SECTION 5
SYSTEM INTEGRATION MODULE

This section is an overview of the system integration module (SIM) function. Refer to the ***SIM Reference Manual*** (SIMRM/AD) for a comprehensive discussion of SIM capabilities. Refer to **D.2 System Integration Module** for information concerning the SIM address map and register structure.

5.1 General

The SIM consists of six functional blocks. **Figure 5-1** shows a block diagram of the SIM.

The system configuration block controls MCU configuration parameters.

The system clock generates clock signals used by the SIM, other IMB modules, and external devices.

The system protection block provides bus and software watchdog monitors. In addition, it also provides a periodic interrupt timer to support execution of time-critical control routines.

The external bus interface handles the transfer of information between IMB modules and external address space.

The chip-select block provides 12 chip-select signals. Each chip-select signal has an associated base address register and option register that contain the programmable characteristics of that chip-select.

The system test block incorporates hardware necessary for testing the MCU. It is used to perform factory tests, and its use in normal applications is not supported.

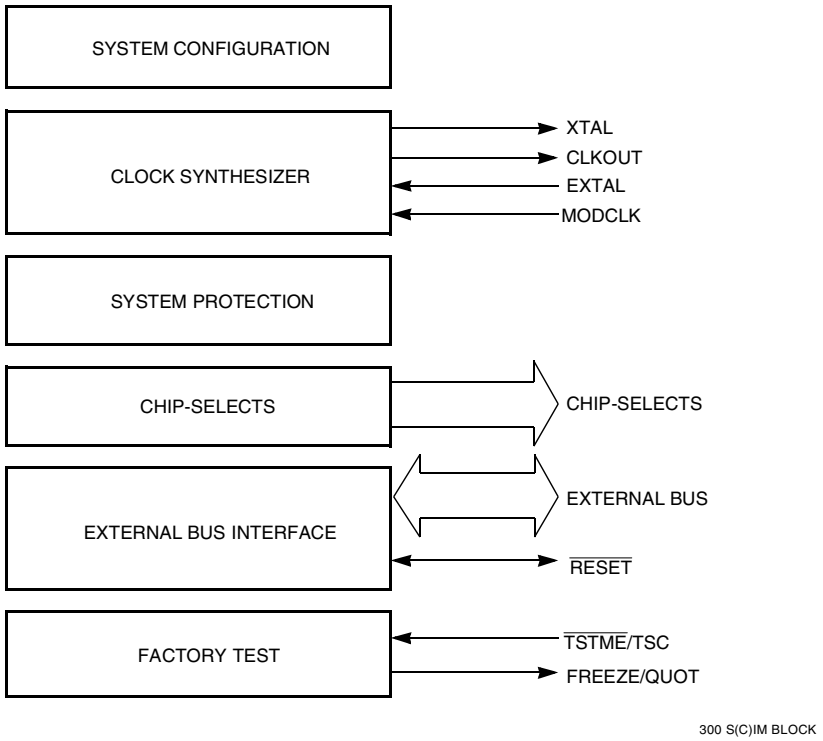


Figure 5-1 System Integration Module Block Diagram

5.2 System Configuration

The SIM configuration register (SIMCR) governs several aspects of system operation. The following paragraphs describe those configuration options controlled by SIMCR.

5.2.1 Module Mapping

Control registers for all the modules in the microcontroller are mapped into a 4-Kbyte block. The state of the module mapping bit (MM) in the SIM configuration register (SIMCR) determines where the control register block is located in the system memory map. When MM = 0, register addresses range from $7FF000 to $7FFFFF; when MM = 1, register addresses range from $FFF000 to $FFFFFF.

5.2.2 Interrupt Arbitration

Each module that can request interrupts has an interrupt arbitration (IARB) field. Arbitration between interrupt requests of the same priority is performed by serial contention between IARB field bit values. Contention must take place whenever an interrupt request is acknowledged, even when there is only a single request pending. For an interrupt to be serviced, the appropriate IARB field must have a non-zero value. If an interrupt request from a module with an IARB field value of %0000 is recognized, the CPU32 processes a spurious interrupt exception.

Because the SIM routes external interrupt requests to the CPU32, the SIM IARB field value is used for arbitration between internal and external interrupts of the same prior-

ity. The reset value of IARB for the SIM is %1111, and the reset IARB value for all other modules is %0000, which prevents SIM interrupts from being discarded during initialization. Refer to **5.8 Interrupts** for a discussion of interrupt arbitration.

5.2.3 Show Internal Cycles

A show cycle allows internal bus transfers to be monitored externally. The SHEN field in SIMCR determines what the external bus interface does during internal transfer operations. **Table 5-1** shows whether data is driven externally, and whether external bus arbitration can occur. Refer to **5.6.6.1 Show Cycles** for more information.

Table 5-1 Show Cycle Enable Bits

SHEN[1:0]	Action
00	Show cycles disabled, external arbitration enabled
01	Show cycles enabled, external arbitration disabled
10	Show cycles enabled, external arbitration enabled
11	Show cycles enabled, external arbitration enabled; internal activity is halted by a bus grant

5.2.4 Register Access

The CPU32 can operate at one of two privilege levels. Supervisor level is more privileged than user level — all instructions and system resources are available at supervisor level, but access is restricted at user level. Effective use of privilege level can protect system resources from uncontrolled access. The state of the S bit in the CPU status register determines access level, and whether the user or supervisor stack pointer is used for stacking operations. The SUPV bit places SIM global registers in either supervisor or user data space. When SUPV = 0, registers with controlled access are accessible from either the user or supervisor privilege level; when SUPV = 1, registers with controlled access are restricted to supervisor access only.

5.2.5 Freeze Operation

The FREEZE signal halts MCU operations during debugging. FREEZE is asserted internally by the CPU32 if a breakpoint occurs while background mode is enabled. When FREEZE is asserted, only the bus monitor, software watchdog, and periodic interrupt timer are affected. The halt monitor and spurious interrupt monitor continue to operate normally. Setting the freeze bus monitor (FRZBM) bit in SIMCR disables the bus monitor when FREEZE is asserted. Setting the freeze software watchdog (FRZSW) bit disables the software watchdog and the periodic interrupt timer when FREEZE is asserted.

5.3 System Clock

The system clock in the SIM provides timing signals for the IMB modules and for an external peripheral bus. Because the MCU is a fully static design, register and memory contents are not affected when the clock rate changes. System hardware and software support changes in clock rate during operation.

The system clock signal can be generated from one of two sources. An internal phase-locked loop (PLL) can synthesize the clock from a fast reference, or the clock signal can be directly input from an external frequency source. The fast reference is typically a 4.194 MHz crystal, but may be generated by sources other than a crystal. Keep these sources in mind while reading the rest of this section. Refer to **Table A-4** in the **APPENDIX A ELECTRICAL CHARACTERISTICS** for clock specifications.

Figure 5-2 is a block diagram of the clock submodule.

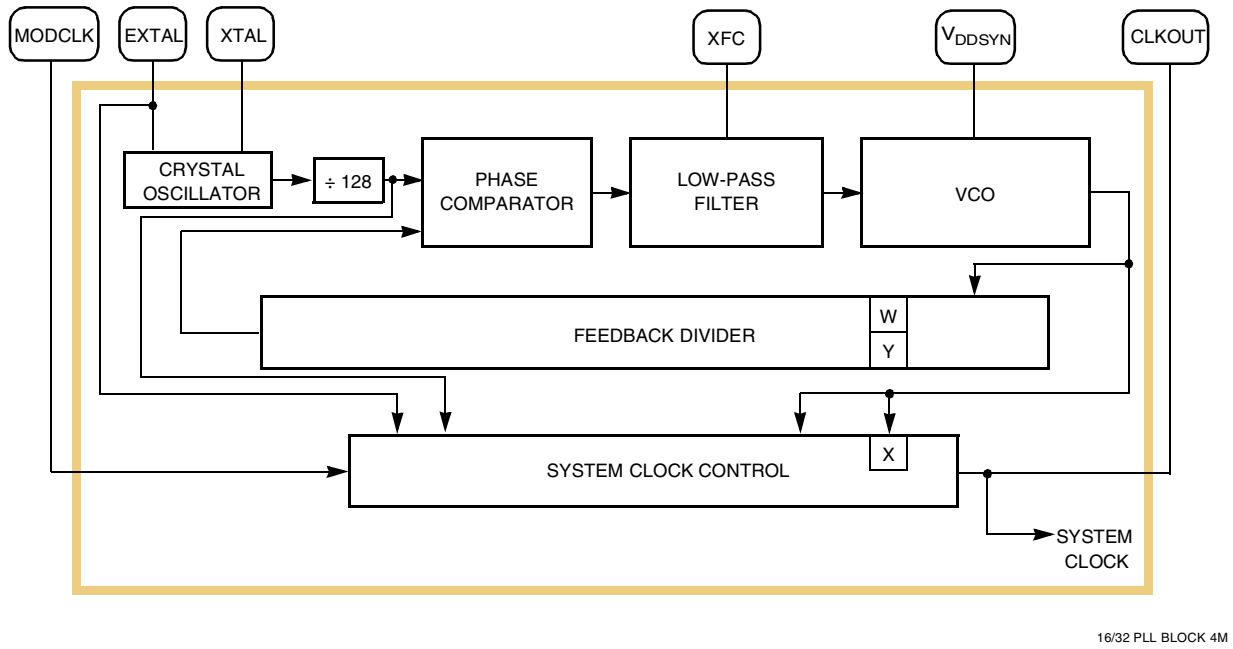


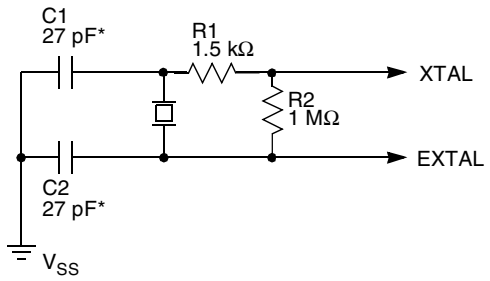
Figure 5-2 System Clock Block Diagram

5.3.1 Clock Sources

The state of the clock mode (MODCLK) pin during reset determines the system clock source. When MODCLK is held high during reset, the clock synthesizer generates a clock signal from an external reference frequency. The clock synthesizer control register (SYNCR) determines operating frequency and mode of operation. When MODCLK is held low during reset, the clock synthesizer is disabled and an external system clock signal must be driven onto the EXTAL pin.

The input clock is referred to as f_{ref}, and can be either a crystal or an external clock source. The output of the clock system is referred to as f_{sys}. Ensure that f_{ref} and f_{sys} are within normal operating limits.

To generate a reference frequency using the crystal oscillator, a reference crystal must be connected between the EXTAL and XTAL pins. Typically, a 4.194 MHz crystal is used, but the frequency may vary between 1 and 6 MHz. **Figure 5-3** shows a typical circuit.



* RESISTANCE AND CAPACITANCE BASED ON A TEST CIRCUIT CONSTRUCTED WITH A KDS041-18 4.194 MHz CRYSTAL. SPECIFIC COMPONENTS MUST BE BASED ON CRYSTAL TYPE. CONTACT CRYSTAL VENDOR FOR EXACT CIRCUIT.
32 OSCILLATOR 4M

Figure 5-3 System Clock Oscillator Circuit

If a fast reference frequency is provided to the PLL from a source other than a crystal, or an external system clock signal is applied through the EXTAL pin, the XTAL pin must be left floating.

