 2.5 20	µA
Output High Voltage (See Notes 1 and 2) I_{OH} =−0.8 mA, V_{DD} =4.5 V All Outputs except HALT, RESET	V_{OH}	V_{DD} −0.8	—	V
Output Low Voltage (see Note 1) I_{OL} =1.6 mA CLKOUT, FREEZE/QUOT, IPIPE, Group 1 I/O Pins I_{OL} =5.3 mA CSBOOT, BG/ CS and Group 2 I/O Pins I_{OL} =15.3 mA HALT, RESET	V_{OL}	 — — —	 0.4 0.4 0.4	V
RAM Standby Voltage	V_{SB}	3.0	V_{DD}	V
RAM Standby Current	I_{SB}	—	50	µA
Total Supply Current RUN (See Note 3) STOP (VCO Off)	I_{DD} SI_{DD}	 — —	 125 500	 mA µA
Power Dissipation	P_D	—	690	mW
Input Capacitance (See Notes 1 and 4) All Input-Only Pins All I/O Pins	C_{in}	 — —	 10 20	pF
Load Capacitance (See Note 1) CLKOUT, FREEZE/QUOT, IPIPE and Group 1 I/O Pins CSBOOT, BG/CS and Group 2 I/O Pins HALT, RESET	C_L	 — — —	 90 100 130	pF

NOTES:
1. Input-Only Pins: TSTME/TSC, BKPT, T2CLK, RXD
   Output-Only Pins: CSBOOT, BG/CS, CLKOUT, FREEZE/QUOT, IPIPE
   Input/Output Pins:
       Group 1: D15–D0, IFETCH, Port A (TP15–TP8), Port B (TP7–TP0)
       Group 2: Port C (A23–A19/CS, FC2–FC0/CS), Port D (MISO, MOSI/SDA, SCK, PCS0/SS, PCS3–
           PCS1,TXD), PortE DSACK0, DSACK1, AVEC, RMC, DS, AS, SIZ0, SIZ1), Port F (MODCK, IRQ7–
           IRQ1), A18–A0, R/W, BERR, BR/CS, BGACK/CS
       Group 3: HALT, RESET
2. V_{OH} specification for HALT and RESET is not applicable because they are open-drain pins. V_{OH} specification is not applicable to Port D (MISO, MOSI/SDA, SCK, PCS0/SS, PCS3–PCS1 ,TXD) in wire-OR mode.
3. Supply current measured with system clock frequency of 16.78 MHz.
4. Input capacitance is periodically sampled rather than 100% tested.

10

10.6 AC Timing Specifications (V_{DD} = 5.0 Vdc ±10%, V_{SS} = 0 Vdc, T_A = T_L to T_H, see Note 1)

Num	Characteristic	Symbol	Min	Max	Unit
	Frequency of Operation	f	0.13	16.78	MHz
1	Clock Period	t_{cyc}	59.6	—	ns
1A	E Clock Period	t_{Ecyc}	476	—	ns
2, 3	Clock Pulse Width	t_{CW}	28	—	ns
2A, 3A	E Clock Pulse Width	t_{ECW}	236	—	ns
4, 5	Clock Rise and Fall Time	t_{Crf}	—	5	ns
4A, 5A	Rise and Fall Time — All Outputs except CLKOUT	t_{rf}	—	8	ns
6	Clock High to Address, FC, SIZE, $\overline{RMC}$ Valid	t_{CHAV}	0	30	ns
7	Clock High to Address, Data, FC, SIZE, $\overline{RMC}$ High Impedance	t_{CHAZx}	0	60	ns
8	Clock High to Address, FC, SIZE, $\overline{RMC}$ Invalid	t_{CHAZn}	0	—	ns
9	Clock Low to $\overline{AS}$, $\overline{DS}$, $\overline{CS}$, $\overline{IFETCH}$ Asserted	t_{CLSA}	3	30	ns
9A[2]	$\overline{AS}$ to $\overline{DS}$ or $\overline{CS}$ Asserted (Read)	t_{STSA}	−15	15	ns
11	Address, FC, SIZE, $\overline{RMC}$ Valid to $\overline{AS}$, $\overline{CS}$ (and $\overline{DS}$ Read) Asserted	t_{AVSA}	15	—	ns
12	Clock Low to $\overline{AS}$, $\overline{DS}$, $\overline{CS}$, $\overline{IFETCH}$ Negated	t_{CLSN}	3	30	ns
13	$\overline{AS}$, $\overline{DS}$, $\overline{CS}$ Negated to Address, FC, SIZE Invalid (Address Hold)	t_{SNAI}	15	—	ns
14	$\overline{AS}$, $\overline{CS}$ (and $\overline{DS}$ Read) Width Asserted	t_{SWA}	100	—	ns
14A	$\overline{DS}$, $\overline{CS}$ Width Asserted Write	t_{SWAW}	45	—	ns
14B	$\overline{AS}$, $\overline{CS}$ (and $\overline{DS}$ Read) Width Asserted (Synchronous Cycle)	t_{SWDW}	40	—	ns
15[3]	$\overline{AS}$, $\overline{DS}$, $\overline{CS}$ Width Negated	t_{SN}	40	—	ns
16	Clock High to $\overline{AS}$, $\overline{DS}$, R/$\overline{W}$ High Impedance	t_{CHSZ}	—	60	ns
17	$\overline{AS}$, $\overline{DS}$, $\overline{CS}$ Negated to R/$\overline{W}$ High	t_{SNRN}	15	—	ns
18	Clock High to R/$\overline{W}$ High	t_{CHRH}	0	30	ns
20	Clock High to R/$\overline{W}$ Low	t_{CHRL}	0	30	ns
21	R/$\overline{W}$ High to $\overline{AS}$ Asserted	t_{RAAA}	15	—	ns
22	R/$\overline{W}$ Low to $\overline{DS}$ Asserted (Write)	t_{RASA}	70	—	ns
23	Clock High to Data Out Valid	t_{CHDO}	—	30	ns
24	Data Out Valid to Negating Edge of $\overline{AS}$ (Synchronous Write)	t_{DVASN}	15	—	ns
25	$\overline{DS}$, $\overline{CS}$ Negated to Data Out Invalid (Data Out Hold)	t_{SNDOI}	15	—	ns
26	Data Out Valid to $\overline{DS}$ Asserted (Write)	t_{DVSA}	15	—	ns
27[9]	Data In Valid to Clock Low (Data Setup)	t_{DICL}	5	—	ns
27A	Late $\overline{BERR}$, $\overline{HALT}$, $\overline{BKPT}$ Asserted to Clock Low (Setup Time)	t_{BELCL}	20	—	ns
28	$\overline{AS}$, $\overline{DS}$ Negated to $\overline{DSACKx}$, $\overline{BERR}$, $\overline{HALT}$, $\overline{AVEC}$ Negated	t_{SNDN}	0	80	ns
29[4]	$\overline{DS}$ Negated to Data In Invalid (Data In Hold)	t_{SNDI}	0	—	ns
29A[4]	$\overline{DS}$ Negated to Data In High Impedance	t_{SHDI}	—	60	ns
30[4]	CLKOUT Low to Data In Invalid (Synchronous Hold)	t_{CLDI}	15	—	ns
30A[4]	CLKOUT Low to Data In High Impedance	t_{CLDH}	—	90	ns

10

10.6 AC Timing Specifications (continued)

Num	Characteristic	Symbol	Min	Max	Unit
31[5]	$\overline{\text{DSACKx}}$ Asserted to Data In Valid	t_{DADI}	—	50	ns
32	$\overline{\text{HALT}}$ and $\overline{\text{RESET}}$ Input Transition Time	t_{HRrf}	—	200	ns
33	Clock Low to $\overline{\text{BG}}$ Asserted	t_{CLBA}	—	30	ns
34	Clock Low to $\overline{\text{BG}}$ Negated	t_{CLBN}	—	30	ns
35[6]	$\overline{\text{BR}}$ Asserted to $\overline{\text{BG}}$ Asserted ($\overline{\text{RMC}}$ Not Asserted)	t_{BRAGA}	1	—	Clks
37	$\overline{\text{BGACK}}$ Asserted to $\overline{\text{BG}}$ Negated	t_{GAGN}	1	2	Clks
39	$\overline{\text{BG}}$ Width Negated	t_{GH}	2	—	Clks
39A	$\overline{\text{BG}}$ Width Asserted	t_{GA}	1	—	Clks
46	$R/\overline{W}$ Width Asserted (Write or Read)	t_{RWA}	150	—	ns
46A	$R/\overline{W}$ Width Asserted (Synchronous Write or Read)	t_{RWAS}	90	—	ns
47A[9]	Asynchronous Input Setup Time $\overline{\text{BR}}$,$\overline{\text{BG}}$, $\overline{\text{DSACKx}}$, $\overline{\text{BERR}}$, $\overline{\text{AVEC}}$, $\overline{\text{HALT}}$	t_{AIST}	5	—	ns
47B	Asynchronous Input Hold Time	t_{AIHT}	15	—	ns
48[7]	$\overline{\text{DSACKx}}$ Asserted to $\overline{\text{BERR}}$, $\overline{\text{HALT}}$ Asserted	t_{DABA}	—	30	ns
53	Data Out Hold from Clock High	t_{DOCH}	0	—	ns
54	Clock High to Data Out High Impedance	t_{CHDH}	—	28	ns
55	$R/\overline{W}$ Asserted to Data Bus Impedance Change	t_{RADC}	40	—	ns
56	$\overline{\text{RESET}}$ Pulse Width (Reset Instruction)	t_{HRPW}	512	—	Clks
57	$\overline{\text{BERR}}$ Negated to $\overline{\text{HALT}}$ Negated (Rerun)	t_{BNHN}	0	—	ns
70	Clock Low to Data Bus Driven (Show)	t_{SCLDD}	0	30	ns
71	Data Setup Time to Clock Low (Show)	t_{SCLDS}	15	—	ns
72	Data Hold from Clock Low (Show)	t_{SCLDH}	10	—	ns
80	Address, $R/\overline{W}$ Valid to E Rise	t_{EAV}	75	—	ns
81	Address, $R/\overline{W}$ Hold Time	t_{EAH}	30	—	ns
82[8]	Address Access Time	t_{EACCA}	289	—	ns
83[8]	MPU Access Time	t_{EACCE}	206	—	ns
84	Read Data Setup Time	t_{EDSR}	30	—	ns
85	Read Data Hold Time	t_{EDHR}	5	—	ns
85A	E Low to Data In High Impedance	t_{ELDI}	—	60	ns
86	E Fall to Write Data Driven	t_{EFWDD}	0	60	ns
87	Write Data Delay Time	t_{EDDW}	—	0	ns
88	Write Data Hold Time	t_{EDHW}	30	—	ns
89	Address, $R/\overline{W}$ Valid to $\overline{\text{AS}}$, $\overline{\text{CS}}$ Fall	t_{EASL}	15	—	ns
90	Delay Time, $\overline{\text{AS}}$ to E Rise	t_{EASED}	150	—	ns
91	Delay Time, E Low to $\overline{\text{AS}}$ Negated	t_{ELASN}	0	—	ns
91A	Delay Time, E Low to $\overline{\text{CS}}$ Negated	t_{ELCSN}	20	—	ns
92	Pulse Width $\overline{\text{AS}}$ Negated (E Cycle)	t_{EPWASH}	30	—	ns
S1	Slave Mode $\overline{\text{AS}}$, $\overline{\text{DS}}$ Valid to Clock High	t_{SASCH}	10	—	ns

10

10.6 AC Timing Specifications (concluded)

Num	Characteristic	Symbol	Min	Max	Unit
S2	Slave Mode Address, R/$\overline{W}$, FC, SIZ Valid to Clock Low	t_{SAVCL}	15	—	ns
S3	Slave Mode Address, R/$\overline{W}$, FC, SIZ Hold Time from Clock Low	t_{SAHCL}	15	—	ns
S4	Slave Mode Clock High to $\overline{DSACK}$ Asserted	t_{SDSKA}	—	30	ns
S5	Slave Mode $\overline{DSACK}$ Hold Time from Clock Low	t_{SDSCKH}	0	30	ns
S6	Slave Mode Clock Low to Read Data Valid	t_{SCLDV}	—	30	ns
S7	Slave Mode Read Data Hold Time from Clock Low	t_{SSRDH}	0	30	ns
S8	Slave Mode Write Data Input Setup Time to Clock Low	t_{SDSCL}	20	—	ns
S9	Slave Mode Write Data Hold Time from Clock Low	t_{SDHCL}	20	—	ns
S10	Slave Mode $\overline{DSACK}$ Asserted to $\overline{AS}$, $\overline{DS}$ Negated	t_{SDLAH}	0	—	ns

NOTES:
1. All AC timing is shown with respect to 20% V_{DD} and 70% V_{DD} 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 $\overline{CS}$ 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. The $\overline{CS}$ width negated specification between multiple chip selects does not apply to chip selects being used for synchronous E cycles.
4. These hold times are specified with respect to $\overline{DS}$ on asynchronous reads and with respect to CLKOUT on synchronous reads. The user is free to use either hold time.
5. If the asynchronous setup time (#47A) requirements are satisfied, the $\overline{DSACKx}$ low to data setup time (#31) and $\overline{DSACKx}$ low to $\overline{BERR}$ low setup time (#48) can be ignored. The data must only satisfy the data-in to clock low setup time (#27) for the following clock cycle; $\overline{BERR}$ must satisfy only the late $\overline{BERR}$ low to clock low setup time (#27A) for the following clock cycle.
6. To ensure coherency during every operand transfer, $\overline{BG}$ will not be asserted in response to $\overline{BR}$ until after all cycles of the current operand transfer are complete and RMC is negated.
7. In the absence of DSACKx, BERR is an asynchronous input using the asynchronous setup time (#47A).
8. Synchronous E cycle address access time = $t_{EAV}+t_{rf}+t_{ECW}-t_{EDSR}$ = 289 ns (16.78 MHz clock).
   Synchronous E Cycle MPU access time = $t_{ECW}-t_{EDSR}$=206 ns (16.78 MHz clock).
9. Initial masks of MC68332 devices require increased input setup times. Motorola will be testing the input data setup time (t_{DICL}) and the asynchronous input setup time (t_{AIST}) at 15 ns maximum until circuit improvements are completed. The interim data setup time value will decrease each access time defined in Note 10 by 10 ns.
10. Address access time = $2t_{cyc}+t_{CW}-t_{CHAV}-t_{DICL}$ = 112.2 ns (16.78 MHz clock).
    Chip-Select access time = $2t_{cyc}-t_{CLSA}-t_{DICL}$ = 84.2 ns (16.78 MHz clock).
    Synchronous Address Access Time = $t_{cyc}+t_{CW}-t_{CHAV}-t_{DICL}$ = 52.6 ns (16.78 MHz clock).
    Synchronous Chip-Select Access Time = $t_{cyc}-t_{CLSA}-t_{DICL}$ = 24.6 ns (16.78 MHz clock).
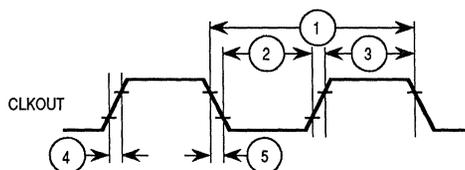
NOTE: All timing shown with respect to 20% and 70% V$_{DD}$