When an external system clock signal is applied (MODCLK = 0 during reset), the PLL is disabled. The duty cycle of this signal is critical, especially at operating frequencies close to maximum. The relationship between clock signal duty cycle and clock signal period is expressed as follows:

$$\text{Minimum External Clock Period} = \frac{\text{Minimum External Clock High/Low Time}}{50\% - \text{Percentage Variation of External Clock Input Duty Cycle}}$$

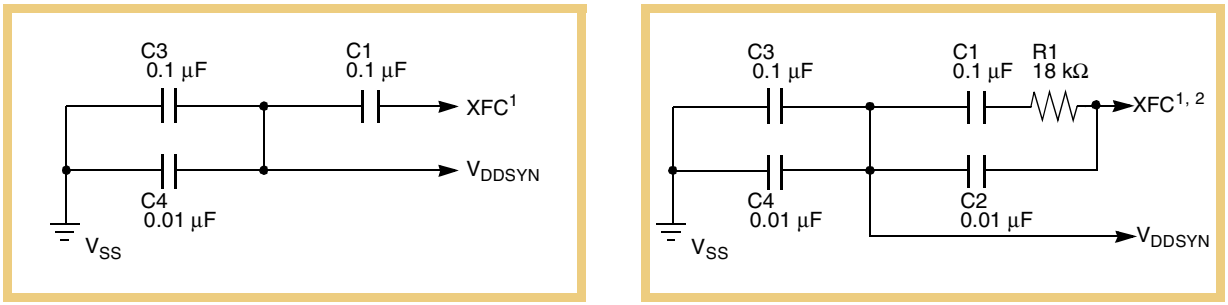
5.3.2 Clock Synthesizer Operation

V_{DDSYN} is used to power the clock circuits when the system clock is synthesized from either a crystal or an externally supplied reference frequency. A separate power source increases MCU noise immunity and can be used to run the clock when the MCU is powered down. A quiet power supply must be used as the V_{DDSYN} source. Adequate external bypass capacitors should be placed as close as possible to the V_{DDSYN} pin to assure a stable operating frequency. When an external system clock signal is applied and the PLL is disabled, V_{DDSYN} should be connected to the V_{DD} supply. Refer to the ***SIM Reference Manual*** (SIMRM/AD) for more information regarding system clock power supply conditioning.

A voltage controlled oscillator (VCO) in the PLL generates the system clock signal. To maintain a 50% clock duty cycle, the VCO frequency (f_{VCO}) is either two or four times the system clock frequency, depending on the state of the X bit in SYNCR. The clock signal is fed back to a divider/counter. The divider controls the frequency of one input to a phase comparator. The other phase comparator input is a reference signal, either from the crystal oscillator or from an external source. The comparator generates a control signal proportional to the difference in phase between the two inputs. This signal is low-pass filtered and used to correct the VCO output frequency.

Filter circuit implementation can vary, depending upon the external environment and required clock stability. **Figure 5-4** shows two recommended system clock filter networks. XFC pin leakage must be kept as low as possible to maintain optimum stability and PLL performance.

An external filter network connected to the XFC pin is not required when an external system clock signal is applied and the PLL is disabled (MODCLK = 0 at reset). The XFC pin must be left floating in this case.



NORMAL OPERATING ENVIRONMENT

HIGH-STABILITY OPERATING ENVIRONMENT

1. MAINTAIN LOW LEAKAGE ON THE XFC NODE. REFER TO **APPENDIX A ELECTRICAL CHARACTERISTICS** FOR MORE INFORMATION.
2. RECOMMENDED LOOP FILTER FOR REDUCED SENSITIVITY TO LOW FREQUENCY NOISE.

NORMAL/HIGH-STABILITY XFC CONN

Figure 5-4 System Clock Filter Networks

The synthesizer locks when the VCO frequency is equal to f_{ref}. Lock time is affected by the filter time constant and by the amount of difference between the two comparator inputs. Whenever a comparator input changes, the synthesizer must relock. Lock status is shown by the SLOCK bit in SYNCR. During power-up, the MCU does not come out of reset until the synthesizer locks. Crystal type, characteristic frequency, and layout of external oscillator circuitry affect lock time.

When the clock synthesizer is used, SYNCR determines the system clock frequency and certain operating parameters. The W and Y[5:0] bits are located in the PLL feedback path, enabling frequency multiplication by a factor of up to 256. When the W or Y values change, VCO frequency changes, and there is a VCO relock delay. The SYNCR X bit controls a divide-by circuit that is not in the synthesizer feedback loop. When X = 0 (reset state), a divide-by-four circuit is enabled, and the system clock frequency is one-fourth the VCO frequency (f_{VCO}). When X = 1, a divide-by-two circuit is enabled and system clock frequency is one-half the VCO frequency (f_{VCO}). There is no relock delay when clock speed is changed by the X bit.

Clock frequency is determined by SYNCR bit settings as follows:

$$f_{sys} = \frac{f_{ref}}{128}[4(Y+1)(2^{(2W+X)})]$$

The reset state of SYNCR ($3F00) results in a power-on f_{sys} of 8.388 MHz when f_{ref} is 4.194 MHz.

For the device to operate correctly, the clock frequency selected by the W, X, and Y bits must be within the limits specified for the MCU.

Internal VCO frequency is determined by the following equations:

$$f_{VCO} = 4f_{sys} \text{ if } X = 0$$

or

$$f_{VCO} = 2f_{sys} \text{ if } X = 1$$

Table 5-2 shows clock control multipliers for all possible combinations of SYNCR bits. To obtain clock frequency, find counter modulus in the leftmost column, then multiply the reference frequency by the value in the appropriate prescaler cell. Shaded cells exceed the maximum system clock frequency at the time of manual publication; however, they may be usable in the future. Refer to **APPENDIX A ELECTRICAL CHARACTERISTICS** for maximum allowable clock rate.

Table 5-3 shows clock frequencies available with a 4.194 MHz reference and a maximum specified clock frequency of 20.97 MHz. To obtain clock frequency, find counter modulus in the leftmost column, then refer to appropriate prescaler cell. Shaded cells exceed the maximum system clock frequency at the time of manual publication; however, they may be usable in the future. Refer to **APPENDIX A ELECTRICAL CHARACTERISTICS** for maximum system frequency (f_{sys}).

Table 5-2 Clock Control Multipliers

Modulus	Prescalers			
Y	[W:X] = 00	[W:X] = 01	[W:X] = 10	[W:X] = 11
000000	.03125	.625	.125	.25
000001	.0625	.125	.25	.5
000010	.09375	.1875	.375	.75
000011	.125	.25	.5	1
000100	.15625	.3125	.625	1.25
000101	.1875	.375	.75	1.5
000110	.21875	.4375	.875	1.75
000111	.25	.5	1	2
001000	.21825	.5625	1.125	2.25
001001	.3125	.625	1.25	2.5
001010	.34375	.6875	1.375	2.75
001011	.375	.75	1.5	3
001100	.40625	.8125	1.625	3.25
001101	.4375	.875	1.75	3.5
001110	.46875	.9375	1.875	3.75
001111	.5	1	2	4

Table 5-2 Clock Control Multipliers (Continued)

Modulus	Prescalers			
Y	[W:X] = 00	[W:X] = 01	[W:X] = 10	[W:X] = 11
010000	.53125	1.0625	2.125	4.25
010001	.5625	1.125	2.25	4.5
010010	.59375	1.1875	2.375	4.75
010011	.625	1.25	2.5	5
010100	.65625	1.3125	2.625	5.25
010101	.6875	1.375	2.75	5.5
010110	.71875	1.4375	2.875	5.75
010111	.75	1.5	3	6
011000	.78125	1.5625	3.125	6.25
011001	.8125	1.625	3.25	6.5
011010	.84375	1.6875	3.375	6.75
011011	.875	1.75	3.5	7
011100	.90625	1.8125	3.625	7.25
011101	.9375	1.875	3.75	7.5
011110	.96875	1.9375	3.875	7.75
011111	1	2	4	8
100000	1.03125	2.0625	4.125	8.25
100001	1.0625	2.125	4.25	8.5
100010	1.09375	2.1875	4.375	8.75
100011	1.125	2.25	4.5	9
100100	1.15625	2.3125	4.675	9.25
100101	1.1875	2.375	4.75	9.5
100110	1.21875	2.4375	4.875	9.75
100111	1.25	2.5	5	10
101000	1.28125	2.5625	5.125	10.25
101001	1.3125	2.625	5.25	10.5
101010	1.34375	2.6875	5.375	10.75
101011	1.375	2.75	5.5	11
101100	1.40625	2.8125	5.625	11.25
101101	1.4375	2.875	5.75	11.5
101110	1.46875	2.9375	5.875	11.75
101111	1.5	3	6	12
110000	1.53125	3.0625	6.125	12.25
110001	1.5625	3.125	6.25	12.5
110010	1.59375	3.1875	6.375	12.75
110011	1.625	3.25	6.5	13
110100	1.65625	3.3125	6.625	13.25
110101	1.6875	3.375	6.75	13.5
110110	1.71875	3.4375	6.875	13.75
110111	1.75	3.5	7	14
111000	1.78125	3.5625	7.125	14.25

Table 5-2 Clock Control Multipliers (Continued)

Modulus	Prescalers			
Y	[W:X] = 00	[W:X] = 01	[W:X] = 10	[W:X] = 11
111001	1.8125	3.625	7.25	14.5
111010	1.84375	3.6875	7.375	14.75
111011	1.875	3.75	7.5	15
111100	1.90625	3.8125	7.625	15.25
111101	1.9375	3.875	7.75	15.5
111110	1.96875	3.9375	7.875	15.75
111111	2	4	8	16

Table 5-3 System Frequencies from 4.194 MHz Reference

Modulus	Prescaler			
Y	[W:X] = 00	[W:X] = 01	[W:X] = 10	[W:X] = 11
000000	131 kHz	262 kHz	524 kHz	1049 kHz
000001	262	524	1049	2097
000010	393	786	1573	3146
000011	524	1049	2097	4194
000100	655	1311	2621	5243
000101	786	1573	3146	6291
000110	918	1835	3670	7340
000111	1049	2097	4194	8389
001000	1180	2359	4719	9437
001001	1311	2621	5243	10486
001010	1442	2884	5767	11534
001011	1573	3146	6291	12583
001100	1704	3408	6816	13631
001101	1835	3670	7340	14680
001110	1966	3932	7864	15729
001111	2097	4194	8389	16777
010000	2228	4456	8913	17826
010001	2359	4719	9437	18874
010010	2490	4981	9961	19923
010011	2621	5243	10486	20972
010100	2753	5505	11010	22020
010101	2884	5767	11534	23069
010110	3015	6029	12059	24117
010111	3146	6291	12583	25166
011000	3277	6554	13107	26214
011001	3408	6816	13631	27263
011010	3539	7078	14156	28312
011011	3670	7340	14680	29360
011100	3801	7602	15204	30409

Table 5-3 System Frequencies from 4.194 MHz Reference (Continued)

Modulus	Prescaler			
Y	[W:X] = 00	[W:X] = 01	[W:X] = 10	[W:X] = 11
011101	3932	7864	15729	31457
011110	4063	8126	16253	32506
011111	4194	8389	16777	33554
100000	4325 kHz	8651 kHz	17302 kHz	34603 kHz
100001	4456	8913	17826	35652
100010	4588	9175	18350	36700
100011	4719	9437	18874	37749
100100	4850	9699	19399	38797
100101	4981	9961	19923	39846
100110	5112	10224	20447	40894
100111	5243	10486	20972	41943
101000	5374	10748	21496	42992
101001	5505	11010	22020	44040
101010	5636	11272	22544	45089
101011	5767	11534	23069	46137
101100	5898	11796	23593	47186
101101	6029	12059	24117	48234
101110	6160	12321	24642	49283
101111	6291	12583	25166	50332
110000	6423	12845	25690	51380
110001	6554	13107	26214	52428
110010	6685	13369	26739	53477
110011	6816	13631	27263	54526
110100	6947	13894	27787	55575
110101	7078	14156	28312	56623
110110	7209	14418	28836	57672
110111	7340	14680	29360	58720
111000	7471	14942	2988	59769
111001	7602	15204	30409	60817
111010	7733	15466	30933	61866
111011	7864	15729	31457	62915
111100	7995	15991	31982	63963
111101	8126	16253	32506	65011
111110	8258	16515	33030	66060
111111	8389	16777	33554	67109

5.3.3 External Bus Clock

The state of the E-clock division bit (EDIV) in SYNCR determines clock rate for the E-clock signal (ECLK) available on pin ADDR23. ECLK is a bus clock for M6800 devices and peripherals. ECLK frequency can be set to system clock frequency divided by eight or system clock frequency divided by sixteen. The clock is enabled by the $\overline{CS10}$

field in chip-select pin assignment register 1 (CSPAR1). ECLK operation during low-power stop is described in the following paragraph. Refer to **5.9 Chip-Selects** for more information about the external bus clock.

5.3.4 Low-Power Operation

Low-power operation is initiated by the CPU32. To reduce power consumption selectively, the CPU can set the STOP bits in each module configuration register. To minimize overall microcontroller power consumption, the CPU can execute the LPSTOP instruction, which causes the SIM to turn off the system clock.

When individual module STOP bits are set, clock signals inside each module are turned off, but module registers are still accessible.

When the CPU executes LPSTOP, a special CPU space bus cycle writes a copy of the current interrupt mask into the clock control logic. The SIM brings the MCU out of low-power stop mode when one of the following exceptions occur:

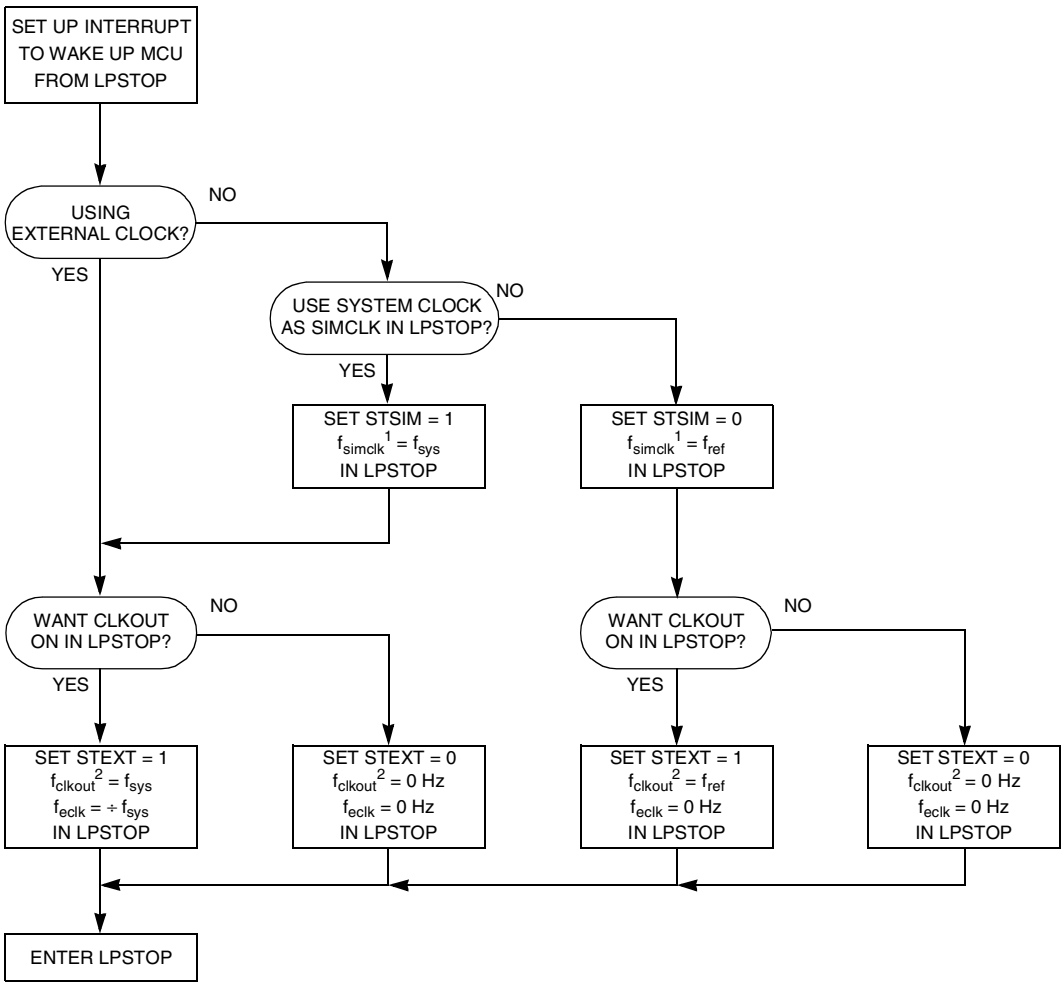
- U-bus interface
- RESET
- Trace
- SIM interrupt of higher priority than the stored interrupt mask

Refer to **5.6.4.2 LPSTOP Broadcast Cycle** and **4.8.2.1 Low-Power Stop (LPSTOP)** for more information.

During low-power stop mode, unless the system clock signal is supplied by an external source and that source is removed, the SIM clock control logic and the SIM clock signal (SIMCLK) continue to operate. The periodic interrupt timer and input logic for the RESET and IRQ pins are clocked by SIMCLK, and can be used to bring the processor out of LPSTOP. Optionally, the SIM can also continue to generate the CLKOUT signal while in low-power stop mode.

STSIM and STEXT bits in SYNCR determine clock operation during low-power stop mode.

The flowchart shown in **Figure 5-5** summarizes the effects of the STSIM and STEXT bits when the MCU enters normal low power stop mode. Any clock in the off state is held low. If the synthesizer VCO is turned off during low-power stop mode, there is a PLL relock delay after the VCO is turned back on.



NOTES:
1. THE SIMCLK IS USED BY THE PIT, $\overline{IRQ}$, AND INPUT BLOCKS OF THE SIM.
2. CLKOUT CONTROL DURING LPSTOP IS OVERRIDDEN BY THE EXOFF BIT IN SIMCR. IF EXOFF = 1, THE CLKOUT PIN IS ALWAYS IN A HIGH IMPEDANCE STATE AND STEXT HAS NO EFFECT IN LPSTOP. IF EXOFF = 0, CLKOUT IS CONTROLLED BY STEXT IN LPSTOP.

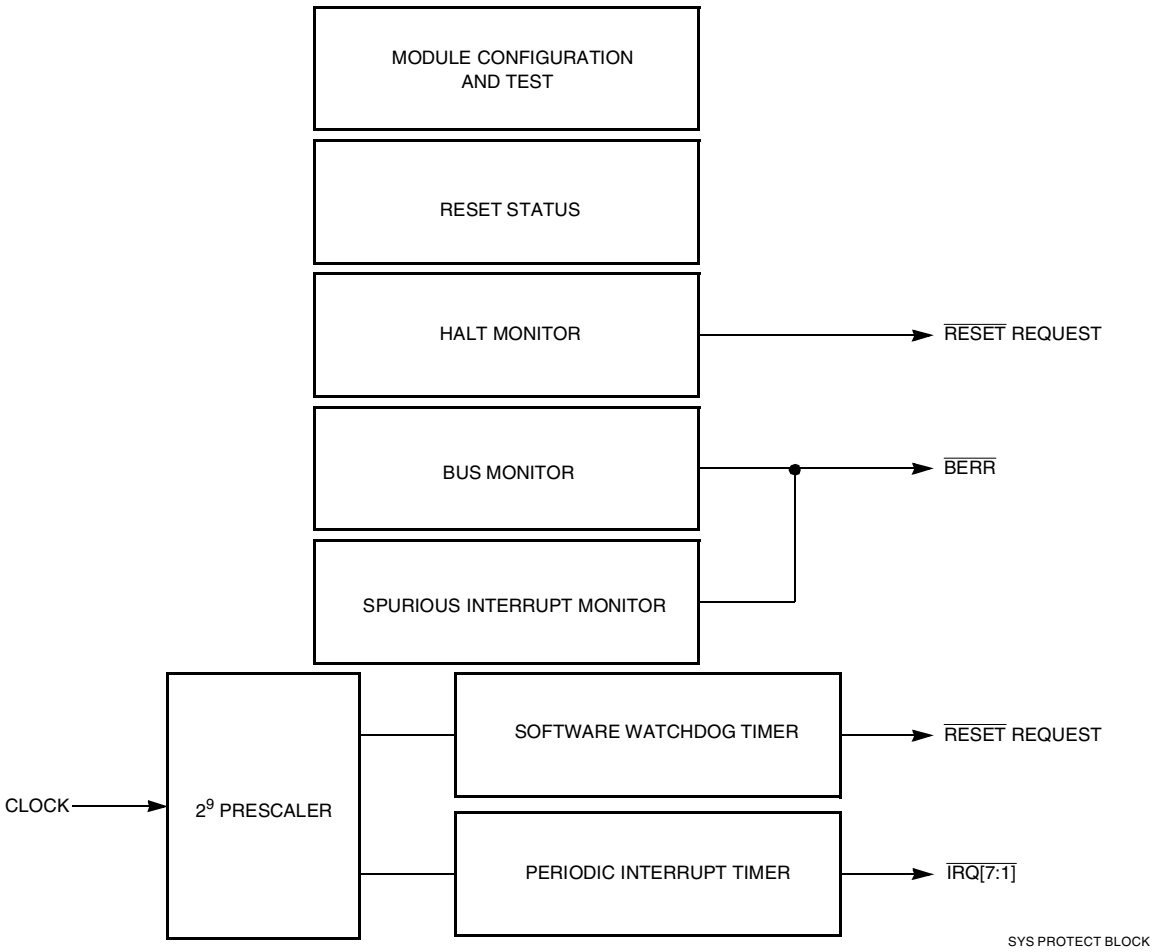
LPSTOPFLOW

Figure 5-5 LPSTOP Flowchart

5.4 System Protection

The system protection block preserves reset status, monitors internal activity, and provides periodic interrupt generation. **Figure 5-6** is a block diagram of the submodule.

Figure 5-6 System Protection Block



5.4.1 Reset Status

The reset status register (RSR) latches internal MCU status during reset. Refer to **5.7.10 Reset Status Register** for more information.

5.4.2 Bus Monitor

The internal bus monitor checks data and size acknowledge ($\overline{\text{DSACK}}$) or autovector ($\overline{\text{AVEC}}$) signal response times during normal bus cycles. The monitor asserts the internal bus error ($\overline{\text{BERR}}$) signal when the response time is excessively long.

$\overline{\text{DSACK}}$ and $\overline{\text{AVEC}}$ response times are measured in clock cycles. Maximum allowable response time can be selected by setting the bus monitor timing (BMT[1:0]) field in the system protection control register (SYPCR). **Table 5-4** shows the periods allowed.