Figure 10-1. Clock Output Timing

Figure 10-2. Read Cycle Timing Diagram
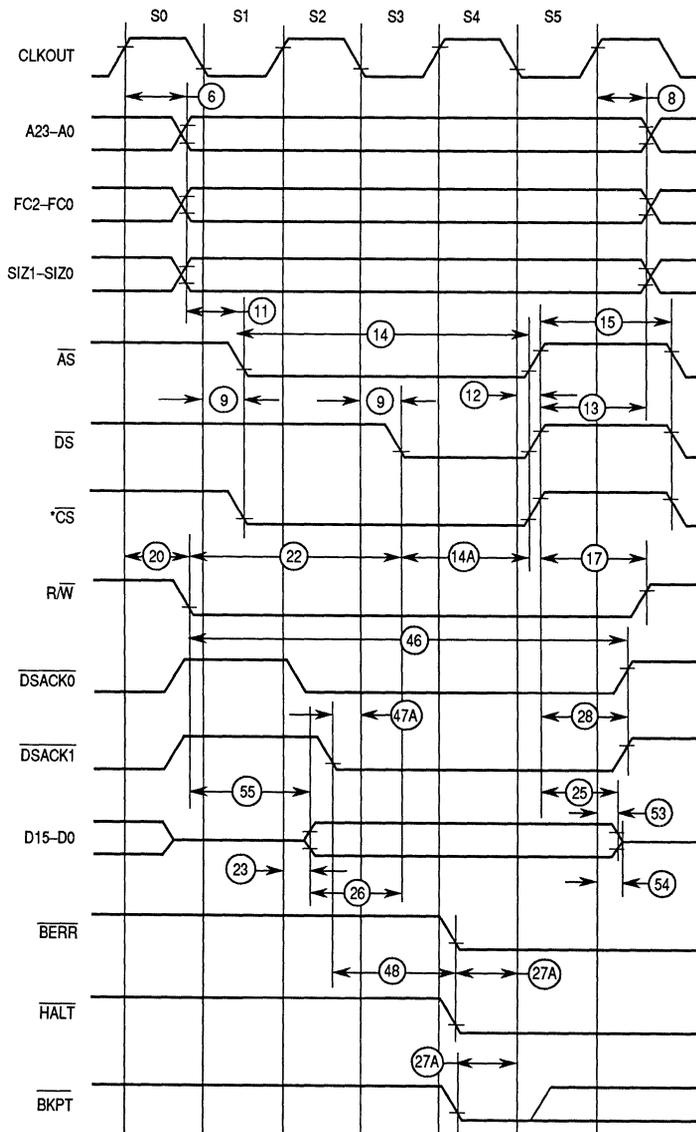
NOTE: All timing with respect to 20% and 70% V_{DD}
*Timing dependent on programed options (i.e., timed with respect to $\overline{\text{DS}}$ or $\overline{\text{AS}}$ assertion)

Figure 10-3. Write Cycle Timing Diagram
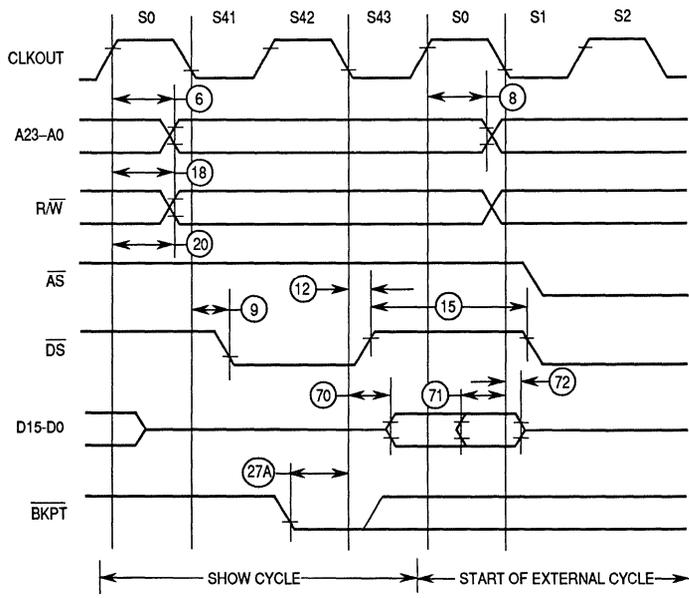
Figure 10-4. Show Cycle Timing Diagram

NOTE: All timing with respect to 20% and 70% V_{DD}

10
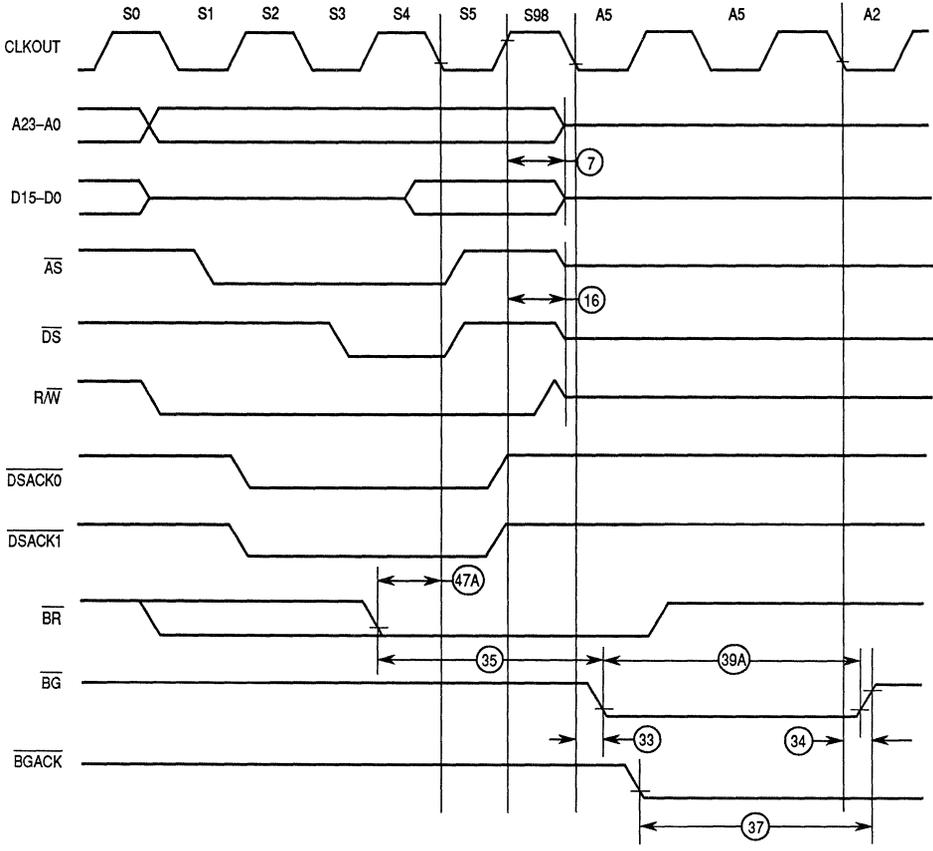
Figure 10-5. Bus Arbitration Timing Diagram — Active Bus Case

NOTE: All timing with respect to 20% and 70% V$_{DD}$
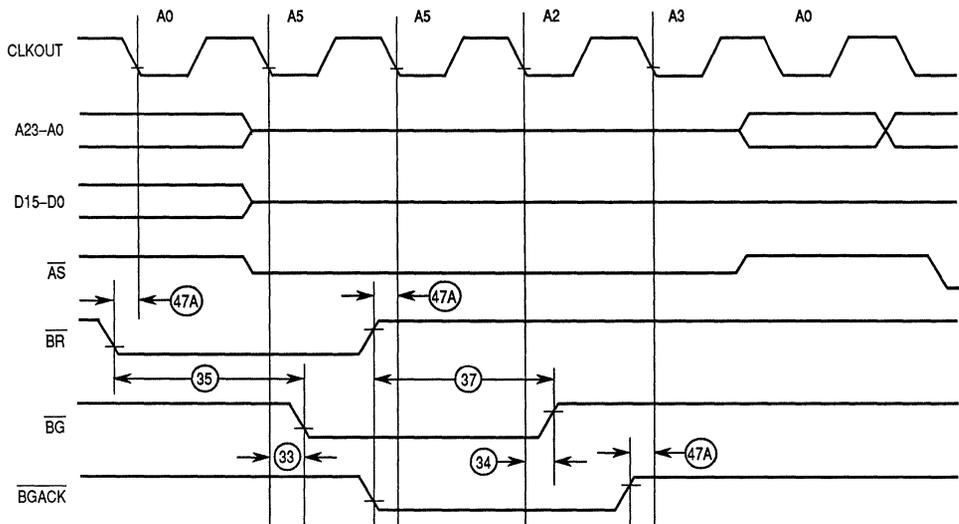
NOTE: All timing with respect to 20% and 70% V_{DD}

Figure 10-6. Bus Arbitration Timing Diagram — Idle Bus Case
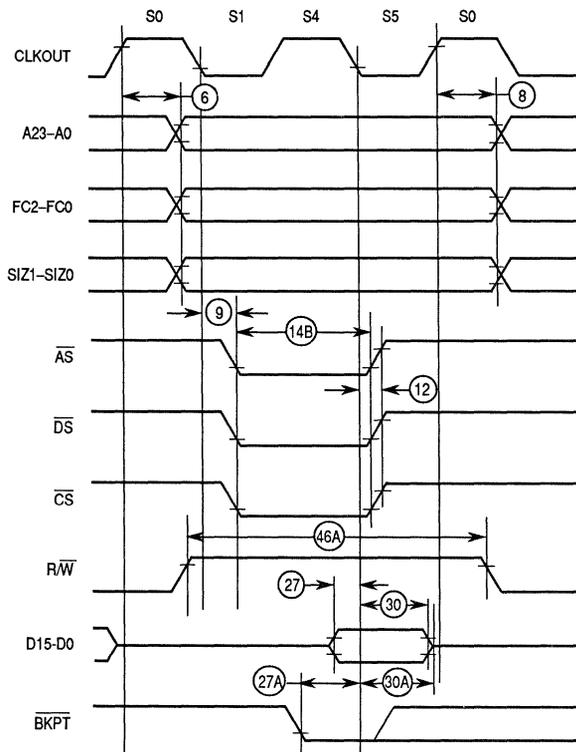
NOTE: All timing with respect to 20% and 70% V_{DD}

Figure 10-7. Synchronous Read Cycle Timing Diagram

ELECTRICAL CHARACTERISTICS   MC68332 USER'S MANUAL
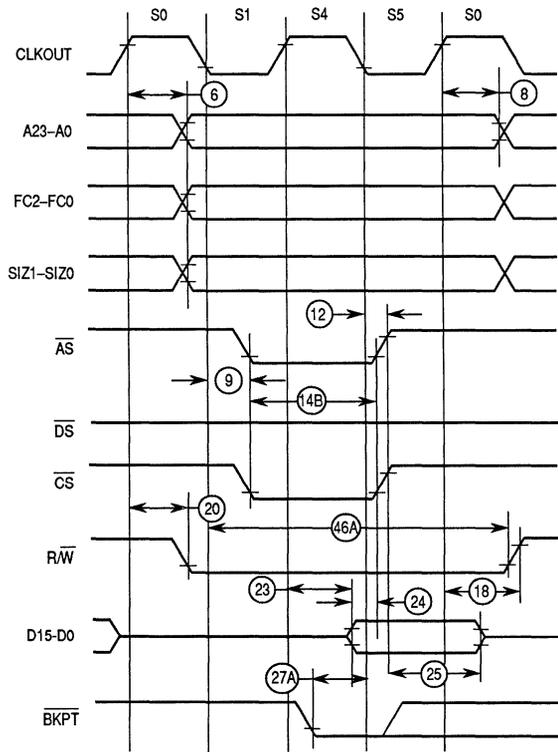
NOTE: All timing with respect to 20% and 70% V_{DD}

Figure 10-8. Synchronous Write Cycle Timing Diagram
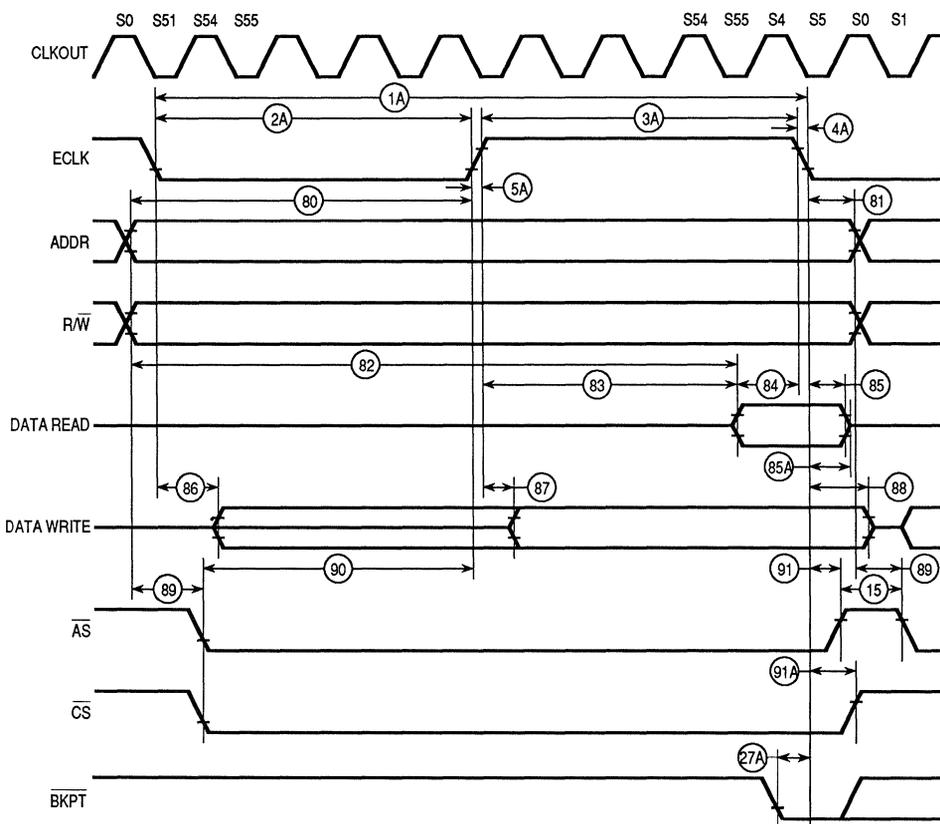
NOTE: All timing is shown with respect to 20% and 70% V_{DD}.