Table 5-4 Bus Monitor Period

BMT[1:0]	Bus Monitor Time-Out Period
00	64 system clocks
01	32 system clocks
10	16 system clocks
11	8 system clocks

The monitor does not check $\overline{\text{DSACK}}$ response on the external bus unless the CPU32 initiates a bus cycle. The BME bit in SYPCR enables the internal bus monitor for internal to external bus cycles. If a system contains external bus masters, an external bus monitor must be implemented and the internal-to-external bus monitor option must be disabled.

When monitoring transfers to an 8-bit port, the bus monitor does not reset until both byte accesses of a word transfer are completed. Monitor time-out period must be at least twice the number of clocks that a single byte access requires.

5.4.3 Halt Monitor

The halt monitor responds to an assertion of the $\overline{\text{HALT}}$ signal on the internal bus, caused by a double bus fault. A flag in the reset status register (RSR) indicates that the last reset was caused by the halt monitor. Halt monitor reset can be inhibited by the halt monitor (HME) enable bit in SYPCR. Refer to **5.6.5.2 Double Bus Faults** for more information.

5.4.4 Spurious Interrupt Monitor

During interrupt exception processing, the CPU32 normally acknowledges an interrupt request, recognizes the highest priority source, and then either acquires a vector or responds to a request for autovectoring. The spurious interrupt monitor asserts the internal bus error signal ($\overline{\text{BERR}}$) if no interrupt arbitration occurs during interrupt exception processing. The assertion of $\overline{\text{BERR}}$ causes the CPU32 to load the spurious interrupt exception vector into the program counter. The spurious interrupt monitor cannot be disabled. Refer to **5.8 Interrupts** for more information. For detailed information about interrupt exception processing, refer to **4.9 Exception Processing**.

5.4.5 Software Watchdog

The software watchdog is controlled by the software watchdog enable (SWE) bit in SYPCR. When enabled, the watchdog requires that a service sequence be written to the software service register (SWSR) on a periodic basis. If servicing does not take place, the watchdog times out and asserts the $\overline{\text{RESET}}$ signal.

Each time the service sequence is written, the software watchdog timer restarts. The sequence to restart consists of the following steps:

1. Write $55 to SWSR.
2. Write $AA to SWSR.

Both writes must occur before time-out in the order listed. Any number of instructions can be executed between the two writes.

Watchdog clock rate is affected by the software watchdog prescale (SWP) bit and the software watchdog timing (SWT[1:0]) field in SYPCR.

SWP determines system clock prescaling for the watchdog timer and determines that one of two options, either no prescaling or prescaling by a factor of 512, can be

selected. The value of SWP is affected by the state of the MODCLK pin during reset, as shown in **Table 5-5**. System software can change SWP value.

Table 5-5 MODCLK Pin and SWP Bit During Reset

MODCLK	SWP
0 (PLL disabled)	1 (÷ 512)
1 (PLL enabled)	0 (÷ 1)

SWT[1:0] selects the divide ratio used to establish the software watchdog time-out period. The following equation calculates the time-out period for a fast reference frequency.

$$\text{Time-out Period} = \frac{(128)(\text{Divide Ratio Specified by SWP and SWT[1:0]})}{f_{ref}}$$

The following equation calculates the time-out period for an externally input clock frequency.

$$\text{Time-out Period} = \frac{\text{Divide Ratio Specified by SWP and SWT[1:0]}}{f_{ref}}$$

Table 5-6 shows the divide ratio for each combination of SWP and SWT[1:0] bits. When SWT[1:0] are modified, a watchdog service sequence must be performed before the new time-out period can take effect.

Table 5-6 Software Watchdog Ratio

SWP	SWT[1:0]	Watchdog Time-Out Period
0	00	$2^9 \div f_{sys}$
0	01	$2^{11} \div f_{sys}$
0	10	$2^{13} \div f_{sys}$
0	11	$2^{15} \div f_{sys}$
1	00	$2^{18} \div f_{sys}$
1	01	$2^{20} \div f_{sys}$
1	10	$2^{22} \div f_{sys}$
1	11	$2^{24} \div f_{sys}$

Figure 5-7 is a block diagram of the watchdog timer and the clock control for the periodic interrupt timer.

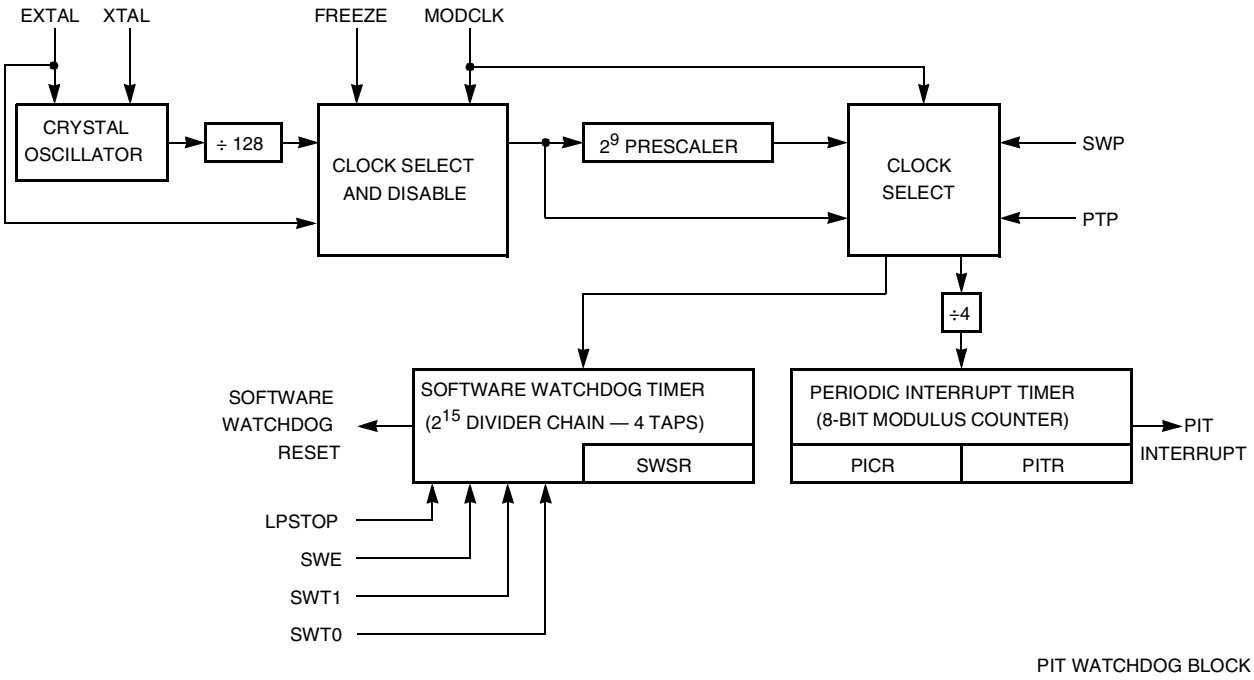


Figure 5-7 Periodic Interrupt Timer and Software Watchdog Timer

5.4.6 Periodic Interrupt Timer

The periodic interrupt timer (PIT) allows the generation of interrupts of specific priority at predetermined intervals. This capability is often used to schedule control system tasks that must be performed within time constraints. The timer consists of a prescaler, a modulus counter, and registers that determine interrupt timing, priority and vector assignment. Refer to **4.9 Exception Processing** for more information.

The periodic interrupt timer modulus counter is clocked by one of two signals. When the PLL is enabled (MODCLK = 1 during reset), $f_{ref} \div 128$ is used. When the PLL is disabled (MODCLK = 0 during reset), f_{ref} is used. The value of the periodic timer prescaler (PTP) bit in the periodic interrupt timer register (PITR) determines system clock prescaling for the periodic interrupt timer. One of two options, either no prescaling, or prescaling by a factor of 512, can be selected. The value of PTP is affected by the state of the MODCLK pin during reset, as shown in **Table 5-7**. System software can change PTP value.

Table 5-7 MODCLK Pin and PTP Bit at Reset

MODCLK	PTP
0 (PLL disabled)	1 (÷ 512)
1 (PLL enabled)	0 (÷ 1)

Either clock signal selected by the PTP is divided by four before driving the modulus counter. The modulus counter is initialized by writing a value to the periodic interrupt timer modulus (PITM[7:0]) field in PITR. A zero value turns off the periodic timer. When

the modulus counter value reaches zero, an interrupt is generated. The modulus counter is then reloaded with the value in PITM[7:0] and counting repeats. If a new value is written to PITR, it is loaded into the modulus counter when the current count is completed.

When a fast reference frequency is used, the PIT period can be calculated as follows:

$$\text{PIT Period} = \frac{(128)(\text{PITM}[7:0])(1 \text{ if PTP} = 0, 512 \text{ if PTP} = 1)(4)}{f_{ref}}$$

When an externally input clock frequency is used, the PIT period can be calculated as follows:

$$\text{PIT Period} = \frac{(\text{PITM}[7:0])(1 \text{ if PTP} = 0, 512 \text{ if PTP} = 1)(4)}{f_{ref}}$$

5.4.7 Interrupt Priority and Vectoring

Interrupt priority and vectoring are determined by the values of the periodic interrupt request level (PIRQL[2:0]) and periodic interrupt vector (PIV) fields in the periodic interrupt control register (PICR).

The PIRQL field is compared to the CPU32 interrupt priority mask to determine whether the interrupt is recognized. **Table 5-8** shows PIRQL[2:0] priority values. Because of SIM hardware prioritization, a PIT interrupt is serviced before an external interrupt request of the same priority. The periodic timer continues to run when the interrupt is disabled.

Table 5-8 Periodic Interrupt Priority

PIRQL[2:0]	Priority Level
000	Periodic interrupt disabled
001	Interrupt priority level 1
010	Interrupt priority level 2
011	Interrupt priority level 3
100	Interrupt priority level 4
101	Interrupt priority level 5
110	Interrupt priority level 6
111	Interrupt priority level 7

The PIV field contains the periodic interrupt vector. The vector is placed on the IMB when an interrupt request is made. The vector number is used to calculate the address of the appropriate exception vector in the exception vector table. The reset value of the PIV field is $0F, which corresponds to the uninitialized interrupt exception vector.

5.4.8 Low-Power STOP Mode Operation

When the CPU32 executes the LPSTOP instruction, the current interrupt priority mask is stored in the clock control logic, internal clocks are disabled according to the state of the STSIM bit in the SYNCR, and the MCU enters low-power stop mode. The bus monitor, halt monitor, and spurious interrupt monitor are all inactive during low-power stop mode.

During low-power stop mode, the clock input to the software watchdog timer is disabled and the timer stops. The software watchdog begins to run again on the first rising clock edge after low-power stop mode ends. The watchdog is not reset by low-power stop mode. A service sequence must be performed to reset the timer.

The periodic interrupt timer does not respond to the LPSTOP instruction, but continues to run during LPSTOP. To stop the periodic interrupt timer, PITR must be loaded with a zero value before the LPSTOP instruction is executed. A PIT interrupt, or an external interrupt request, can bring the MCU out of low-power stop mode if it has a higher priority than the interrupt mask value stored in the clock control logic when low-power stop mode is initiated. LPSTOP can be terminated by a reset.

5.5 External Bus Interface

The external bus interface (EBI) transfers information between the internal MCU bus and external devices. **Figure 5-8** shows a basic system with external memory and peripherals.

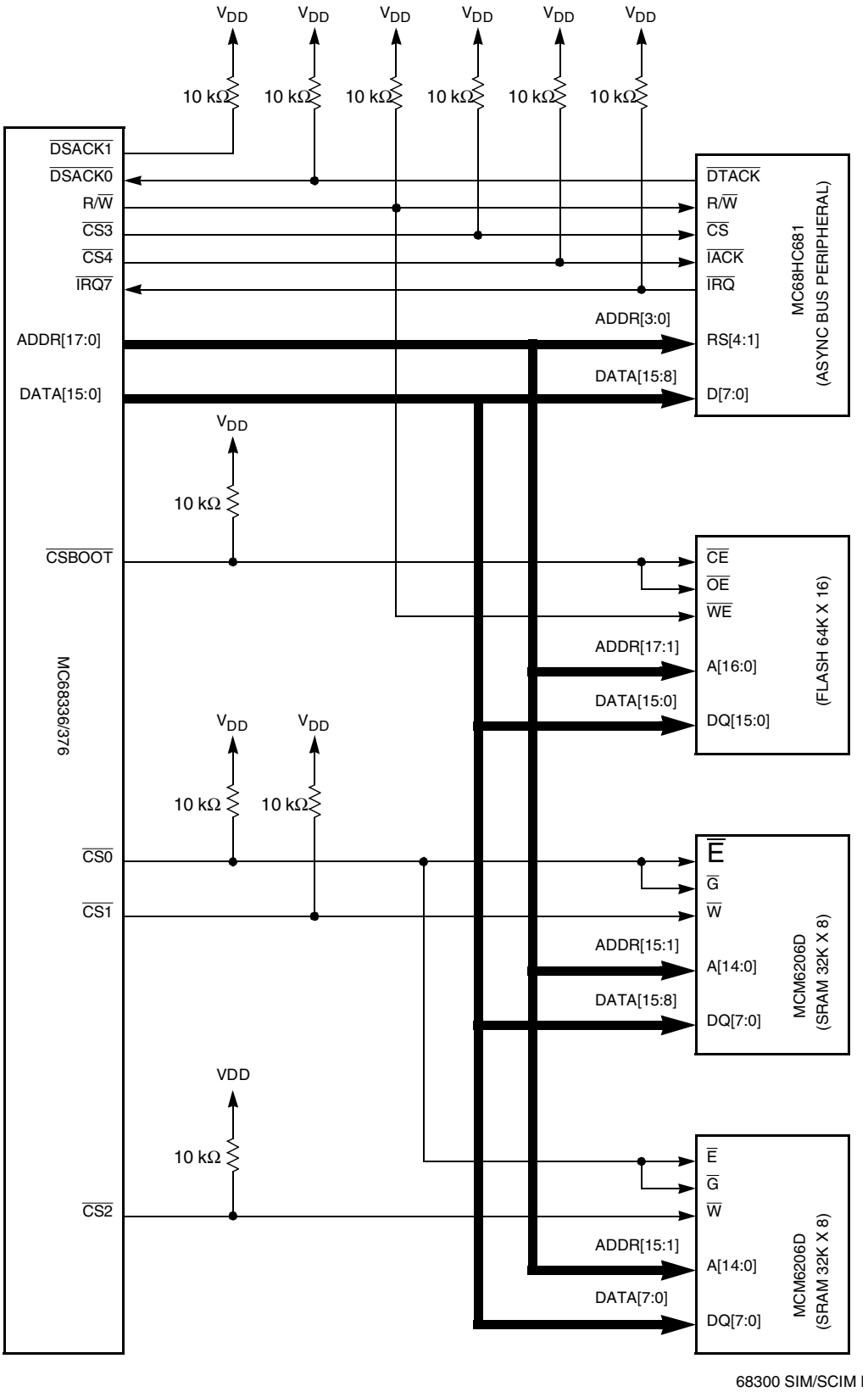


Figure 5-8 MCU Basic System

The external bus has 24 address lines and 16 data lines. The EBI provides dynamic sizing between 8-bit and 16-bit data accesses. It supports byte, word, and long-word transfers. Port width is the maximum number of bits accepted or provided during a bus transfer. Widths of eight and sixteen bits are accessed through the use of asynchronous cycles controlled by the size (SIZ1 and SIZ0) and data and size acknowledge ($\overline{\text{DSACK1}}$ and $\overline{\text{DSACK0}}$) pins. Multiple bus cycles may be required for dynamically sized transfers.

To add flexibility and minimize the necessity for external logic, MCU chip-select logic can be synchronized with EBI transfers. Refer to **5.9 Chip-Selects** for more information.

5.5.1 Bus Control Signals

The address bus provides addressing information to external devices. The data bus transfers 8-bit and 16-bit data between the MCU and external devices. Strobe signals, one for the address bus and another for the data bus, indicate the validity of an address and provide timing information for data.

Control signals indicate the beginning of each bus cycle, the address space it is to take place in, the size of the transfer, and the type of cycle. External devices decode these signals and respond to transfer data and terminate the bus cycle. The EBI operates in an asynchronous mode for any port width.

5.5.1.1 Address Bus

Bus signals ADDR[23:0] 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 $\overline{\text{AS}}$ is asserted.

5.5.1.2 Address Strobe

Address strobe ($\overline{\text{AS}}$) 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.

5.5.1.3 Data Bus

Signals DATA[15:0] form a bidirectional, non-multiplexed parallel bus that transfers data to or from the MCU. A read or write operation can transfer eight or sixteen bits of data 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 $\overline{\text{AS}}$ is asserted in a write cycle.

5.5.1.4 Data Strobe

Data strobe ($\overline{\text{DS}}$) is a timing signal. For a read cycle, the MCU asserts $\overline{\text{DS}}$ to signal an external device to place data on the bus. $\overline{\text{DS}}$ is asserted at the same time as $\overline{\text{AS}}$ during a read cycle. For a write cycle, $\overline{\text{DS}}$ signals an external device that data on the bus is

valid. The MCU asserts $\overline{DS}$ one full clock cycle after the assertion of $\overline{AS}$ during a write cycle.

5.5.1.5 Read/Write Signal

The read/write signal (R/$\overline{W}$) 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.

5.5.1.6 Size Signals

Size signals (SIZ[1:0]) indicate the number of bytes remaining to be transferred during an operand cycle. They are valid while the $\overline{AS}$ is asserted. **Table 5-9** shows SIZ0 and SIZ1 encoding.

Table 5-9 Size Signal Encoding

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

5.5.1.7 Function Codes

The CPU generates function code signals (FC[2:0]) to indicate the type of activity occurring on the data or address bus. These signals can be considered address extensions that can be externally decoded to determine which of eight external address spaces is accessed during a bus cycle.

Address space 7 is designated CPU space. CPU space is used for control information not normally associated with read or write bus cycles. Function codes are valid while $\overline{AS}$ is asserted.

Table 5-10 shows address space encoding.

Table 5-10 Address Space Encoding

FC2	FC1	FC0	Address Space
0	0	0	Reserved
0	0	1	User data space
0	1	0	User program space
0	1	1	Reserved
1	0	0	Reserved
1	0	1	Supervisor data space
1	1	0	Supervisor program space
1	1	1	CPU space

The supervisor bit in the status register determines whether the CPU is operating in supervisor or user mode. Addressing mode and the instruction being executed determine whether a memory access is to program or data space.

5.5.1.8 Data and Size Acknowledge Signals

During normal bus transfers, external devices assert the data and size acknowledge signals ($\overline{\text{DSACK}}$[1:0]) to indicate port width to the MCU. During a read cycle, these signals tell the MCU to terminate the bus cycle and to latch data. During a write cycle, the signals indicate that an external device has successfully stored data and that the cycle can terminate. $\overline{\text{DSACK}}$[1:0] can also be supplied internally by chip-select logic. Refer to **5.9 Chip-Selects** for more information.

5.5.1.9 Bus Error Signal

The bus error signal ($\overline{\text{BERR}}$) is asserted when a bus cycle is not properly terminated by $\overline{\text{DSACK}}$ or $\overline{\text{AVEC}}$ assertion. It can also be asserted in conjunction with $\overline{\text{DSACK}}$ to indicate a bus error condition, provided it meets the appropriate timing requirements. Refer to **5.6.5 Bus Exception Control Cycles** for more information.

The internal bus monitor can generate the $\overline{\text{BERR}}$ signal for internal-to-internal and internal-to-external transfers. In systems with an external bus master, the SIM bus monitor must be disabled and external logic must be provided to drive the $\overline{\text{BERR}}$ pin, because the internal $\overline{\text{BERR}}$ monitor has no information about transfers initiated by an external bus master. Refer to **5.6.6 External Bus Arbitration** for more information.

5.5.1.10 Halt Signal

The halt signal ($\overline{\text{HALT}}$) 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. The $\overline{\text{HALT}}$ signal affects external bus cycles only. As a result, a program not requiring use of the external bus may continue executing, unaffected by the $\overline{\text{HALT}}$ signal.

When the MCU completes a bus cycle with the $\overline{\text{HALT}}$ signal asserted, DATA[15:0] is placed in a high-impedance state and bus control signals are driven inactive; the address, function code, size, and read/write signals remain in the same state. If $\overline{\text{HALT}}$ is still asserted once bus mastership is returned to the MCU, 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. Refer to **5.6.5 Bus Exception Control Cycles** for more information.

5.5.1.11 Autovector Signal

The autovector signal ($\overline{\text{AVEC}}$) can be used to terminate external interrupt acknowledge cycles. Assertion of $\overline{\text{AVEC}}$ causes the CPU32 to generate vector numbers to locate an interrupt handler routine. If $\overline{\text{AVEC}}$ is continuously asserted, autovectors are generated for all external interrupt requests. $\overline{\text{AVEC}}$ is ignored during all other bus cycles. Refer to **5.8 Interrupts** for more information. $\overline{\text{AVEC}}$ for external interrupt requests can also be supplied internally by chip-select logic. Refer to **5.9 Chip-**

Selects for more information. The autovector function is disabled when there is an external bus master. Refer to **5.6.6 External Bus Arbitration** for more information.

5.5.2 Dynamic Bus Sizing

The MCU dynamically interprets the port size of an addressed device during each bus cycle, allowing operand transfers to or from 8-bit and 16-bit ports.

During an operand transfer cycle, an external device signals its port size and indicates completion of the bus cycle to the MCU through the use of the $\overline{\text{DSACK}}$ inputs, as shown in **Table 5-11**. Chip-select logic can generate data and size acknowledge signals for an external device. Refer to **5.9 Chip-Selects** for more information.

Table 5-11 Effect of $\overline{\text{DSACK}}$ Signals

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

If the CPU 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 then 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 $\overline{\text{DSACK}}$ signals to indicate the port width. For instance, a 16-bit device always returns $\overline{\text{DSACK}}$ 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 minimizes the number of bus cycles needed to transfer data and ensures that the MCU transfers valid data.

The MCU always attempts to transfer the maximum amount of data on all bus cycles. For any bus access, it is assumed that the port is 16 bits wide when the bus cycle begins.

Operand bytes are designated as shown in **Figure 5-9**. OP[0:3] represent the order of access. For instance, OP0 is the most significant byte of a long-word operand, and is accessed first, while OP3, the least significant byte, is accessed last. The two bytes of a word-length operand are OP0 (most significant) and OP1. The single byte of a byte-length operand is OP0.