Figure 10-9. Synchronous E-Cycle Timing Diagram

SECTION 11
ORDERING INFORMATION

This section contains detailed information to be used as a guide when ordering.

The XC68332FE is a conditionally qualified part and is currently available. The MC68332FC is the fully qualified part and should be available in the second quarter of 1991. Please check with your local Motorola representative or distributor for current availability of these parts.

11.1 Standard MC68332 Ordering Information

Package Type	Frequency (MHz)	Temperature	Order Number
Ceramic Surface Mount FE Suffix	16.78	−40°C to +85°C	XC68332FE (now)
			MC68332FE (1Q91)
			SPAKXC68332FE*
			SPAKMC68332FE*
Plastic Surface Mount FC Suffix	16.78	−40°C to +85°C	XC68332FC (1Q91)
			MC68332FC (2Q91)
			SPAKXC68332FC*
			SPAKMC68332FC*

*These are sample packs and must be ordered in multiples of two parts and are shipped in special cases.

11

11

SECTION 12
MECHANICAL DATA

12.1 FC and FE Suffix — Pin Assignment
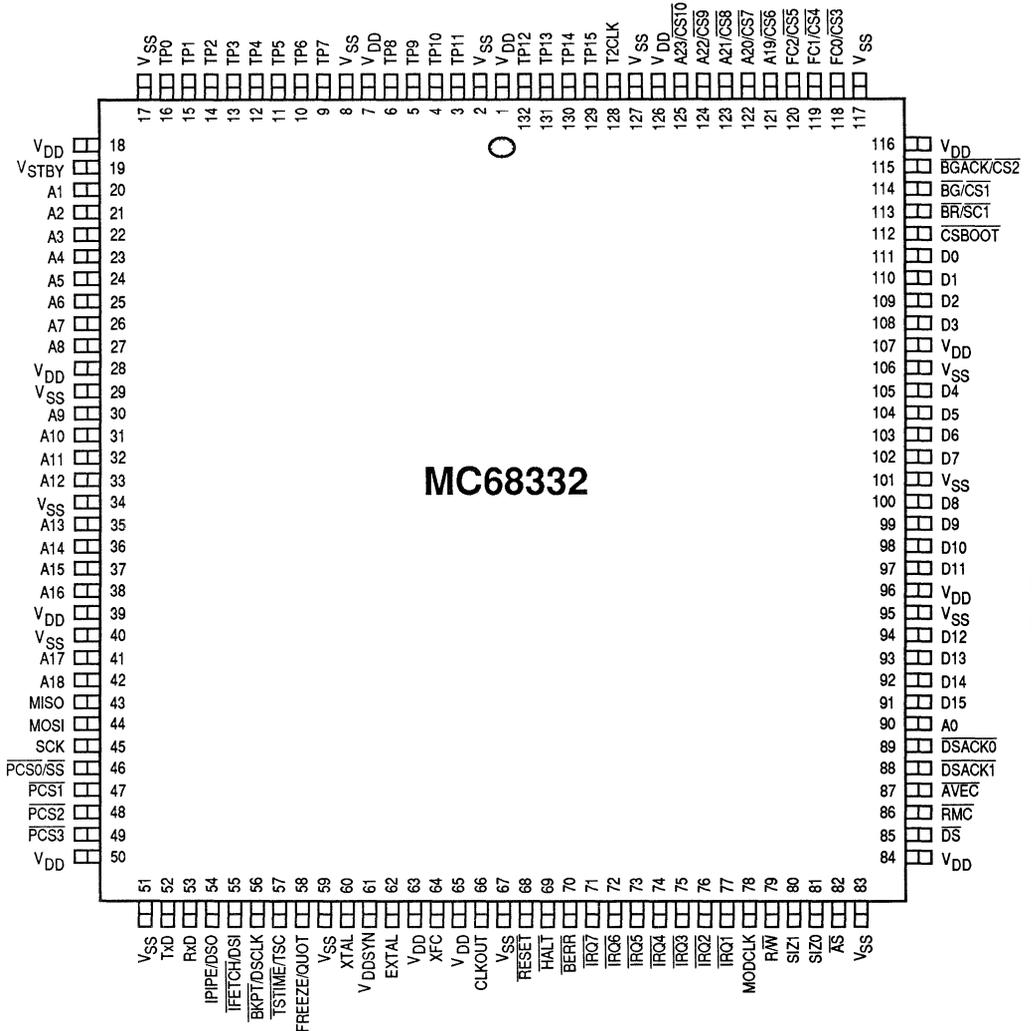
12.2 FC Suffix — Package Dimensions

FC SUFFIX

PLASTIC SURFACE MOUNT

CASE 831A-01
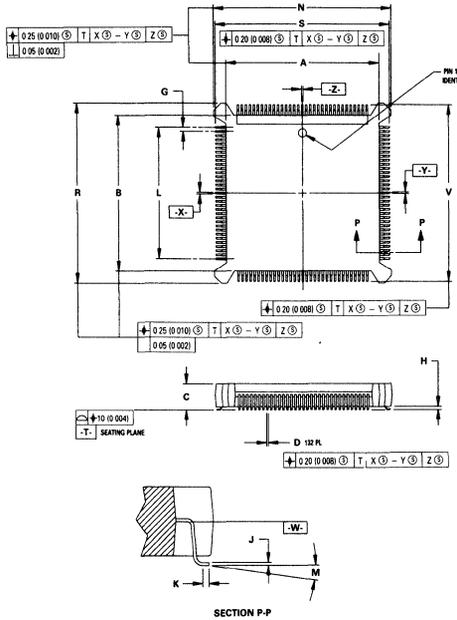
DIM	MILLIMETERS		INCHES	
	MIN	MAX	MIN	MAX
A	24.06	24.20	0.947	0 953
B	24 06	24.20	0.947	0.953
C	4.07	4.57	0.160	0.180
D	0.21	0.30	0.008	0.012
G	0.64 BSC		0 025 BSC	
H	0 51	1 01	0 020	0.040
J	0.16	0 20	0 006	0 008
K	0 51	0 76	0.020	0 030
L	20.32 REF		0 800 REF	
M	0°	8°	0°	8°
N	27.88	28 01	1 097	1.103
R	27 88	28.01	1.097	1.103
S	27 31	27 55	1.075	1.085
V	27 31	27 55	1.075	1 085

NOTES:
1. DIMENSIONING AND TOLERANCING PER ANSI Y14 5M, 1982.
2. CONTROLLING DIMENSION: INCH
3. DIMENSIONS A, B, N. AND R DO NOT INCLUDE MOLD PROTRUSION. ALLOWABLE MOLD PROTRUSION FOR DIMENSIONS A AND B IS 0 25 (0.010), FOR DIMENSIONS N AND R IS 0.18 (0.007).
4. DATUM PLANE -W- IS LOCATED AT THE UNDERSIDE OF LEADS WHERE LEADS EXIT PACKAGE BODY
5. DATUMS X-Y AND Z TO BE DETERMINED WHERE CENTER LEADS EXIT PACKAGE BODY AT DATUM -W-
6. DIMENSIONS S AND V TO BE DETERMINED AT SEATING PLANE, DATUM -T-.
7. DIMENSIONS A, B, N, AND R TO BE DETERMINED AT DATUM PLANE -W-

12

12.3 FE Suffix — Package Dimensions

FE SUFFIX
CERAMIC SURFACE MOUNT
CASE 831-01
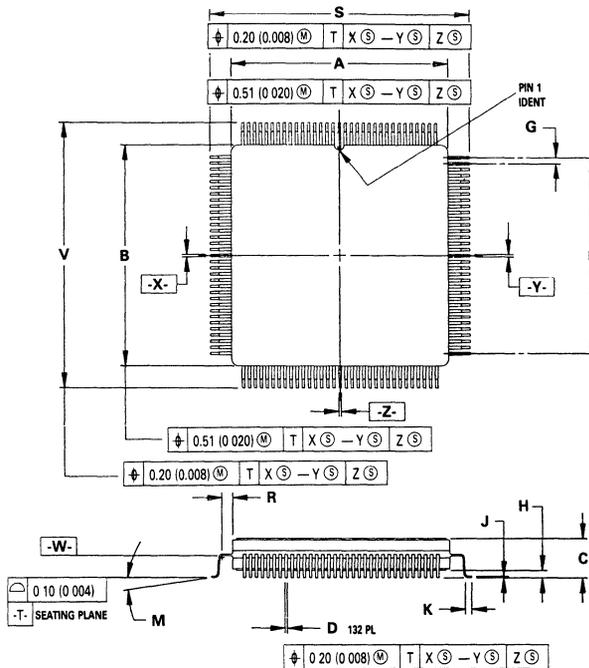
DIM	MILLIMETERS		INCHES	
	MIN	MAX	MIN	MAX
A	21.85	22.86	0.860	0.900
B	21.85	22.86	0.860	0.900
C	3.94	4 31	0.155	0 170
D	0.204	0.292	0.0080	0 0115
G	0 64 BSC		0 025 BSC	
H	0.64	0.88	0 025	0 035
J	0.13	.0 20	0.005	0 008
K	0.51	0.76	0.020	0 030
L	20 32 REF		0 800 REF	
M	0°	8°	0°	8°
R	0 64	—	0 025	—
S	27 31	27 55	1.075	1 085
V	27.31	27 55	1.075	1.085

CASE 831-01

NOTES.
1 DIMENSIONING AND TOLERANCING
   PER ANSI Y14 5M, 1982
2 CONTROLLING DIMENSION INCH
3 DIM A AND B DEFINE MAXIMUM CERAMIC BODY
   DIMENSIONS INCLUDING GLASS PROTRUSION
   AND MISMATCH OF CERAMIC BODY TOP AND
   BOTTOM.
4 DATUM PLANE -W- IS LOCATED AT THE
   UNDERSIDE OF LEADS WHERE LEADS EXIT
   PACKAGE BODY
5 DATUMS X–Y AND Z TO BE DETERMINED
   WHERE CENTER LEADS EXIT PACKAGE BODY AT
   DATUM -W-
6 DIM S AND V TO BE DETERMINED AT SEATING
   PLANE, DATUM -T-
7 DIM A AND B TO BE DETERMINED AT DATUM
   PLANE -W-

12

12

MECHANICAL DATA

APPENDIX A
MC68332 MEMORY MAP

A.1 MC68332 Module Memory Map

$YFF000	
$YFF800	
$YFFA00	SIM
$YFFA80	— —RESERVED— —
$YFFB00	RAM CTRL
$YFFB40	— —RESERVED— —
$YFFC00	
	QSM
$YFFE00	
	TPU
$YFFFFF	

RAM ARRAY
(2.0K BYTES)
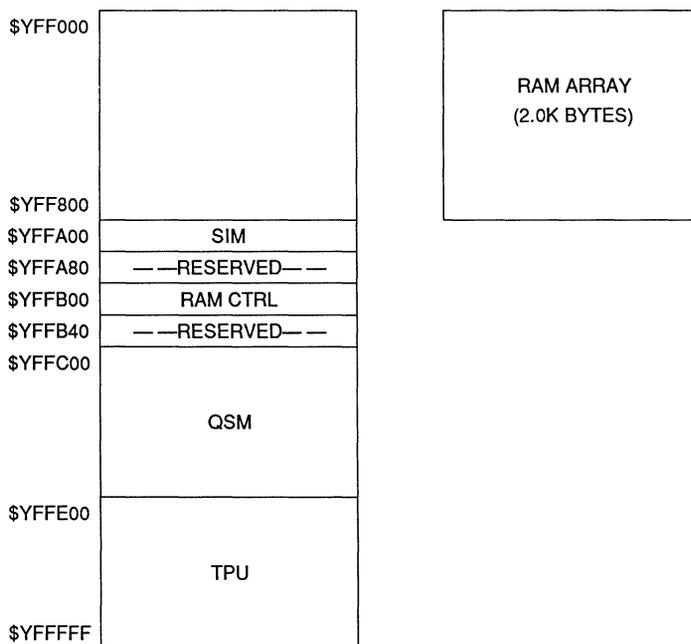
Module	Size (Bytes)	Address Bus Decoding						Base Address
		A23	—	A12	A11	—	A0	
SIM	128	M111	1111	1111	1010	0XXX	XXXX	$YFFA00
RAM CTRL	64	M111	1111	1111	1011	00XX	XXXX	$YFFB00
QSM	512	M111	1111	1111	110X	0XXX	XXXX	$YFFC00
TPU	512	M111	1111	1111	111X	0XXX	XXXX	$YFFE00

Y = m111, where m is the modmap signal state on the IMB, which reflects the state of the modmap bit in the module configuration register of the system integration module (Y = $7 or $F).