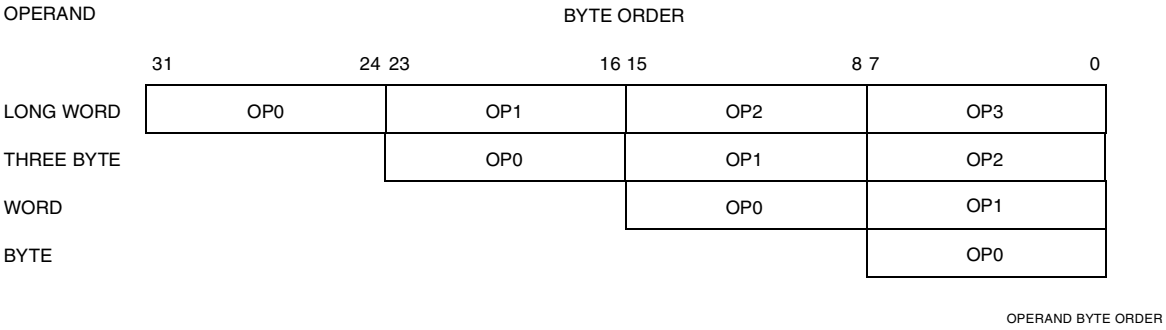


Figure 5-9 Operand Byte Order

5.5.3 Operand Alignment

The EBI 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. Positioning of bytes is determined by the size and address outputs. SIZ1 and SIZ0 indicate the number of bytes remaining to be transferred during the current bus cycle. The number of bytes transferred is equal to or less than the size indicated by SIZ1 and SIZ0, depending on port width.

ADDR0 also affects the operation of the data multiplexer. During an operand transfer, ADDR[23:1] indicate the word base address of the portion of the operand to be accessed. ADDR0 indicates the byte offset from the base.

5.5.4 Misaligned Operands

The CPU32 uses a basic operand size of 16 bits. An operand is misaligned when it overlaps a word boundary. This is determined by the value of ADDR0. When ADDR0 = 0 (an even address), the address is on a word and byte boundary. When ADDR0 = 1 (an odd address), the address is on a byte boundary only. A byte operand is aligned at any address; a word or long-word operand is misaligned at an odd address. The CPU32 does not support misaligned transfers.

The largest amount of data that can be transferred by a single bus cycle is an aligned word. If the MCU transfers a long-word operand through a 16-bit port, the most significant operand word is transferred on the first bus cycle and the least significant operand word is transferred on a following bus cycle.

5.5.5 Operand Transfer Cases

Table 5-12 is a summary of how operands are aligned for various types of transfers. OPn entries are portions of a requested operand that are read or written during a bus cycle and are defined by SIZ1, SIZ0, and ADDR0 for that bus cycle. The following paragraphs discuss all the allowable transfer cases in detail.

Table 5-12 Operand Alignment

Current Cycle	Transfer Case[1]	SIZ1	SIZ0	ADDR0	$\overline{\text{DSACK1}}$	$\overline{\text{DSACK0}}$	DATA [15:8]	DATA [7:0]	Next Cycle
1	Byte to 8-bit port (even)	0	1	0	1	0	OP0	(OP0)[2]	—
2	Byte to 8-bit port (odd)	0	1	1	1	0	OP0	(OP0)	—
3	Byte to 16-bit port (even)	0	1	0	0	1	OP0	(OP0)	—
4	Byte to 16-bit port (odd)	0	1	1	0	1	(OP0)	OP0	—
5	Word to 8-bit port	1	0	0	1	0	OP0	(OP1)	2
6	Word to 16-bit port	1	0	0	0	1	OP0	OP1	—
7	3-Byte to 8-bit port[3]	1	1	1	1	0	OP0	(OP0)	5
8	Long word to 8-bit port	0	0	0	1	0	OP0	(OP0)	7
9	Long word to 16-bit port	0	0	0	0	1	OP0	OP1	6

NOTES:
1. All transfers are aligned. The CPU32 does not support misaligned word or long-word transfers.
2. Operands in parentheses are ignored by the CPU32 during read cycles.
3. Three-Byte transfer cases occur only as a result of a long word to 8-bit port transfer.

5.6 Bus Operation

Internal microcontroller modules are typically accessed in two system clock cycles. Regular external bus cycles use handshaking between the MCU and external peripherals to manage transfer size and data. These accesses take three system clock cycles, with no wait states. During regular cycles, wait states can be inserted as needed by bus control logic. Refer to **5.6.2 Regular Bus Cycles** for more information.

Fast termination cycles, which are two-cycle external accesses with no wait states, use chip-select logic to generate handshaking signals internally. Chip-select logic can also be used to insert wait states before internal generation of handshaking signals. Refer to **5.6.3 Fast Termination Cycles** and **5.9 Chip-Selects** for more information. Bus control signal timing, as well as chip-select signal timing, are specified in **APPENDIX A ELECTRICAL CHARACTERISTICS**. Refer to the ***SIM Reference Manual*** (SIMRM/AD) for more information about each type of bus cycle.

5.6.1 Synchronization to CLKOUT

External devices connected to the MCU bus can operate at a clock frequency different from the frequencies of the MCU as long as the external devices satisfy the interface signal timing constraints. Although bus cycles are classified as asynchronous, they are interpreted relative to the MCU system clock output (CLKOUT).

Descriptions are made in terms of individual system clock states, labeled {S0, S1, S2,..., S_N}. The designation “state” refers to the logic level of the clock signal and does not correspond to any implemented machine state. A clock cycle consists of two successive states. Refer to **Table A-4** for more information.

Bus cycles terminated by $\overline{\text{DSACK}}$ assertion normally require a minimum of three CLKOUT cycles. To support systems that use CLKOUT to generate $\overline{\text{DSACK}}$ and other inputs, asynchronous input setup time and asynchronous input hold times are speci-

fied. When these specifications are met, the MCU is guaranteed to recognize the appropriate signal on a specific edge of the CLKOUT signal.

For a read cycle, when assertion of DSACK is recognized on a particular falling edge of the clock, valid data is latched into the MCU 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.

When a system asserts DSACK for the required window around the falling edge of S2 and obeys the bus protocol by maintaining DSACK and BERR or HALT until and throughout the clock edge that negates AS (with the appropriate asynchronous input hold time), no wait states are inserted. The bus cycle runs at the maximum speed of three clocks per cycle.

To ensure proper operation in a system synchronized to CLKOUT, when either BERR or BERR and HALT is asserted after DSACK, BERR (or BERR and HALT) assertion must satisfy the appropriate data-in setup and hold times before the falling edge of the clock cycle after DSACK is recognized.

5.6.2 Regular Bus Cycles

The following paragraphs contain a discussion of cycles that use external bus control logic. Refer to **5.6.3 Fast Termination Cycles** for information about fast termination cycles.

To initiate a transfer, the MCU asserts an address and the SIZ[1:0] signals. The SIZ signals and ADDR0 are externally decoded to select the active portion of the data bus. Refer to **5.5.2 Dynamic Bus Sizing**. When AS, DS, and R/W are valid, a peripheral device either places data on the bus (read cycle) or latches data from the bus (write cycle), then asserts a DSACK[1:0] combination that indicates port size.

The DSACK[1:0] signals can be asserted before the data from a peripheral device is valid on a read cycle. To ensure valid data is latched into the MCU, a maximum period between DSACK assertion and DS assertion is specified.

There is no specified maximum for the period between the assertion of AS and DSACK. Although the MCU can transfer data in a minimum of three clock cycles when the cycle is terminated with DSACK, the MCU inserts wait cycles in clock period increments until either DSACK signal goes low.

If bus termination signals remain unasserted, the MCU will continue to insert wait states, and the bus cycle will never end. If no peripheral responds to an access, or if an access is invalid, external logic should assert the BERR or HALT signals to abort the bus cycle (when BERR and HALT are asserted simultaneously, the CPU32 acts as though only BERR is asserted). When enabled, the SIM bus monitor asserts BERR when DSACK response time exceeds a predetermined limit. The bus monitor timeout period is determined by the BMT[1:0] field in SYPCR. The maximum bus monitor timeout period is 64 system clock cycles.

5.6.2.1 Read Cycle

During a read cycle, the MCU transfers data from an external 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 portion of the data bus from which each byte is read depends on operand size, peripheral address, and peripheral port size. **Figure 5-10** is a flowchart of a word read cycle. Refer to **5.5.2 Dynamic Bus Sizing**, **5.5.4 Misaligned Operands**, and the ***SIM Reference Manual*** (SIMRM/AD) for more information.

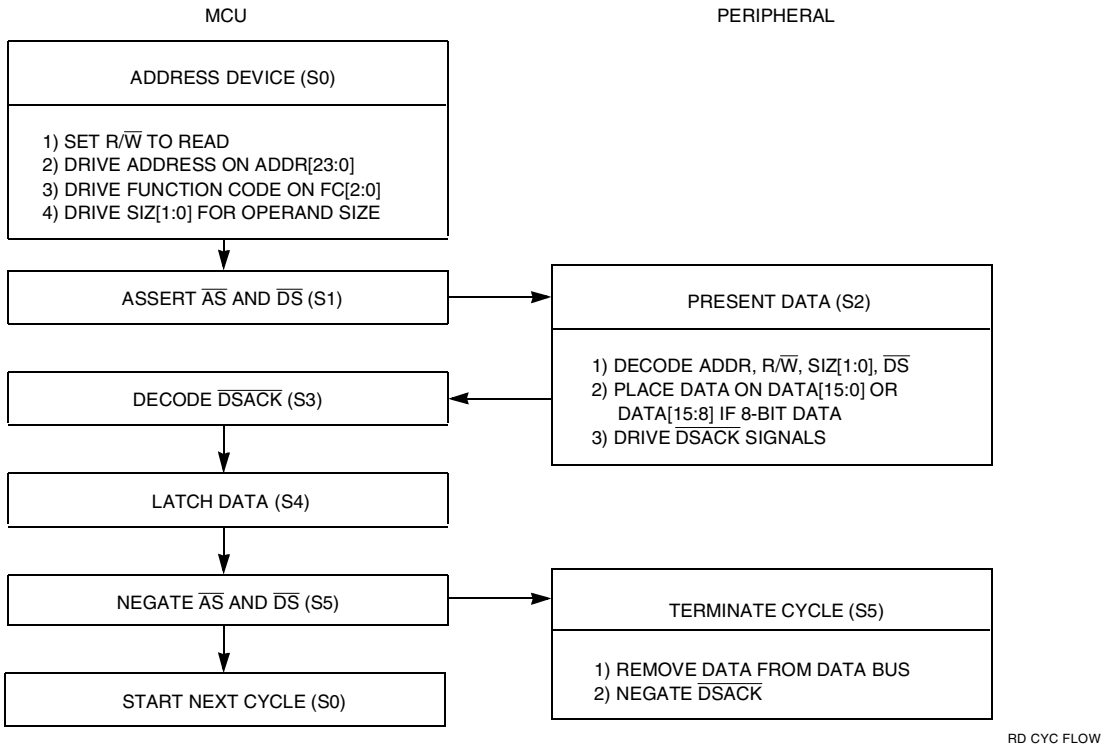


Figure 5-10 Word Read Cycle Flowchart

5.6.2.2 Write Cycle

During a write cycle, the MCU transfers data to an external memory or peripheral device. If the instruction specifies a long-word or word operation, the MCU attempts to write two bytes at once. For a byte operation, the MCU writes one byte. The portion of the data bus upon which each byte is written depends on operand size, peripheral address, and peripheral port size.

Refer to **5.5.2 Dynamic Bus Sizing** and **5.5.4 Misaligned Operands** for more information. **Figure 5-11** is a flowchart of a write-cycle operation for a word transfer. Refer to the ***SIM Reference Manual*** (SIMRM/AD) for more information.

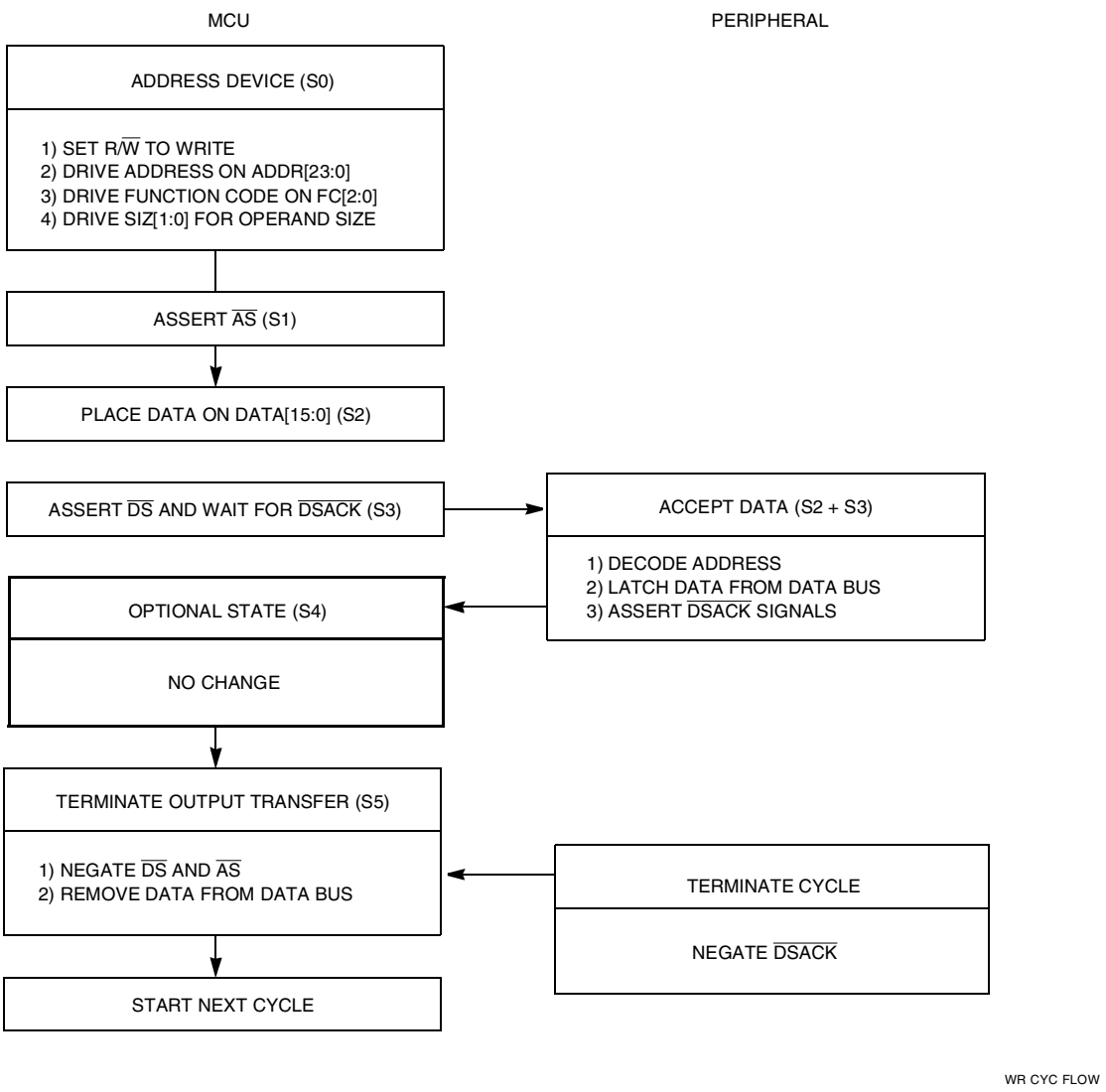


Figure 5-11 Write Cycle Flowchart

5.6.3 Fast Termination Cycles

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

If multiple chip-selects are to be used to provide control signals to a single device and match conditions occur simultaneously, all MODE, STRB, and associated DSACK fields must be programmed to the same value. This prevents a conflict on the internal bus when the wait states are loaded into the DSACK counter shared by all chip-selects.

Fast termination cycles use internal handshaking signals generated by the chip-select logic. To initiate a transfer, the MCU asserts an address and the SIZ[1:0] signals. When AS, DS, and R/W are valid, a peripheral device either places data on the bus (read cycle) or latches data from the bus (write cycle). At the appropriate time, chip-select logic asserts data and size acknowledge signals.

The DSACK option fields in the chip-select option registers determine whether internally generated DSACK or externally generated DSACK is used. The external DSACK lines are always active, regardless of the setting of the DSACK field in the chip-select option registers. Thus, an external DSACK can always terminate a bus cycle. Holding a DSACK line low will cause all external bus cycles to be three-cycle (zero wait states) accesses unless the chip-select option register specifies fast accesses.

For fast termination cycles, the fast termination encoding (%1110) must be used. Refer to **5.9.1 Chip-Select Registers** for information about fast termination setup.

To use fast termination, an external device must be fast enough to have data ready within the specified setup time (for example, by the falling edge of S4). Refer to **Table A-6**, **Figure A-6** and **Figure A-7** for information about fast termination timing.

When fast termination is in use, DS is asserted during read cycles but not during write cycles. The STRB field in the chip-select option register used must be programmed with the address strobe encoding to assert the chip-select signal for a fast termination write.

5.6.4 CPU Space Cycles

Function code signals FC[2:0] designate which of eight external address spaces is accessed during a bus cycle. Address space 7 is designated CPU space. CPU space is used for control information not normally associated with read or write bus cycles. Function codes are valid only while AS is asserted. Refer to **5.5.1.7 Function Codes** for more information on codes and encoding.

During a CPU space access, ADDR[19:16] are encoded to reflect the type of access being made. **Figure 5-12** shows the three encodings used by 68300 family microcontrollers. These encodings represent breakpoint acknowledge (Type $0) cycles, low power stop broadcast (Type $3) cycles, and interrupt acknowledge (Type $F) cycles. Refer to **5.8 Interrupts** for information about interrupt acknowledge bus cycles.

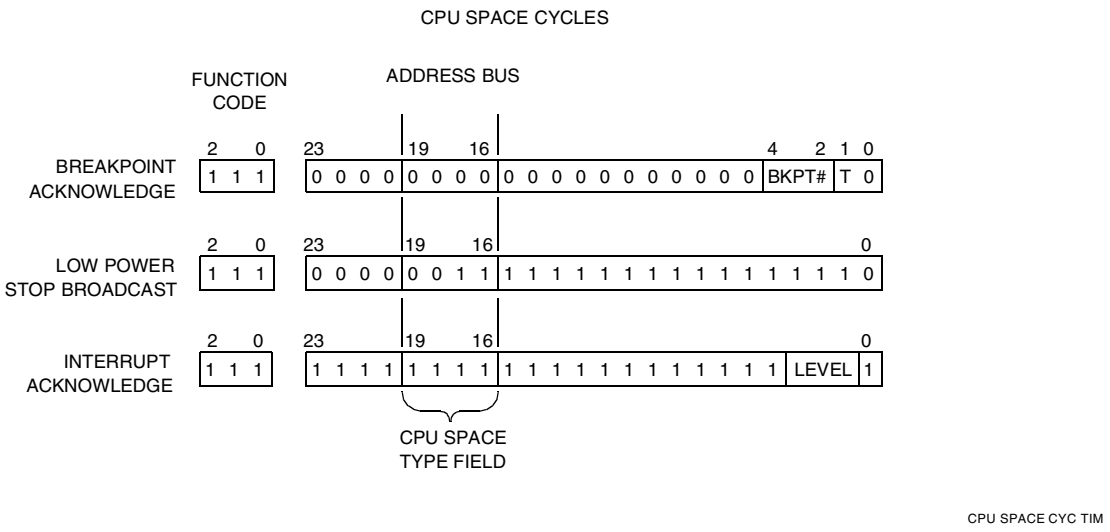


Figure 5-12 CPU Space Address Encoding

5.6.4.1 Breakpoint Acknowledge Cycle

Breakpoints stop program execution at a predefined point during system development. Breakpoints can be used alone or in conjunction with background debug mode. In M68300 microcontrollers, both hardware and software can initiate breakpoints.

The CPU32 $\overline{\text{BKPT}}$ instruction allows the user to insert breakpoints through software. The CPU responds to this instruction by initiating a breakpoint acknowledge read cycle in CPU space. It places the breakpoint acknowledge (%0000) code on ADDR[19:16], the breakpoint number (bits [2:0] of the BKPT opcode) on ADDR[4:2], and %0 (indicating a software breakpoint) on ADDR1.

External breakpoint circuitry decodes the function code and address lines and responds by either asserting $\overline{\text{BERR}}$ or placing an instruction word on the data bus and asserting $\overline{\text{DSACK}}$. If the bus cycle is terminated by $\overline{\text{DSACK}}$, the CPU32 reads the instruction on the data bus and inserts the instruction into the pipeline. (For 8-bit ports, this instruction fetch may require two read cycles.)

If the bus cycle is terminated by $\overline{\text{BERR}}$, the CPU32 then performs illegal instruction exception processing: it acquires the number of the illegal instruction exception vector, computes the vector address from this number, loads the content of the vector address into the PC, and jumps to the exception handler routine at that address.

Assertion of the $\overline{\text{BKPT}}$ input initiates a hardware breakpoint. The CPU32 responds by initiating a breakpoint acknowledge read cycle in CPU space. It places the breakpoint acknowledge code of %0000 on ADDR[19:16], the breakpoint number value of %111 on ADDR[4:2], and ADDR1 is set to %1, indicating a hardware breakpoint.

External breakpoint circuitry decodes the function code and address lines, places an instruction word on the data bus, and asserts $\overline{\text{BERR}}$. The CPU32 then performs hard-

ware breakpoint exception processing: it acquires the number of the hardware breakpoint exception vector, computes the vector address from this number, loads the content of the vector address into the PC, and jumps to the exception handler routine at that address. If the external device asserts $\overline{\text{DSACK}}$ rather than $\overline{\text{BERR}}$, the CPU32 ignores the breakpoint and continues processing.

When $\overline{\text{BKPT}}$ assertion is synchronized with an instruction prefetch, processing of the breakpoint exception occurs at the end of that instruction. The prefetched instruction is "tagged" with the breakpoint when it enters the instruction pipeline. The breakpoint exception occurs after the instruction executes. If the pipeline is flushed before the tagged instruction is executed, no breakpoint occurs. When $\overline{\text{BKPT}}$ assertion is synchronized with an operand fetch, exception processing occurs at the end of the instruction during which $\overline{\text{BKPT}}$ is latched.

Refer to the ***CPU32 Reference Manual*** (CPU32RM/AD) and the ***SIM Reference Manual*** (SIMRM/AD) for additional information. Breakpoint operation flow for the CPU32 is shown in **Figure 5-13**.