A

APPENDIX B
SYSTEM INTEGRATION MODULE
MEMORY MAP AND REGISTERS

B.1 SIM Register Map

SIM Register Map (Sheet 1 of 2)

FC	ADDRESS	15 8	7 0	
101	YFFA00	MODULE CONFIGURATION (MCR)		
101	YFFA02	MODULE TEST (SIMTR)		TEST
101	YFFA04	CLOCK SYNTHESIZER CONTROL (SYNCR)		CLOCK
101	YFFA06	UNUSED	RESET STATUS REGISTER (RSR)	
101	YFFA08	MODULE TEST E (SIMTRE)		EBI
101	YFFA0A	UNUSED	UNUSED	
101	YFFA0C	UNUSED	UNUSED	
101	YFFA0E	UNUSED	UNUSED	
X01	YFFA10	UNUSED	PORTE DATA (PORTE)	EBI
X01	YFFA12	UNUSED	PORTE DATA (PORTE)	│
X01	YFFA14	UNUSED	PORTE DATA DIRECTION (DDRE)	│
101	YFFA16	UNUSED	PORTE PIN ASSIGNMENT (PEPAR)	│
X01	YFFA18	UNUSED	PORTF DATA (PORTF)	│
X01	YFFA1A	UNUSED	PORTF DATA (PORTF)	│
X01	YFFA1C	UNUSED	PORTF DATA DIRECTION (DDRF)	│
101	YFFA1E	UNUSED	PORTF PIN ASSIGNMENT (PFPAR)	EBI
101	YFFA20	UNUSED	SYSTEM PROTECTION CONTROL (SYPCR)	SYS-PROTECT
101	YYFA22	PERIODIC INTERRUPT CONTROL (PICR)		MOD CONF
101	YFFA24	PERIODIC INTERRUPT TIMING (PITR)		MOD CONF
101	YFFA26	UNUSED	SOFTWARE SERVICE (SWSR)	SYS-PROTECT
101	YFFA28	UNUSED	UNUSED	
101	YFFA30	TEST MODULE MASTER SHIFT A (TSTMSRA)		TEST
101	YFFA32	TEST MODULE MASTER SHIFT B (TSTMSRB)		│
101	YFFA34	TEST MODULE SHIFT COUNT.A.(TSTSCA)	TEST MODULE SHIFT COUNT.B.(TSTSCB)	│
101	YFFA36	TEST MODULE REPETITION COUNTER (TSTRC)		│
101	YFFA38	TEST MODULE CONTROL (CREG)		│
X01	YFFA3A	TEST MODULE DISTRIBUTED REGISTER (DREG)		TEST
	YFFA3C	UNUSED	UNUSED	
	YFFA3E	UNUSED	UNUSED	
X01	YFFA40	UNUSED	PORT C DATA (CSPDR)	CHIP-SELECT
X01	YFFA42	UNUSED	UNUSED	│

FC	ADDRESS	15 8 7 0	
101	YFFA44	CHIP-SELECT PIN ASSIGNMENT (CSPAR0)	CHIP-SELECT
101	YFFA46	CHIP-SELECT PIN ASSIGNMENT (CSPAR1)	
101	YFFA48	CHIP-SELECT BASE BOOT (CSBARBT)	
101	YFFA4A	CHIP-SELECT OPTION BOOT (CSORBT)	
101	YFFA4C	CHIP-SELECT BASE 0 (CSBAR0)	
101	YFFA4E	CHIP-SELECT OPTION 0 (CSOR0)	
101	YFFA50	CHIP-SELECT BASE 1 (CSBAR1)	
101	YFFA52	CHIP-SELECT OPTION 1 (CSOR1)	
101	YFFA54	CHIP-SELECT BASE 2 (CSBAR2)	
101	YFFA56	CHIP-SELECT OPTION 2 (CSOR2)	
101	YFFA58	CHIP-SELECT BASE 3 (CSBAR3)	
101	YFFA5A	CHIP-SELECT OPTION 3 (CSOR3)	
101	YFFA5C	CHIP-SELECT BASE 4 (CSBAR4)	
101	YFFA5E	CHIP-SELECT OPTION 4 (CSOR4)	
101	YFFA60	CHIP-SELECT BASE 5 (CSBAR5)	
101	YFFA62	CHIP-SELECT OPTION 5 (CSOR5)	
101	YFFA64	CHIP-SELECT BASE 6 (CSBAR6)	
101	YFFA66	CHIP-SELECT OPTION 6 (CSOR6)	
101	YFFA68	CHIP-SELECT BASE 7 (CSBAR7)	
101	YFFA6A	CHIP-SELECT OPTION 7 (CSOR7)	
101	YFFA6C	CHIP-SELECT BASE 8 (CSBAR8)	
101	YFFA6E	CHIP-SELECT OPTION 8 (CSOR8)	
101	YFFA70	CHIP-SELECT BASE 9 (CSBAR9)	
101	YFFA72	CHIP-SELECT OPTION 9 (CSOR9)	
101	YFFA72	CHIP-SELECT OPTION 9 (CSOR9)	
101	YYFA74	CHIP-SELECT BASE 10 (CSBAR10)	
101	YFFA76	CHIP SELECT OPTION 10 (CSOR10)	CHIP-SELECT
	YFFA78	UNUSED	UNUSED
	YFFA7A	UNUSED	UNUSED
	YFFA7C	UNUSED	UNUSED
	YFFA7E	UNUSED	UNUSED

X = Depends on state of SUPV bit in SIM MCR

Y = m111, where m is the modmap bit in the SIM MCR (Y = $7 or $F)

B.2 SIM Registers

MCR — Module Control Register   **$YFFA00**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
EXOFF	FRZSW	FRZBM	0	SLVEN	0	SHEN1	SHEN0	SUPV	MM	0	0	IARB3	IARB2	IARB1	IARB0

RESET

0	0	0	0	DB11	0	0	0	1	1	0	0	0	0	0	0

EXOFF — External Clock Off
    1 = The CLKOUT pin is placed in a high-impedance state.
    0 = The CLKOUT pin is driven from an internal clock source.

FRZSW — Freeze Software Enable
    1 = When FREEZE is asserted, the software watchdog and periodic interrupt timer counters are disabled, preventing interrupts from occurring when software is debugged.
    0 = When FREEZE is asserted, the software watchdog and periodic interrupt timer counters continue to run.

FRZBM — 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.

SLVEN — Slave Mode Enabled
    1 = Any external master winning control of the external bus also gains direct access to the internal peripherals.
    0 = The internal peripherals are not available to an external master. This bit is a read-only status bit that reflects the state of DB11 during reset.

SHEN1–0 — Show Cycle Enable
These two control bits determine what the EBI does with the external bus during internal transfer operations.

SHEN1	SHEN0	Action
0	0	Show cycles disabled, external arbitration enabled
0	1	Show cycles enabled, external arbitration disabled
1	0	Show cycles enabled, external arbitration enabled
1	1	Show cycles enabled, external arbitration enabled; internal activity is halted by a bus grant.

B

SUPV — Supervisor/Unrestricted Data Space
> 1 = Registers with access controlled by the SUPV bit are restricted to supervisor access only when FC2 = 1.
> 0 = Registers with access controlled by the SUPV bit are unrestricted (FC2 is a don't care).

MM — Module Mapping
> 1 = Internal modules are addressed from $FFF000–$FFFFFF, which is in the absolute short addressing range.
> 0 = Internal modules are addressed from $7FF000–$7FFFFF.

IARB3–IARB0 — Interrupt Arbitration Bits
> The system software must initialize the IARB field to a value from $F (highest priority) to $1 (lowest priority).

SIMTR — System Integration Module Test Register   **$YYFA02**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
MASK						0	0	SOSEL1	SOSEL0	SHIRQ1	SHIRQ0	FBIT1	FBIT0	BWC1	BWC0

RESET:

0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0

MASK — Revision Number for this Part

SOSEL1–SOSEL0 — Scan-Out Select

SHIRQ1–SHIRQ0 — Show Interrupt Request

FBIT1–FBIT0 — Force Bits

BWC1–BWC0 — Bandwidth Control Bits

B

SYNCR — Clock Synthesizer Control Register   **$YFFA04**

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

RESET:

0	0	1	1	1	1	1	1	0	0	0	U	U	0	0	0

U = Unaffected by reset

$$F_{SYSTEM} = F_{CRYSTAL} [4(Y+1)2^{2W + X}]$$

W — Frequency Control Bit

X — Frequency Control Bit

Y5–Y0 — Frequency Control Bits

EDIV — E-Clock Divide Rate
    1 = E-clock = system clock divided by 16
    0 = E-clock = system clock divided by 8

SLIMP — Limp Mode
    1 = A loss of crystal reference has been detected and the VCO is running at
        approximately half of maximum speed.
    0 = External crystal frequency is VCO reference.

SLOCK — Synthesizer Lock
    1 = VCO has locked on to the desired frequency (or system clock is driven ex-
        ternally).
    0 = VCO is enabled, but has not yet locked.

RSTEN — Reset Enable
    1 = Loss of crystal causes a system reset.
    0 = Loss of crystal causes the VCO to operate at a nominal speed without
        external reference (limp mode), and the MCU continues to operate at that
        speed.

STSIM — Stop Mode System Integration Clock
    1 = When the LPSTOP instruction is executed, the SIM clock is driven from the
        VCO.
    0 = When the LPSTOP instruction is executed, the SIM clock is driven from the
        crystal 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 SIM clock, as determined by the STSIM bit.
    0 = When the LPSTOP instruction is executed, the external clock is held low to
        conserve power.

B

Some System Frequencies from 32.768-kHz Reference

Y	W = 0 X = 0	W = 0 X = 1	W = 1 X = 0	W = 1 X = 1
0 = 000000	131	262	524	1049
1 = 000001	262	524	1049	2097
2 = 000010	393	786	1573	3146
3 = 000011	524	1049	2097	4194
4 = 000100	655	1311	2621	5243
5 = 000101	786	1573	3146	6291
6 = 000110	918	1835	3670	7340
7 = 000111	1049	2097	4194	8389
8 = 001000	1180	2359	4719	9437
9 = 001001	1311	2621	5243	10486
10 = 001010	1442	2884	5767	11534
11 = 001011	1573	3146	6291	12583
12 = 001100	1704	3408	6816	13631
13 = 001101	1835	3670	7340	14680
14 = 001110	1966	3932	7864	15729
15 = 001111	2097	4194	8389	16777
16 = 010000	2228	4456	8913	

RSR — Reset Status Register   **$YFFA07**

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

EXT — External Reset
    1 = The last reset was caused by an external signal.

POW — Power-Up Reset
    1 = The last reset was caused by the power-up reset circuit.

SW — Software Watchdog Reset
    1 = The last reset was caused by the software watchdog circuit.

HLT — Halt Monitor Reset
    1 = The last reset was caused by the system protection submodule halt monitor.

LOC — Loss of Clock Reset
    1 = The last reset was caused by a loss of frequency reference to the clock
        submodule.

SYS — System Reset
    1 = The last reset was caused by the CPU executing a reset instruction.

TST — Test Submodule Reset
    1 = The last reset was caused by the test submodule.

SIMTRE — System Integration Module Test Register (E Clock)   **YFFA08**

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

RESET·

This write-only register is reserved for factory testing. A write to this register in test mode forces the E-clock phase to synchronize with the system clock.

PORTE — Port E Data Register   **$YFFA11, YFFA13**

7	6	5	4	3	2	1	0
PE7	PE6	PE5	PE4	PE3	PE2	PE1	PE0

RESET:
U	U	U	U	U	U	U	U

PE7–0 — Port E Data

DDRE — Port E Data Direction Register   **$YFFA15**

7	6	5	4	3	2	1	0
DDE7	DDE6	DDE5	DDE4	DDE3	DDE2	DDE1	DDE0

RESET:
0	0	0	0	0	0	0	0

DDE7–0 — Data Direction E (Input/Output)
    1 = Input
    0 = Output

B

PEPAR — Port E Pin Assignment Register   **$YFFA17**

7	6	5	4	3	2	1	0
PEPA7 (SIZ1)	PEPA6 (SIZ0)	PEPA5 ($\overline{AS}$)	PEPA4 ($\overline{DS}$)	PEPA3 ($\overline{RMC}$)	PEPA2 ($\overline{AVEC}$)	PEPA1 $\overline{DSACK1}$	PEPA0 $\overline{DSACK0}$

RESET:

DB8	DB8	DB8	DB8	DB8	DB8	DB8	DB8

PEPA7–0 — Port E Pin Assignment (I/O Function)
SIZ1–$\overline{DSACK0}$ — Control Bus Function
A one on DB8 at reset sets the pins to the bus control function; otherwise, they are general-purpose I/O.

PORTF — Port F Data Register   **$YFFA19,YFFA1B**

7	6	5	4	3	2	1	0
PF7	PF6	PF5	PF4	PF3	PF2	PF1	PF0

RESET:

U	U	U	U	U	U	U	U

PF7–0 — Port F Data

DDRF — Port F Data Direction Register   **$YFFA1D**

7	6	5	4	3	2	1	0
DDF7	DDF6	DDF5	DDF4	DDF3	DDF2	DDF1	DDF0

RESET:

0	0	0	0	0	0	0	0

DDF7–0 — Data Direction F (Input/Output)
    1 = Input
    0 = Output

B

PFPAR — Port F Pin Assignment Register   **$YFFA1F**

7	6	5	4	3	2	1	0
PFPA7	PFPA6	PFPA5	PFPA4	PFPA3	PFPA2	PFPA1	PFPA0
($\overline{\text{IRQ7}}$)	($\overline{\text{IRQ6}}$)	($\overline{\text{IRQ5}}$)	($\overline{\text{IRQ4}}$)	($\overline{\text{IRQ3}}$)	($\overline{\text{IRQ2}}$)	($\overline{\text{IRQ1}}$)	(MODCK)

RESET:

DB9	DB9	DB9	DB9	DB9	DB9	DB9	DB9

PFPA7–0 — Port E Pin Assignment (I/O Function)
IRQ7–1 — Control Bus Function
MODCLK
A one on DB9 at reset sets the pins to the bus control function; otherwise, they are general purpose I/O.

SYPCR — System Protection Control Register   **$YFFA21**

7	6	5	4	3	2	1	0
SWE	SWP	SWTI	SWTO	HME	BME	BMT1	BMT0

RESET:

1	$\overline{\text{MODCK}}$	0	0	0	0	0	0

SWE — Software Watchdog Enable
    1 = Software watchdog enabled
    0 = Software watchdog disabled

SWP — Software Watchdog Prescale
    1 = Software watchdog clock prescaled by 512
    0 = Software watchdog clock not prescaled
At reset SWP takes on the inverted value of the MODCLK pin.

SWT1–SWT0 — Software Watchdog Timing
These bits control the divide ratio used to establish the timeout period for the software watchdog timer. The calculation for the software timeout period is given in the following equation.

$$\text{Timeout Period} = 1 / (\text{EXTAL Frequency/Divide Count})$$

or

$$\text{Timeout Period} = \text{Divide Count/EXTAL Frequency}$$

B

Software Timeout Periods for Watchdog Timeout

Bits 6-4	Software Timeout Period	32.768-kHz Crystal Period	16.718-MHz External Clock Period
000	2^9/EXTAL Input Frequency	15.6 Milliseconds	30.6 Microseconds
001	2^{11}/EXTAL Input Frequency	62.5 Milliseconds	122.5 Microseconds
010	2^{13}/EXTAL Input Frequency	250 Milliseconds	490 Microseconds
011	2^{15}/EXTAL Input Frequency	1 Second	1.96 Microseconds
100	2^{18}/EXTAL Input Frequency	8 Seconds	15.6 Milliseconds
101	2^{20}/EXTAL Input Frequency	32 Seconds	62.7 Milliseconds
110	2^{22}/EXTAL Input Frequency	128 Seconds	250 Milliseconds
111	2^{24}/EXTAL Input Frequency	512 Seconds	1 Second

HME — Halt Monitor Enable
    1 = Enable halt monitor function
    0 = Disable halt monitor function

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

BMT — Bus Monitor Timing

Bits 1–0	Bus Monitor Timeout Period
00	64 System Clocks (CLK)
01	32 System Clocks (CLK)
10	16 System Clocks (CLK)
11	8 System Clocks (CLK)

PICR — Periodic Interrupt Control Register   **$YFFA22**

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

PIRQL2–0 — Periodic Interrupt Request Level

Bits 10–8	Interrupt Request Level
000	Periodic Interrupt Disabled
001	Interrupt Request Level 1
010	Interrupt Request Level 2
011	Interrupt Request Level 3
100	Interrupt Request Level 4
101	Interrupt Request Level 5
110	Interrupt Request Level 6
111	Interrupt Request Level 7

PIV7–PIV0 — Periodic Interrupt Vector
    These bits contain the value of the vector generated during an IACK cycle in response to an interrupt from the periodic timer.

PITR — Periodic Interrupt Timing Register $YFFA24

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	PTP	PITR7	PITR6	PITR5	PITR4	PITR3	PITR2	PITR1	PITR0

RESET:

| 0 | 0 | 0 | 0 | 0 | 0 | 0 | $\overline{MODCK}$ | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

PTP — Periodic Timer Prescaler
    1 = Periodic timer clock prescaled by a value of 512
    0 = Periodic timer clock not prescaled

MODCK	PTP
0	1
1	0

PITR7–PITR0 — PITM Field (Periodic Interrupt Timing Modulus)
    The periodic interrupt timing register (PITR) contains the count value for the periodic timer. A zero value turns off the periodic timer.

    The period of the periodic timer can be calculated using the following equation:

$$PIT\ Period = PITM/(EXTAL/Prescaler)/4$$

where

    PIT Period = Periodic Interrupt Timer Period
    PITM = Periodic Interrupt Timer Register Modulus (PITR7–PITR0)

B

EXTAL = Crystal Frequency
Prescaler = 512 or 1 depending on the state of the PTP bit in the PITR

SWSR — Software Service Register $YFFA27

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

The software watchdog service sequence consists of the following two steps:

1. Write $55 to the software service register (SWSR)
2. Write $AA to the SWSR

Both writes must occur in the order shown before the watchdog timeout, but any number of instructions can be executed between the two writes.

TSTMSRA — Master Shift Register A $YFFA30

Master shift register A contains the stimulus to be transferred from the test submodule to the module under test.

TSTMSRB — Master Shift Register B $YFFA32

Master shift register B collects the response data shifted from the module under test to the test submodule.

TSTSCA — Shift Count Register A and Shift Counter A $YFFA34

Shift count register A is an 8-bit shift register that can be accessed by the bus master. Shift counter A is an 8-bit counter that is loaded by shift count register A and is not accessible to the bus master.

TSTSCB — Shift Count Register B and Shift Counter B $YFFA35

Shift count register B is an 8-bit shift register that can be accessed by the bus master. Shift counter B is an 8-bit counter that is loaded by shift count register B and is not accessible to the bus master.

TSTRC — Test Module Repetition Counter $YFFA36

The reps counter determines the number of psuedo-random vectors generated in the automatic mode of operation.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
BUSY	TMARM	COMP	IMBTST	CPUTR	QBIT	MUXEL	—	—	—	—	ACUT	SCONT	SSHOP	SATO	ETM

RESET:

| 1 | TSTME | U | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

ETM — Enter Test Mode
   1 = Enter test mode
   0 = Stay in normal mode

SATO — Start Automatic Test Operation
   1 = Start an automatic test operation
   0 = Stay in normal mode

SSHOP — Start Shifting Operation
   1 = Start a shifting operation
   0 = Stay in normal mode

SCONT — Start Continuous Operation
   1 = Start continuous operation
   0 = Stop continuous operation

ACUT — Activate Circuit Under Test
   1 = Assert the ACUTL line
   0 = Stay in normal mode

MUXSEL — Multiplexer Select Bit
   1 = Shift in source for master shift register B (MSRB) is the external interrupt pin
   0 = Shift in source for MSRB is the internal test line

QBIT — Quotient Bit
   1 = The least significant bit of master shift register B is available at the quotient/freeze (FREEZE/QUOT) pin.
   0 = The internal freeze status is available at the FREEZE/QUOT pin.

CPUTR — CPU Test Register
   1 = Scan lines connected to the CPU test register
   0 = Scan lines disconnected from the CPU test register

IMBTST — Intermodule Bus Test
   1 = Internal interconnect lines are configured as test lines.
   0 = Internal interconnect lines have normal function.

COMP — Compare Status Bit
    1 = Master shift register B contains the correct answer for the user basic self-test.
    0 = Master shift register B does not contain the correct answer for the basic user self-test.

TMARM — Test Mode Armed Status Bit
    1 = TSTME pin is asserted; test mode can be entered by setting the ETM control bit.
    0 = TSTME pin is negated; test mode cannot be entered. This status bit can be read at any time, but cannot be written.

BUSY — Test Submodule Busy Status Bit
    1 = Test submodule is busy.
    0 = Test submodule is not busy.

DREG — Distributed Register   **$YFFA3A**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
—	—	—	—	—	WAIT3	WAIT2	WAIT1	MSRA18	MSRA17	MSRA16	MSRAC	MSRB18	MSRB17	MSRB16	MSRBC

RESET:

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

WAIT3–WAIT1 — Wait Counter Preset 3–1
    These bits program the delay time between automatic test sequences.

MSRA18–MSRA16 — Master Shift Register A Bits 18–16

MSRAC — Master Shift Register A Configuration
    1 = Master shift register A configured as a 19-bit serial pattern generator.
    0 = Master shift register A configured as a 16-bit shift register.

MSRB18–MSRB16 — Master Shift Register B Bits 18–16

MSRBC — Master Shift Register B Configuration
    1 = Master shift register B configured as a 19-bit serial signature analyzer.
    0 = Master shift register B configured as a 16-bit shift register.

CSPDR — Chip-Select Pin Data Register   **$YFFA41**

7	6	5	4	3	2	1	0
0	DO6	DO5	DO4	DO3	DO2	DO1	DO0

RESET:

0	0	0	0	0	0	0	0

D6–0 — Pin Data

CSPAR0 — Chip-Select Pin Assignment Register 0   **$YFFA44**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	CS5	(FC2)	CS4	(FC1)	CS3	(FC0)	CS2 (BGACK)		CS1	(BG)	SWSR1		SWSR0	

RESET:

0	0	DB2	1	DB2	1	DB2	1	DB1	1	DB1	1	DB1	1	1	DB0

Bits 15, 14 — Not Used
These bits always read zero; write has no effect.

CSPAR1 — Chip-Select Pin Assignment Register 1   **$YFFA46**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	CS10 (A23)		CS9 (A22)		CS8 (A21)		CS7 (A20)		CS6 (A19)	

RESET:

0	0	0	0	0	0	DB7	1	DB6	1	DB5	1	DB4	1	DB3	1

Bits 15–10 — Not Used
These bits always read zero; write has no effect.

Hierarchical Selection Structure of CSPAR1

DATA BUS PINS AT RESET					DEFAULT /ALTERNATE FUNCTION				
DB7	DB6	DB5	DB4	DB3	CS10/A23	CS9/A22	CS8/A21	CS7/A20	CS6/A19
1	1	1	1	1	CS10	CS9	CS8	CS7	CS6
1	1	1	1	0	CS10	CS9	CS8	CS7	A19
1	1	1	0	X	CS10	CS9	CS8	A20	A19
1	1	0	X	X	CS10	CS9	A21	A20	A19
1	0	X	X	X	CS10	A22	A21	A20	A19
0	X	X	X	X	A23	A22	A21	A20	A19

Pin Assignment Register Bit Encoding

Bits	Description
00	Discrete Output (E Clock on A23)*
01	Alternate Function
10	Chip Select (8-Bit Port)
11	Chip Select (16-Bit Port)

*Except for BR, BG, and BGACK

B

CSBARBT — Chip-Select Base Address Register Boot $\hspace{2cm}$ **$YFFA48**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
A23	A22	A21	A20	A19	A18	A17	A16	A15	A14	A13	A12	A11	BLKSZ		

RESET:

0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1

CSBAR0-CSBAR10 — Chip-Select Base Address Registers $\hspace{1cm}$ **$YFFA4C–YFFA74**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
A23	A22	A21	A20	A19	A18	A17	A16	A15	A14	A13	A12	A11	BLKSZ		

RESET:

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

BLKSZ — Block Size Field

Block Size Field	Block Size	Address Lines Compared
000	2K	A23–A11
001	8K	A23–A13
010	16K	A23–A14
011	64K	A23–A16
100	128K	A23–A17
101	256K	A23–A18
110	512K	A23–A19
111	1M	A23–A20

Bits 15–3 — Base Address Field

In supervisor/user space, this field sets the starting address of a particular address space.

CSORBT — Chip-Select Option Register $\hspace{4cm}$ **$YFFA4A**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
MODE	BYTE		R/W̄	STRB	DSACK				SPACE		IPL			AVEC̄

RESET:

0	1	1	1	1	0	1	1	0	1	1	1	0	0	0	0

B

Option Register Functions Summary

MODE	BYTE	R/W	STRB	$\overline{\text{DSACK}}$	SPACE	IPL	$\overline{\text{AVEC}}$
0 = ASYNC	00 = Off	00 = Rsvd	0 = AS	0000 = 0 WAIT	00 = CPU SP	000 = All	0 = Off
1 = SYNC	01 = Lower	01 = Read	1 = DS	0001 = 1 WAIT	01 = User SP	001 = Level 1	1 = On
	10 = Upper	10 = Write		0010 = 2 WAIT	10 = Supv SP	010 = Level 2	
	11 = Both	11 = Both		0011 = 3 WAIT	11 = S/U SP	011 = Level 3	
				0100 = 4 WAIT		100 = Level 4	
				0101 = 5 WAIT		101 = Level 5	
				0110 = 6 WAIT		110 = Level 6	
				0111 = 7 WAIT		111 = Level 7	
				1000 = 8 WAIT			
				1001 = 9 WAIT			
				1010 = 10 WAIT			
				1011 = 11 WAIT			
				1100 = 12 WAIT			
				1101 = 13 WAIT			
				1110 = F term			
				1111 = External			

CSOR0–CSOR10 — Chip-Select Option Registers   **$YFFA4E-YFFA76**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
MODE	BYTE		R/$\overline{\text{W}}$		STRB	DSACK				SPACE		IPL			$\overline{\text{AVEC}}$

RESET:

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

MODE — Asynchronous/Synchronous Mode
    1 = Synchronous mode selected
    0 = Asynchronous mode selected

BYTE — Upper/Lower Byte Option
    This field is used when the chip-select 16-bit port option in the pin assignment register is selected.

Bits	Description
00	Disable
01	Lower Byte
10	Upper Byte
11	Both Bytes

B

R/$\overline{\text{W}}$ — Read/Write

This option causes the chip select to be asserted only for a read, only for a write, or for both read and write.

Bits	Description
00	Reserved
01	Read Only
10	Write Only
11	Read/Write

STRB — Address Strobe/Data Strobe

    1 = Data strobe
    0 = Address strobe

This option controls the timing for assertion of a chip select in asynchronous mode.

$\overline{\text{DSACK}}$ — Data Strobe Acknowledge

In asynchronous mode, this option field specifies the source of the $\overline{\text{DSACK}}$ (externally or internally generated).

Bits	Description
0000	No Wait States
0001	1 Wait State
0010	2 Wait States
0011	3 Wait States
0100	4 Wait States
0101	5 Wait States
0110	6 Wait States
0111	7 Wait States
1000	8 Wait States
1001	9 Wait States
1010	10 Wait States
1011	11 Wait States
1100	12 Wait States
1101	13 Wait States
1110	Fast Termination
1111	External $\overline{\text{DSACK}}$

SPACE — Address Space

This option field checks the address spaces indicated by the function codes generated by the CPU.

Bits	Description
00	CPU Space
01	User Space
10	Supervisor Space
11	Supervisor/User Space

IPL — Interrupt Priority Level

In an IACK cycle, the chip-select logic checks the acknowledged interrupt level on address lines A3–A1. If that level matches the level set in the IPL field, the chip select can be asserted if the match conditions in the other fields are met.

Bits	Description
000	Any Level
001	IPL1
010	IPL2
011	IPL3
100	IPL4
101	IPL5
110	IPL6
111	IPL7

$\overline{\text{AVEC}}$ — Autovector Enable

1 = Autovector enabled
0 = External interrupt vector enabled

B

B

APPENDIX C
QUEUED SERIAL MODULE
MEMORY MAP AND REGISTERS

C.1 QSM Memory Map

	15 8 7 0	
$YFFC00	QMCR	
$YFFC02	QTEST	SUPERVISOR-ONLY DATA SPACE
$YFFC04	QILR | QIVR	
$YFFC06	RESERVED	
$YFFC08	SCCR0	
$YFFC0A	SCCR1	
$YFFC0C	SCSR	
$YFFC0E	SCDR	
$YFFC10	RESERVED	
$YFFC12	RESERVED	
$YFFC14	RESERVED | QPDR	
$YFFC16	QPAR | QDDR	ASSIGNABLE DATA SPACE
$YFFC18	SPCR0	(SUPERVISOR-ONLY OR UNRESTRICTED)
$YFFC1A	SPCR1	
$YFFC1C	SPCR2	
$YFFC1E	SPCR3 | SPSR	
$YFFC20-FF	RESERVED	
$YFFD00-1F	REC.RAM	
$YFFD20-3F	TRAN.RAM	QUEUE RAM
$YFFD40-4F	COMD .RAM	

Y = m111 where m is the modmap bit in the SIM MCR (Y = $7 or $F).

C

C.2 QSM Registers

QMCR — QSM Configuration Register **$YFFC00**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
STOP	FRZ1	FRZ0	0	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

STOP — Stop Enable
   1 = QSM clock operation stopped
   0 = Normal QSM clock operation

FRZ1 — Freeze 1
   1 = Halt the QSM (on a transfer boundary)
   0 = Ignore the FREEZE signal on the IMB

FRZ0 — Freeze 0
   Reserved for future enhancement.

Bits 12–8 — Not Implemented

SUPV — Supervisor/Unrestricted
   1 = Supervisor access
   0 = User access

Bits 6–4 — Not Implemented

IARB — Interrupt Arbitration Identification Number
   System software should initialize the IARB field to a value between $F (top priority) and $1 (lowest priority). Otherwise, any interrupts generated are identified by the CPU as spurious.

QTEST — QSM Test Register **$YFFC02**

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	TSBD	SYNC	TQSM	TMM

RESET:

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

TSBD — SPI Test Scan Path Select
    1 = Enable delay to SCK scan path
    0 = Enable SPI baud clock scan path

SYNC — SCI Baud Clock Synchronization Signal
    1 = Inhibit SCI source signal (QCSCI1)
    0 = Activate SCI source signal

TQSM — QSM Test Enable
    1 = Enable QSM to send test scan paths
    0 = Disable scan path

TMM — Test Memory Map
    1 = QSM responds to test memory addresses.
    0 = QSM responds to QSM memory addresses.

QILR — QSM Interrupt Level Register   **$YFFC04**

15	14	13	12	11	10	9	8			0
0	0	ILQSPI			ILSCI			XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX		

RESET:

0	0	0	0	0	0	0	0

ILQSPI — Interrupt Level for QSPI
    ILQSPI determines the priority level of all QSPI interrupts. Program this field to a value between $0 (interrupts disabled) to $7 (highest priority).

ILSCI — Interrupt Level of SCI
    LSCI determines the priority level of all SCI interrupts. Program this field to a value between $0 (interrupts disabled) to $7 (highest priority).

QIVR — QSM Interrupt Vector Register   **$YFFC05**

15		7	6	5	4	3	2	1	0
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX		INTV							

RESET:

		0	0	0	0	1	1	1	1

INTV — Interrupt Vector

C

SCCR0 — SCI Control Register 0   **$YFFC08**

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

RESET:

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	1	0	0

Bits 15–13 — Not Implemented

SCBR — Baud Rate
  The SCI baud rate is programmed by writing a 13-bit value to SCBR.

$$SCI\ Baud = System\ Clock/(32 \bullet SCBR)$$

  where SCBR equals {1, 2, 3, . . . 8191}.

SCCR1 — SCI Control Register 1   **$YFFC0A**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	LOOPS	WOMS	ILT	PT	PE	M	WAKE	TIE	TCIE	RIE	ILIE	TE	RE	RWU	SBK

RESET:

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

Bit 15 — Not Implemented

LOOPS — LOOP Mode
  LOOPS controls a feedback path on the data serial shifter.
    1 = Test SCI operation, looping, feedback path enabled
    0 = Normal SCI operation, no looping, feedback path disabled

WOMS — Wired-OR Mode for SCI Pins
    1 = If configured as an output, TXD is an open-drain output.
    0 = If configured as an output, TXD is a normal CMOS output.

ILT — Idle-Line Detect Type
    1 = Long idle-line detect (starts counting when the first one is received after a
        stop bit(s))
    0 = Short idle-line detect (starts counting when the first one is received)

PT — Parity Type
    1 = Odd parity
    0 = Even parity

PE — Parity Enable
    1 = SCI parity enabled
    0 = SCI parity disabled

M	PE	Result
0	0	8 Data Bits
0	1	7 Data Bits, 1 Parity Bit
1	0	9 Data Bits
1	1	8 Data Bits, 1 Parity Bit

M — Mode Select
    1 = SCI frame: 1 start bit, 9 data bits, 1 stop bit (11 bits total)
    0 = SCI frame: 1 start bit, 8 data bits, 1 stop bit (10 bits total)

WAKE — Wakeup by Address Mark
    1 = SCI receiver awakened by address mark (eighth or ninth (last) bit set)
    0 = SCI receiver awakened by idle-line detection

TIE — Transmit Interrupt Enable
    1 = SCI TDRE interrupts enabled
    0 = SCI TDRE interrupts inhibited

TCIE — Transmit Complete Interrupt Enable
    1 = SCI TC interrupts enabled
    0 = SCI TC interrupts inhibited

RIE — Receiver Interrupt Enable
    1 = SCI RDRF interrupts enabled
    0 = SCI RDRF interrupts inhibited

ILIE — Idle-Line Interrupt Enable
    1 = SCI IDLE interrupts enabled
    0 = SCI IDLE interrupts inhibited

TE — Transmitter Enable
    1 = SCI transmitter enabled, TXD pin dedicated to the SCI transmitter
    0 = SCI transmitter disabled; TXD pin can be used as general-purpose I/O.

RE — Receiver Enable
    1 = SCI receiver enabled
    0 = SCI receiver disabled

C

RWU — Receiver Wakeup
    1 = Wakeup mode enabled, all received data ignored until awakened
    0 = Normal receiver operation, all received data recognized

SBK — Send Break
    1 = Break frame(s) are transmitted after completion of the current frame.
    0 = Normal operation

SCSR — SCI Status Register   **$YFFC0C**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	TDRE	TC	RDRF	RAF	IDLE	OR	NF	FE	PF

RESET:

0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0

Bits 15–9 — Not Implemented

TDRE — Transmit Data Register Empty Flag
    1 = A new character can now be written to register TDR.
    0 = Register TDR still contains data to be sent to the transmit serial shifter.

TC — Transmit Complete Flag
    1 = SCI transmitter is idle.
    0 = SCI transmitter is busy.

RDRF — Receive Data Register Full Flag
    1 = Register RDR contains new data.
    0 = Register RDR is empty or contains previously read data.

RAF — Receiver Active Flag
    1 = SCI receiver is busy.
    0 = SCI receiver is idle.

IDLE — Idle-Line Detected Flag
    1 = SCI receiver detected an idle-line condition.
    0 = SCI receiver did not detect an idle-line condition.

OR — Overrun Error Flag
    1 = RDRF is not cleared before new data arrives.
    0 = RDRF is cleared before new data arrives.

NF — Noise Error Flag
  1 = Noise occurred on the received data.
  0 = No noise detected on the received data.

FE — Framing Error Flag
  1 = Framing error or break occurred on the received data.
  0 = No framing error on the received data.

PF — Parity Error Flag
  1 = Parity error occurred on the received data.
  0 = No parity error on the received data.

SCDR — SCI Data Register   **$YFFC0E**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	R8/T8	R7/T7	R6/T6	R5/T5	R4/T4	R3/T3	R2/T2	R1/T1	R0/T0

RESET:

0	0	0	0	0	0	0	U	U	U	U	U	U	U	U	U

R8/T8 — Receive 8/Transmit 8
R0–R7/T0–T7 — Receive 0–7/Transmit 0–7

QPDR — QSM Port Data Register   **$YFFC15**

15							7	6	5	4	3	2	1	0
RESERVED							D7 (TXD)	D6 (PCS3)	D5 (PCS2)	D4 (PCS1)	D3 (PSCO/$\overline{SS}$)	D2 (SCK)	D1 (MOSI)	D0 (MISO)

RESET:

							0	0	0	0	0	0	0	0

D7–0 — Pin Data
TXD-MIS0 — Pin Function

QPAR — QSM Pin Assignment Register   **$YFFC16**

15	14	13	12	11	10	9	8		0
0	PCS3	PCS2	PCS1	PCSO/$\overline{SS}$	0	MOSI	MISO	XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX	

RESET:

0	0	0	0	0	0	0	0

0 = General-purpose I/O
1 = QSPI module

Bit 15 — Not Implemented

PCS3–PCS1 — Peripheral Chip Selects 3–1

PCS0/SS — Peripheral Chip Select 0/Slave Select

Bit 10 — Not Implemented

MOSI — Master Out Slave In

MISO — Master In Slave Out
These bits determine whether the associated QSM port pin functions as a general-purpose I/O pin or is assigned to the QSPI submodule.

QDDR — QSM Data Direction Register $YFFC17

15				7	6	5	4	3	2	1	0
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX				TXD	PCS3	PCS2	PCS1	PCS0/$\overline{SS}$	SCK	MOSI	MISO

RESET:

				0	0	0	0	0	0	0	0

    0 = Input
    1 = Output

TXD — Transmit Data
PCS1 — Peripheral Chip Selects 3–1
PSC0/SS — Peripheral Chip Select 0/Slave Select
SCK — Serial Clock
MOSI — Master Out Slave In

SPCR0 — QSPI Control Register 0 $YFFC18

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
MSTR	WOMQ		BITS			CPOL	CPHA				SPBR				

RESET:

0	0	0	0	0	0	0	1	0	0	0	0	0	1	0	0

MSTR — Master/Slave Mode Select
    1 = QSPI is system master and can initiate transmission to external SPI devices.
    0 = QSPI is a slave device, and only responds to externally generated serial MSTR.

C

WOMQ — Wired-OR Mode for QSPI Pins

 1 = All QSPI port pins designated as output by QDDR function as open-drain outputs.

 0 = Output pins have normal outputs instead of open-drain outputs.

BITS — Bits Per Transfer

 In master mode, BITS determines the number of data bits transferred for each serial transfer in the queue.

Bit 13	Bit 12	Bit 11	Bit 10	Bits per Transfer
0	0	0	0	16
0	0	0	1	Reserved
0	0	1	0	Reserved
0	0	1	1	Reserved
0	1	0	0	Reserved
0	1	0	1	Reserved
0	1	1	0	Reserved
0	1	1	1	Reserved
1	0	0	0	8
1	0	0	1	9
1	0	1	0	10
1	0	1	1	11
1	1	0	0	12
1	1	0	1	13
1	1	1	0	14
1	1	1	1	15

CPOL — Clock Polarity

 1 = The inactive state value of SCK is high.

 0 = The inactive state value of SCK is low.

CPHA — Clock Phase

 1 = Data is changed on the leading edge of SCK and captured on the following edge of SCK.

 0 = Data is captured on the leading edge of SCK and changed on the following edge of SCK.

C

SPBR — Serial Clock Baud Rate

The QSPI internally generates the baud rate for SCK, the frequency of which is programmable by the user. The following equation determines the SCK baud rate:

$$\text{SCK Baud Rate} = \text{System Clock}/(2 \bullet \text{SPBR})$$

or

$$\text{SPBR} = \text{System Clock}/(2 \bullet \text{SCK Baud Rate Desired})$$

where SPBR equals {2, 3, 4, . . . , 255}.

SPCR1— QSPI Control Register   **$YFFC1A**

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

RESET:

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	1	0	0

SPE — QSPI Enable

1 = The QSPI is enabled and the pins allocated by QSM register QPAR are controlled by the QSPI.

0 = The QSPI is disabled and the seven QSPI pins can be used as general-purpose I/O pins, regardless of the values in QPAR.

DSCKL — Delay before SCK

This bit determines the length of time the QSPI delays from peripheral chip select (PCS) valid to SCK transition for serial transfers in which the command control bit, DSCK of the QSPI RAM, equals one.

$$\text{PCS to SCK Delay} = [\text{DSCKL/System Clock Frequency}]$$

where DSCKL equals {1,2,3, . . . 127}.

DTL — Length of Delay after Transfer

These bits determine the length of time that the QSPI delays after each serial transfer in which the command control bit, DT of the QSPI RAM, equals one.

$$\text{Delay after Transfer} = [(32 \bullet \text{DTL})/\text{System Clock Frequency}]$$

where DTL equals {1,2,3, . . . 255}.

SPCR2 — QSPI Control Register 2   **$YFFC1C**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
SPIFIE	WREN	WRTO	0		ENDQP			0	0	0	0		NEWQP		

RESET

| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

SPIFIE — SPI Finished Interrupt Enable
    1 = QSPI interrupts enabled
    0 = QSPI interrupts disabled

WREN — Wrap Enable
    1 = Wraparound mode enabled
    0 = Wraparound mode disabled

WRTO — Wrap To
    1 = Wrap to address found in NEWQP
    0 = Wrap to address $0

Bit 12 — Not Implemented

ENDQP — Ending Queue Pointer
    This field determines the last absolute address in the queue to be completed by the
    QSPI.

Bits 7–4 — Not Implemented

NEWQP — New Queue Pointer Value
    NEWQP determines which queue entry the QSPI transfers first.

SPCR3 — QSPI Control Register   **$YFFC1E**

15	14	13	12	11	10	9	8	0
0	0	0	0	0	LOOPQ	HMIE	HALT	XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

RESET:

| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

Bits 15–11 — Not Implemented

LOOPQ — QSPI Loop Mode
    1 = Feedback path enabled
    0 = Feedback path disabled

C

HMIE — HALTA and MODF Interrupt Enable
    1 = HALTA and MODF interrupts enabled
    0 = HALTA and MODF interrupts disabled

HALT — Halt
    1 = Halt enabled
    0 = Halt not enabled
  This bit is used by the CPU to stop the QSPI on a queue boundary.

SPSR — QSPI Status Register   **$YFFC1F**

15				7	6	5	4	3	2	1	0
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX				SPIF	MODF	HALTA	0		CPTQP		

RESET:

				0	0	0	0	0	0	0	0

SPIF — QSPI Finished Flag
    1 = QSPI finished
    0 = QSPI not finished
  SPIF is set when the QSPI finishes executing the last command determined by the address contained in ENDQP in SPCR2.

MODF — Mode Fault Flag
    1 = Another SPI node requested to become the network SPI master while the QSPI was enabled in master mode (MSTR = 1), or the PCS0/SS pin was incorrectly pulled low by external hardware.
    0 = Normal operation

HALTA — Halt Acknowledge Flag
    1 = QSPI halted
    0 = QSPI not halted
  HALTA is asserted by the QSPI when it has come to an orderly halt at the request of the CPU, through the assertion of HALT.

Bit 4 — Not Implemented

CPTQP — Completed Queue Pointer
  CPTQP contains the queue pointer value of the last command in the queue that was completed.

COMD.RAM — Command Ram   **$YFFD40**

7	6	5	4	3	2	1	0
CONT	BITSE	DT	DSCK	PCS3	PCS2	PCS1	PCS0*
—	—	—	—	—	—	—	—
CONT	BITSE	DT	DSCK	PCS3	PCS2	PCS1	PCS0*

                                                                                                                    $YFFD4F

             COMMAND CONTROL | PERIPHERAL CHIP SELECT

*The PCS0 bit represents the dual-function PCS0/$\overline{SS}$.

CONT — Continue
    1 = Keeps peripheral chip selects asserted after transfer is complete.
    0 = Returns control of peripheral chip selects to QPDR after transfer is complete.

BITSE — Bits Enable
    1 = Number of bits to transfer defined in BITS field of SPCR0.
    0 = 8 bits to transfer

DT — Delay After Transfer
    1 = Delay
    0 = No delay

DSCK — PCS to SCK Delay
    1 = DSCKL field in SPCR1 specifies value of delay from PCS valid to SCK
    0 = PCS valid to SCK transition is 1/2 SCK

PCS3–PCS0/$\overline{SS}$ — Peripheral Chip Select
    The four peripheral chip-select bits can be used directly to select one of four external
    chips for the serial transfer, or decoded by external hardware to select one of 16
    chip-select patterns for a serial transfer.

C

C

APPENDIX D
STANDBY RAM —
PROGRAMMER'S MODEL AND REGISTERS

D.1 RAM Module Programmer's Model

Control Registers

A23					A0	15	8	7	0	
Y	1111	1111	1011	1100	000X	MODULE CONFIGURATION REGISTER (RAMMCR)				$YFFB00
Y	1111	1111	1011	1100	001X	TEST REGISTER (RAMTST)				$YFFB02
Y	1111	1111	1011	1100	010X	BASE ADDRESS/STATUS REGISTER (RAMBAR) 0 0 RAMDS				$YFFB04
Y	1111	1111	1011	1100	011X	NOT IMPLEMENTED				
						•	•		•	
						•	•		•	
						•	•		•	
Y	1111	1111	1111	1100	111X	NOT IMPLEMENTED				

Ram Array

A23	A11		A0	15	8	7	0
*——	—*000	0000	000X				
*——	—*000	0000	001X				
*——	—*000	0000	010X				
*——	—*000	0000	011X				
		•		•		•	
		•		•		•	
*——	—*000	0000	111X				

NOTE: All address locations are given in binary.
* — Base address defined in array base address register
X — Zero or one depending on byte/word address
Y — m111 where m is the modmap bit in the SIM MCR (Y = $7 or $F)

D

D.2 Standby Ram Registers

RAMMCR — RAM Module Configuration Register   **$YFFB00**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
STOP			NOT USED				RASP				NOT USED				

RESET:

0				U			1					U			

STOP — Stop Control
    0 = 2K-byte RAM array operates normally.
    1 = 2K-byte RAM array enters low power stop mode.

RASP — RAM Array Space
    0 = 2K-byte RAM array is placed in unrestricted space.
    1 = 2K-byte RAM array is placed in supervisor space.

RAMTST — RAM Test Register   **$YFFBO2**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
SDTEST							RTBA				NOT USED				

RESET:

			0				0					0			

SDTEST — Soft Defect Test Function
    SDTEST is used to select a particular RAM array test when the system is in test mode. Initiate tests by writing a value to SDTEST.The result is either output on data bit zero (D0) of the intermodule bus (IMB) or is determined by checking the array itself.

RTBA — Base Address Register R/W
    0 = RAMBAR may only be written once.
    1 = RAMBAR may be written as desired.
    RTBA controls the "write-once" lock on the array base address and status register (RAMBAR).

RAMBAR — RAM Base Address and Status Register   **$YFFB04**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
A23	A22	A21	A20	A19	A18	A17	A16	A15	A14	A13	A12	A11	NOT USED		RAMDS

RESET:

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	1

Bits 15–3 — RAM Array Base Address

These bits specify address lines A23–A11 of the base address of the RAM array when enabled.

RAMDS — RAM Array Disabled

0 = RAM array is enabled.
1 = RAM array is disabled.

D

D

APPENDIX E
CPU32
PROGRAMMING MODEL AND INSTRUCTION SUMMARY

E.1 User Programming Model

```
31              16 15      8  7        0
┌──────────────────┬──────────┬──────────┐  D0
├──────────────────┼──────────┼──────────┤  D1
├──────────────────┼──────────┼──────────┤  D2
├──────────────────┼──────────┼──────────┤  D3    Data Registers
├──────────────────┼──────────┼──────────┤  D4
├──────────────────┼──────────┼──────────┤  D5
├──────────────────┼──────────┼──────────┤  D6
└──────────────────┴──────────┴──────────┘  D7

31              16 15                    0
┌──────────────────┬──────────────────────┐  A0
├──────────────────┼──────────────────────┤  A1
├──────────────────┼──────────────────────┤  A2
├──────────────────┼──────────────────────┤  A3    Address Registers
├──────────────────┼──────────────────────┤  A4
├──────────────────┼──────────────────────┤  A5
└──────────────────┴──────────────────────┘  A6

31              16 15                    0
┌──────────────────┬──────────────────────┐  A7 (USP)   User Stack Pointer

31                                       0
┌─────────────────────────────────────────┐  PC    Program Counter

                              7          0
                         ┌──────────┐  CCR    Condition Code Register
```

E

E.2 Supervisor Programming Model Supplement

```
    31                      16 15                    0
   +-----------------------+------------------------+  A7' (SSP)  Supervisor Stack Pointer
   |                       |                        |
   +-----------------------+------------------------+
                             15       8 7          0
                            +----------+-----------+    SR       Status Register
                            |          |  (CCR)    |
                            +----------+-----------+
    31                                             0
   +--------------------------------------------+     VBR      Vector Base Register
   |                                            |
   +--------------------------------------------+
                                       2       0
                                      +--------+        SFC      Alternate Function
                                      |        |        DFC      Code Registers
                                      +--------+
```
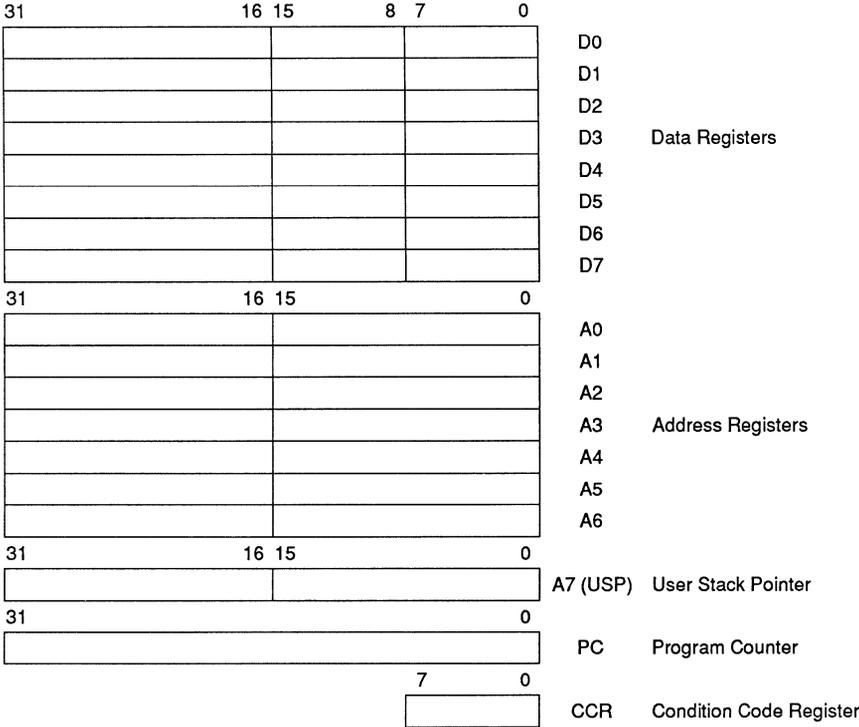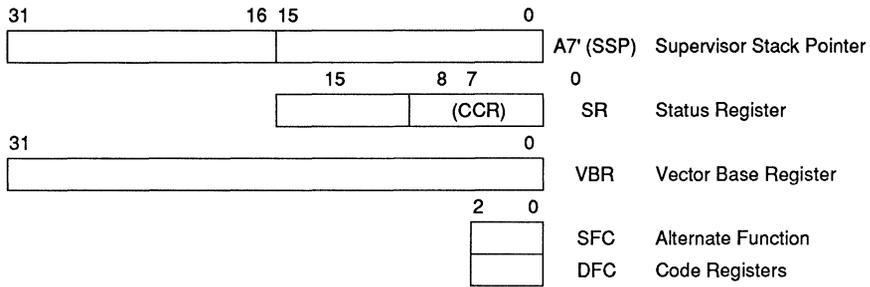
E.3 Status Register

SR — Status Register

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
T1	T0	S	0	0	I2	I1	I0	0	0	0	X	N	Z	V	C

System Byte
  T1–T0 — Trace Enable

  S — Supervisor/User State

  Bits 12–11 — Unimplemented

  I2–I0 — Interrupt Priority Mask

User Byte (Condition Code Register)
  Bits 7–5 — Unimplemented

  X — Extend

  N — Negative

  Z — Zero

  V — Overflow

  C — Carry

E

E.4 Instruction Set Summary

Mnemonic	Description
ABCD	Add Decimal with Extend
ADD	Add
ADDA	Add Address
ADDI	Add Immediate
ADDQ	Add Quick
ADDX	Add with Extend
AND	Logical AND
ANDI	Logical AND Immediate
ASL, ASR	Arithmetic Shift Left and Right
Bcc	Branch Conditionally
BCHG	Test Bit and Change
BCLR	Test Bit and Clear
BGND	Background
BKPT	Breakpoint
BRA	Branch
BSET	Test Bit and Set
BSR	Branch to Subroutine
BTST	Test Bit
CHK, CHK2	Check Register Against Upper and Lower Bounds
CLR	Clear
CMP	Compare
CMPA	Compare Address
CMPI	Compare Immediate
CMPM	Compare Memory to Memory
CMP2	Compare Register Against Upper and Lower Bounds
DBcc	Test Condition, Decrement and Branch
DIVS, DIVSL	Signed Divide
DIVU, DIVUL	Unsigned Divide
EOR	Logical Exclusive OR
EORI	Logical Exclusive OR Immediate
EXG	Exchange Registers
EXT, EXTB	Sign Extend
LEA	Load Effective Address
LINK	Link and Allocate
LPSTOP	Low-Power Stop
LSL, LSR	Logical Shift Left and Right
ILLEGAL	Take Illegal Instruction Trap
JMP	Jump
JSR	Jump to Subroutine

Mnemonic	Description
MOVE	Move
MOVE CCR	Move Condition Code Register
MOVE SR	Move Status Register
MOVE USP	Move User Stack Pointer
MOVEA	Move Address
MOVEC	Move Control Register
MOVEM	Move Multiple Registers
MOVEP	Move Peripheral
MOVEQ	Move Quick
MOVES	Move Alternate Address Space
MULS, MULS.L	Signed Multiply
MULU, MULU.L	Unsigned Multiply
NBCD	Negate Decimal with Extend
NEG	Negate
NEGX	Negate with Extend
NOP	No Operation
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 De-allocate
RTE	Return from Exception
RTR	Return and Restore Codes
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 Register Words
TBLS,TBLSN	Table Lookup and Interpolate (Signed)
TBLU, TBLUN	Table Lookup and Interpolate (Unsigned)
TAS	Test Operand and Set
TRAP	Trap
TRAPcc	Trap Conditionally
TRAPV	Trap on Overflow
TST	Test Operand
UNLK	Unlink

E

E.5 Background Mode Command Summary

Command	Mnemonic	Description
Read D/A Register	RDREG/RAREG	Read the selected address or data register and return the results through the serial interface.
Write D/A Register	WDREG/WAREG	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 background mode.
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 source function code register (SFC) determines the address space accessed.
Write Memory Location	WRITE	Write the operand data to the memory location specified by the long-word address. The destination function code (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 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. Initially, a WRITE is executed to set up the starting address of the block and supply the first operand. The FILL command writes subsequent operands.
Resume Execution	GO	The pipe is flushed and refilled before resuming instruction execution at the current PC.
Patch User Code	CALL	Current program counter is stacked at the location of the current stack pointer. 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 can be used as a null command.

E

APPENDIX F
TIME PROCESSOR UNIT
MEMORY MAP AND REGISTERS

F.1 Register Map

	ADDRESS	15 BYTE n 8 | 7 BYTE n + 1 0
S	$YFFE00	MODULE CONFIGURATION REGISTER
S	$YFFE02	CONFIGURATION REGISTER
S	$YFFE04	DEVELOPMENT SUPPORT CONTROL REGISTER
S	$YFFE06	DEVELOPMENT SUPPORT STATUS REGISTER
S	$YFFE08	INTERRUPT CONFIGURATION REGISTER
S	$YFFE0A	INTERRUPT ENABLE REGISTER
S	$YFFE0C	CHANNEL FUNCTION SELECT REGISTER 0
S	$YFFE0E	CHANNEL FUNCTION SELECT REGISTER 1
S	$YFFE10	CHANNEL FUNCTION SELECT REGISTER 2
S	$YFFE12	CHANNEL FUNCTION SELECT REGISTER 3
X	$YFFE14	HOST SEQUENCE REGISTER 0
X	$YFFE16	HOST SEQUENCE REGISTER 1
X	$YFFE18	HOST SERVICE REQUEST REGISTER 0
X	$YFFE1A	HOST SERVICE REQUEST REGISTER 1
S	$YFFE1C	CHANNEL PRIORITY REGISTER 0
S	$YFFE1E	CHANNEL PRIORITY REGISTER 1
S	$YFFE20	INTERRUPT STATUS REGISTER
S	$YFFE22	LINK REGISTER
S	$YFFE24	SERVICE GRANT LATCH REGISTER
S	$YFFE26	DECODED CHANNEL NUMBER REGISTER

S = Supervisor accessible only
X = Assignable as supervisor accessible only (if SUPV = 1) or unrestricted (if SUPV = 0). Unrestricted allows both user and supervisor access.
Y = m111, where m is the state of the modmap bit in the module configuration register of the system integration module (Y = $7 or $F).

F

F.2 Parameter RAM Map

Channel	Parameter							
	0	1	2	3	4	5	6	7
0	X $YFFF00	02	04	06	08	0A	—	—
1	X $YFFF10	12	14	16	18	1A	—	—
2	X $YFFF20	22	24	26	28	2A	—	—
3	X $YFFF30	32	34	36	38	3A	—	—
4	X $YFFF40	42	44	46	48	4A	—	—
5	X $YFFF50	52	54	56	58	5A	—	—
6	X $YFFF60	62	64	66	68	6A	—	—
7	X $YFFF70	72	74	76	78	7A	—	—
8	X $YFFF80	82	84	86	88	8A	—	—
9	X $YFFF90	92	94	96	98	9A	—	—
10	X $YFFFA0	A2	A4	A6	A8	AA	—	—
11	X $YFFFB0	B2	B4	B6	B8	BA	—	—
12	X $YFFFC0	C2	C4	C6	C8	CA	—	—
13	X $YFFFD0	D2	D4	D6	D8	DA	—	—
14	X $YFFFE0	E2	E4	E6	E8	EA	EC	EE
15	X $YFFFF0	F2	F4	F6	F8	FA	FC	FE

— = Not Implemented
X = Assignable as supervisor accessible only (if SUPV = 1) or unrestricted (if SUPV = 0). Unrestricted allows both user and supervisor access.
Y = m111, where m is the modmap bit in the module configuration register of the system integration module (Y = $7 or $F).

F.3 TPU Registers

TMCR — TPU Module Configuration Register $YFFE00

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
STOP	TCR1P		TCR2P		EMU	T2CG	STF	SUPV	PSCK	0	0	IARB			

RESET:

0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0

STOP — Stop Bit
    0 = Internal clocks not shut down (reset condition)
    1 = Internal clocks shut down

TCR1P — TCR1 Prescaler Control
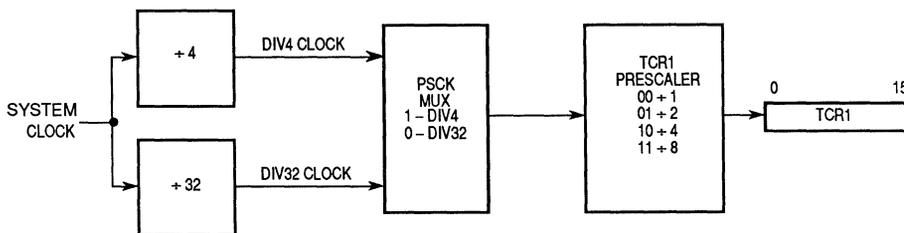
TCR2P — TCR2 Prescaler Control
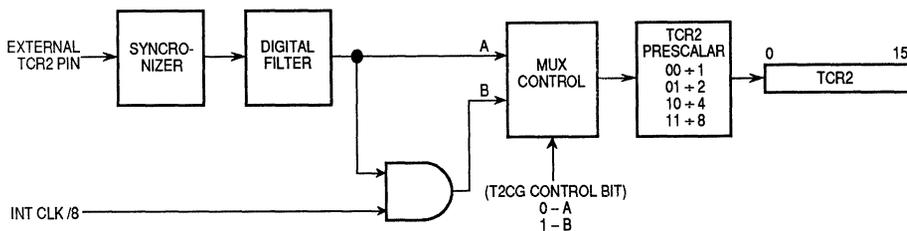
EMU — Emulation Control
    0 = TPU and RAM not in emulation mode (reset condition)
    1 = TPU and RAM in emulation mode

T2CG — TCR2 Clock/Gate Control
    0 = TCR2 pin used as clock source for TCR2 (reset condition)
    1 = TCR2 pin used as gate of DIV8 clock for TCR2

STF — Stop Flag
    0 = TPU operating (reset condition)
    1 = TPU stopped

SUPV — Supervisor Data Space
    0 = Assignable registers are unrestricted (FC2 is ignored).
    1 = (Reset Condition) Assignable registers are restricted (FC2 is decoded).

PSCK — Prescaler Clock
    0 = DIV32 (system clock/32) is input to TCR1 prescaler.
    1 = DIV4 (system clock/4) is input to TCR1 prescaler.

IARB — Interrupt Arbitration ID Bits
   This field contains the arbitration number of the TPU used to arbitrate for the
   intermodule bus (IMB) when two or more modules or peripherals have an interrupt at
   the same priority level.

TCR — Test Configuration Register   **$YFFE02**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	INCAD	TCR1C	ACIITR1	ACITR0	0	0	SOSEL2	SOSEL1	SOSEL0	SISEL2	SISEL1	SISEL0	TMM

RESET:

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

BITS 15–13, 8, 7 — Not Implemented

INCAD — Increment Address
   When set, INCAD forces the µPC to increment once for each assertion of the
   ACUTL line. This feature is used to dump the microROM sequentially .
      0 = Normal operation
      1 = µPC increments

TCR1C — TCR1 Clock
   TCR1C selects the clock source, internal or external, for TCR1. This bit can be read
   or written in test mode only.
      0 = Normal operation (TCR1 clocked internally)
      1 = TCR1 clocked externally via the TCR2 pin; TCR1 is now clocked in parallel
           with TCR2.

ACUTR1–0 — Activate Circuit Under Test Response 1, 0
   The ACUTR bits determine the TPU module response to the test submodule
   asserting ACUT only.

ACUTR1–0	TPU Response
00	None
01	Run one TPU microcycle (Single Step)
10	Assert scheduler end-of-time-slot (Test HS PLA)
11	Reserved

SOSEL2–0 — Scan-Out Select 2–0

These bits define the TPU output scan path to be connected to master shift register B (MSRB) through the SCANB line.

SOSEL1–0	Output Scan Path
000	None
001	μPC
010	Microinstruction
011	Branch PLA
100	μPC Breakpoint
101	Scheduler PLA
110	Channel Breakpoint
111	Reserved

SISEL2–SISEL0 — Scan-In Select 2–0

The bits define the TPU input scan path to be connected to master shift register A (MSRA) through the SCANA line.

SISEL1-0	Input Scan Path
000	None
001	μPC
010	Microinstruction
011	Branch PLA
100	μPC Breakpoint
101	Scheduler PLA
110	Channel Breakpoint
111	Reserved

TMM — Test Memory Map
   0 = Normal memory map
   1 = Module located at $YDF000–$YDFFFF

DSCR — Development Support Control Register   **$YFFE04**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
HOT4	0	0	0	0	BLC	CLKS	FRZ1	FRZ0	CCL	BP	BC	BH	BL	BM	BT

RESET:

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

HOT4 — Hang on T4
   0 = Exit wait on T4 state caused by assertion of HOT4
   1 = Enter wait on T4 state

F

Bits 14–11 — Not Implemented

BLC — Branch Latch Control
    1 = Do not latch conditions into branch condition register before exiting the halted state or during the time-slot transition period.
    0 = Latch conditions into branch condition register prior to exiting halted state.

CLKS — Stop Clocks (to TCRs)
    0 = Do not stop TCRs.
    1 = Stop TCRs during the halted state.

FRZ1 – 0 — IMB FREEZE Response
    The FRZ bits specify the TPU microengine response to the FREEZE signal.

FRZ1-0	TPU Response
00	Ignore Freeze
01	Reserved
10	Freeze at end of current microcycle
11	Freeze at next time-slot boundary

CCL — Channel Conditions Latch
    CCL controls the latching of channel conditions (MRL and TDL) when the CHAN register is written.
    0 = Only the pin state condition of the new channel is latched as a result of the write CHAN register microinstruction.
    1 = Pin state, MRL, and TDL conditions of the new channel are latched as a result of a write CHAN register microinstruction.

BP, BC, BH, BL, BM, and BT — Breakpoint Enable Bits
    Bits 5–0 contain the various breakpoint enable bits for the TPU, specifying the conditions for a breakpoint. The breakpoint is enabled by setting the corresponding bit to one and is disabled by setting the bit to zero.
    BP — Break if μPC equals μPC breakpoint register.
    BC — Break if CHAN register equals channel breakpoint register at beginning of state or when CHAN is changed through microcode.
    BH — Break if host service latch is asserted at beginning of state.
    BL — Break if link service latch is asserted at beginning of state.
    BM — Break if MRL is asserted at beginning of state.
    BT — Break if TDL is asserted at beginning of state.

F

DSSR — Development Support Status Register$YFFE06

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	0	BKPT	PCBK	CHBK	SRBK	TPUF	0	0	0

RESET.

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bits 15–8, 2–0 — Not Implemented

BKPT — Breakpoint Asserted Flag
    If an internal breakpoint caused the TPU to enter the halted state, the TPU asserts
    the BRKPT signal on the IMB and the BKPT flag. The TPU continues to assert
    BKPT until it recognizes a breakpoint acknowledge cycle from a host, or until the
    FREEZE signal on the IMB is asserted.

PCB — μPC Breakpoint Flag
    PCBK is asserted if a breakpoint occurs because of a μPC register match with the
    μPC breakpoint register. PCBK is negated when the BKPT flag is negated.

CHBK — Channel Register Breakpoint Flag
    CHBK is asserted if a breakpoint occurs because of a CHAN register match with the
    channel register breakpoint register. CHBK is negated when the BKPT flag is
    negated.

SRBK — Service Request Breakpoint Flag
    SRBK is asserted if a breakpoint occurs because of any of the service request
    latches being asserted along with their corresponding enable flag in the development
    support control register. SRBK is negated when the BKPT flag is negated.

TPUF — TPU FREEZE Flag
    TPUF is asserted whenever the TPU is in a halted state as a result of FREEZE
    being asserted. This flag is automatically negated when the TPU exits the halted
    state because of FREEZE being negated.

TICR — TPU Interrupt Configuration Register$YFFE08

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

RESET:

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bits 15–11 — Not Implemented

F

CIRL — Channel Interrupt Request Level

The interrupt request level for all channels is specified by this three-bit encoded field. Level seven for this field indicates a nonmaskable interrupt; level zero indicates that all channel interrupts are disabled.

CIBV — Channel Interrupt Base Vector

This field specifies the most significant nibble of all 16 TPU channel interrupt vector numbers.

Bits 3–0 — Not Implemented

CIER — Channel Interrupt Enable Register $YFFE0A

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

RESET:

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

CH 15–0 — Interrupt Enable/Disable for each Channel
  0 = Channel interrupts disabled
  1 = Channel interrupts enabled

CFSR0–3 — Channel Function Select Register 0–3 $YFFE0C–$YFFE12

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

RESET:

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

CH 15–0 — Encoded One of 16 Time Functions for each Channel

HSQR0 — Host Sequence Register 0 $YFFE14

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

RESET:

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

HSQR1 — Host Sequence Register 1   **$YFFE16**

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

RESET:

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

CH 15–0 — Encoded Host Sequence

HSRR0 — Host Service Request Register 0   **$YFFE18**

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

RESET:

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

HSRR1 — Host Service Request Register 1   **$YFFE1A**

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

RESET:

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

CH 15–0 — Encoded Type of Host Service

Bit 1	Bit 0	Service
0	0	No Host Service (Reset Condition)
0	1	Type 1 Host Service
1	0	Type 2 Host Service
1	1	Type 3 Host Service

F

Function Name	Function Code	Host Service Request Code	Host Sequence Code*
DIO Discrete Input/Output	$8	1= Force Output High 2 = Force Output Low 3 = Initialization, Input Specified 3 = Initialization, Periodic Input 3 = Update Pin Status Parameter	0 = Trans Mode — Record Pin on Transition 0 = Trans Mode — Record Pin on Transition 0 = Trans Mode — Record Pin on Transition 1 = Match Mode — Record Pin at Match_Rate 2 = Record Pin State on HSR 11
ITC Input Capture/ Input Transition Counter	$A	0 = None 1 = Initialization 2 = (Not Implemented) 3 = (Not Implemented)	0 = No Link, Single Mode 1 = No Link, Continuous Mode 2 = Link, Single Mode 3 = Link, Continuous Mode
OC Output Compare	$E	0 = None 1 = Host-Initiated Pulse Mode 2 = (Not Implemented) 3 = Continuous Pulse Mode	0 = Execute All Functions 1 = Execute All Functions 2 = Only Update TCRn Parameters 3 = Only Update TCRn Parameters
PWM Pulse-Width Modulation	$9	0 = None 1 = Immediate Update Request 2 = Initialization 3 = (Not Implemented)	(None Implemented)
SPWM Synchronized Pulse- Width Modulation	$7	0 = None 1 = (Not Implemented) 2 = Initialization 3 = Immediate Update Request	0 = Mode 0 1 = Mode 1 2 = Mode 2 3 =(Not Implemented)
PMA/PMM Period Measurement with Additional/Missing Transition Detect	$B	0 = None 1 = Initialization 2 = (Not Implemented) 3 = (Not Implemented)	0 = PMA Bank Mode 1 = PMA Count Mode 2 = PMM Bank Mode 3 = PMM Count Mode
PSP Position-Synchronized Pulse Generator	$C	0 = None 1 = Immediate Update Request 2 = Initialization 3 = Force Change	0 = Pulse Width Set by Angle 1 = Pulse Width Set by Time 2 = Pulse Width Set by Angle 3 = Pulse Width Set by Time
SM Stepper Motor	$D	0 = None 1 = None 2 = Initialization 3 = Step Request	(None Implemented)
PPWA Period/Pulse-Width Accumulator	$F	0 = None 1 = (Not Implemented) 2 = Initialization 3 = (Not Implemented)	0 = 24-Bit Period 1 = 16-Bit Period + Link 2 = 24-Bit Pulse Width 3 = 16-Bit Pulse Width + Link

*Host Sequence Code interpretation is determined by the function; some HSQ codes apply to all HSR codes, some to only one, such as Init.

F

CPR0 — Channel Priority Register 0   **$YFFE1C**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CH 15		CH 14		CH13		CH 12		CH 11		CH 10		CH 9		CH 8	

RESET:

| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

CPR1 — Channel Priority Register 1   **$YFFE1E**

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

RESET:

| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

CH 15–0 Encoded One of Three Channel Priority Level

Bit 1	Bit 0	Service
0	0	Channel Disabled (Reset Condition)
0	1	Low
1	0	Middle
1	1	High

CISR — Channel Interrupt Status Register   **$YFFE20**

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

RESET:

| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

CH 15–0 — Interrupt Status Bit
   0 = Channel interrupt not asserted (reset condition)
   1 = Channel interrupt asserted

LR — Link Register   **$YFFE22**

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

RESET:

| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

CH 15–0 — Test Mode Link Service Request Enable Bit
   0 = Link bit not asserted
   1 = Link bit asserted

F

SGLR — Service Grant Latch Register   **$YFFE24**

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

RESET:

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

CH 15–0 — Service Granted Bit

DCNR — Decoded Channel Number Register   **$YFFE26**

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

RESET:

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

CH 15-0 — Service Status Bit

INDEX

RXD Pin 5-74, 5-79

— S —

(A) MOTOROLA

Literature Distribution Centers:

USA: Motorola Literature Distribution; P.O. Box 20912; Phoenix, Arizona 85036.
EUROPE: Motorola Ltd.; European Literature Center; 88 Tanners Drive, Blakelands, Milton Keynes, MK14 5BP, England.
JAPAN: Nippon Motorola Ltd.; 4-32-1, Nishi-Gotanda, Shinagawa-ku, Tokyo 141 Japan.
ASIA-PACIFIC: Motorola Semiconductors H.K. Ltd.; Silicon Harbour Center, No. 2 Dai King Street, Tai Po Industrial Estate,
        Tai Po, N.T., Hong Kong.