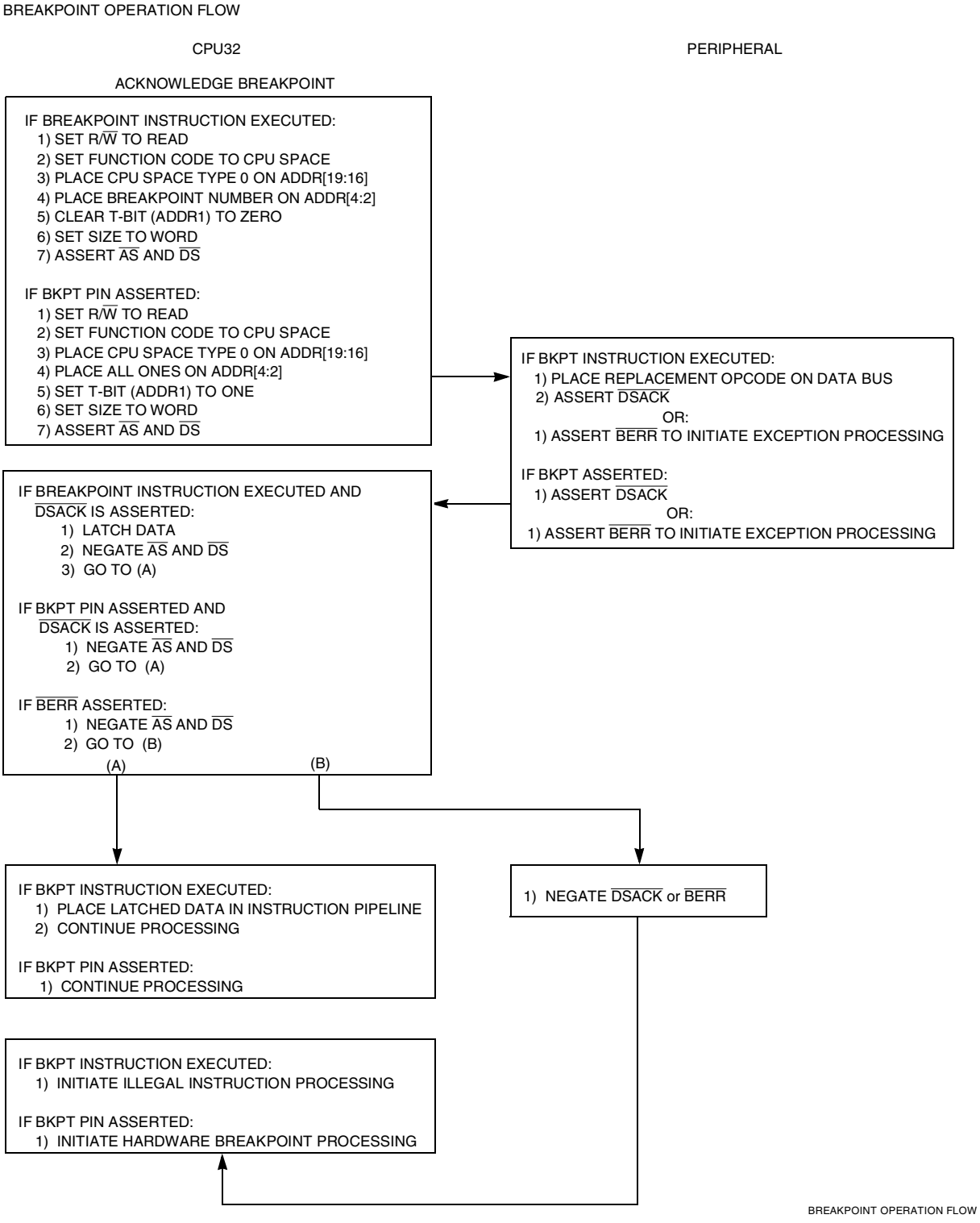


Figure 5-13 Breakpoint Operation Flowchart

5.6.4.2 LPSTOP Broadcast Cycle

Low-power stop mode is initiated by the CPU32. Individual modules can be stopped by setting the STOP bits in each module configuration register, or the SIM can turn off system clocks after execution of the LPSTOP instruction. When the CPU32 executes LPSTOP, an LPSTOP broadcast cycle is generated. The SIM brings the MCU out of low-power stop mode when either an interrupt of higher priority than the stored mask or a reset occurs. Refer to **5.3.4 Low-Power Operation** and **4.8.2.1 Low-Power Stop (LPSTOP)** for more information.

During an LPSTOP broadcast cycle, the CPU32 performs a CPU space write to address $3FFFE. This write puts a copy of the interrupt mask value in the clock control logic. The mask is encoded on the data bus as shown in **Figure 5-14**. The LPSTOP CPU space 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 an internally generated DSACK response to this cycle. The timing of this bus cycle is the same as for a fast termination write cycle. If the bus is not available (arbitrated away), the LPSTOP broadcast cycle is not shown externally.

NOTE

BERR during the LPSTOP broadcast cycle is ignored.

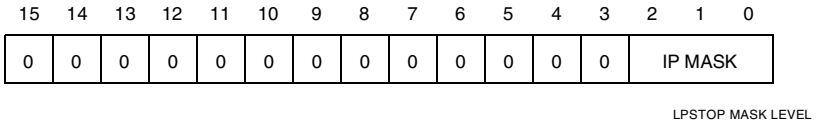


Figure 5-14 LPSTOP Interrupt Mask Level

5.6.5 Bus Exception Control Cycles

An external device or a chip-select circuit must assert at least one of the DSACK[1:0] signals or the AVEC signal to terminate a bus cycle normally. Bus error processing occurs when bus cycles are not terminated in the expected manner. The SIM bus monitor can be used to generate BERR internally, causing a bus error exception to be taken. Bus cycles can also be terminated by assertion of the external BERR or HALT pins signal, or by assertion of the two signals simultaneously.

Acceptable bus cycle termination sequences are summarized as follows. The case numbers refer to **Table 5-13**, which indicates the results of each type of bus cycle termination.

- Normal Termination
  — DSACK is asserted; BERR and HALT remain negated (case 1).
- Halt Termination
  — HALT is asserted at the same time or before DSACK, and BERR remains negated (case 2).
- Bus Error Termination
  — BERR is asserted in lieu of, at the same time as, or before DSACK (case 3),

or after DSACK (case 4), and HALT remains negated; BERR is negated at the same time or after DSACK.

- Retry Termination
  - — HALT and BERR are asserted in lieu of, at the same time as, or before DSACK (case 5) or after DSACK (case 6); BERR is negated at the same time or after DSACK; HALT may be negated at the same time or after BERR.

Table 5-13 shows various combinations of control signal sequences and the resulting bus cycle terminations.

Table 5-13 DSACK, BERR, and HALT Assertion Results

Case Number	Control Signal	Asserted on Rising Edge of State		Result
		N[1]	N + 2	
1	DSACK BERR HALT	A[2] NA[3] NA	S[4] NA X[5]	Normal termination.
2	DSACK BERR HALT	A NA A/S	S NA S	Halt termination: normal cycle terminate and halt. Continue when HALT is negated.
3	DSACK BERR HALT	NA/A A NA	X S X	Bus error termination: terminate and take bus error exception, possibly deferred.
4	DSACK BERR HALT	A A NA	X S NA	Bus error termination: terminate and take bus error exception, possibly deferred.
5	DSACK BERR HALT	NA/A A A/S	X S S	Retry termination: terminate and retry when HALT is negated.
6	DSACK BERR HALT	A NA NA	X A A	Retry termination: terminate and retry when HALT is negated.

NOTES:
1. N = The number of current even bus state (S2, S4, etc.).
2. A = Signal is asserted in this bus state.
3. NA = Signal is not asserted in this state.
4. X = Don't care.
5. S = Signal was asserted in previous state and remains asserted in this state.

To control termination of a bus cycle for a retry or a bus error condition properly, DSACK, BERR, and HALT must be asserted and negated with the rising edge of the MCU clock. This ensures 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 **APPENDIX A ELECTRICAL CHARACTERISTICS** for timing requirements. External circuitry that provides these signals must be designed with these constraints in mind, or else the internal bus monitor must be used.

DSACK, BERR, and HALT may be negated after AS is negated.

WARNING

If $\overline{\text{DSACK}}$ or $\overline{\text{BERR}}$ remain asserted into S2 of the next bus cycle, that cycle may be terminated prematurely.

5.6.5.1 Bus Errors

The CPU32 treats bus errors as a type of exception. Bus error exception processing begins when the CPU32 detects assertion of the IMB $\overline{\text{BERR}}$ signal (by the internal bus monitor or an external source) while the $\overline{\text{HALT}}$ signal remains negated.

$\overline{\text{BERR}}$ assertions do not force immediate exception processing. The signal is synchronized with normal bus cycles and is latched into the CPU32 at the end of the bus cycle in which it was asserted. Because bus cycles can overlap instruction boundaries, bus error exception processing may not occur at the end of the instruction in which the bus cycle begins. Timing of $\overline{\text{BERR}}$ detection/acknowledge is dependent upon several factors:

- Which bus cycle of an instruction is terminated by assertion of $\overline{\text{BERR}}$.
- The number of bus cycles in the instruction during which $\overline{\text{BERR}}$ is asserted.
- The number of bus cycles in the instruction following the instruction in which $\overline{\text{BERR}}$ is asserted.
- Whether $\overline{\text{BERR}}$ is asserted during a program space access or a data space access.

Because of these factors, it is impossible to predict precisely how long after occurrence of a bus error the bus error exception is processed.

CAUTION

The external bus interface does not latch data when an external bus cycle is terminated by a bus error. When this occurs during an instruction prefetch, the IMB precharge state (bus pulled high, or $FF) is latched into the CPU32 instruction register, with indeterminate results.

5.6.5.2 Double Bus Faults

Exception processing for bus error exceptions follows the standard exception processing sequence. Refer to **4.9 Exception Processing** for more information. However, a special case of bus error, called double bus fault, can abort exception processing.

$\overline{\text{BERR}}$ assertion is not detected until an instruction is complete. The $\overline{\text{BERR}}$ latch is cleared by the first instruction of the $\overline{\text{BERR}}$ exception handler. Double bus fault occurs in three ways:

1. When bus error exception processing begins and a second $\overline{\text{BERR}}$ is detected before the first instruction of the exception handler is executed.
2. When one or more bus errors occur before the first instruction after a reset exception is executed.
3. A bus error occurs while the CPU32 is loading information from a bus error stack frame during a return from exception (RTE) instruction.

Multiple bus errors within a single instruction that can generate multiple bus cycles cause a single bus error exception after the instruction has been executed.

Immediately after assertion of a second $\overline{\text{BERR}}$, the MCU halts and drives the $\overline{\text{HALT}}$ line low. Only a reset can restart a halted MCU. However, bus arbitration can still occur. Refer to **5.6.6 External Bus Arbitration** for more information. A bus error or address error that occurs after exception processing has been completed (during the execution of the exception handler routine, or later) does not cause a double bus fault. The MCU continues to retry the same bus cycle as long as the external hardware requests it.

5.6.5.3 Retry Operation

When an external device asserts $\overline{\text{BERR}}$ and $\overline{\text{HALT}}$ during a bus cycle, the MCU enters the retry sequence. A delayed retry can also occur. The MCU terminates the bus cycle, places the $\overline{\text{AS}}$ and $\overline{\text{DS}}$ 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 address, function codes, data (for a write), and control signals. The $\overline{\text{BERR}}$ signal should be negated before S2 of the read cycle to ensure correct operation of the retried cycle.

If $\overline{\text{BR}}$, $\overline{\text{BERR}}$, and $\overline{\text{HALT}}$ are all asserted on the same cycle, the EBI will enter the rerun sequence but first relinquishes the bus to an external master. Once the external master returns the bus and negates $\overline{\text{BERR}}$ and $\overline{\text{HALT}}$, the EBI runs the previous bus cycle. This feature allows an external device to correct the problem that caused the bus error and then try the bus cycle again.

The MCU retries any read or write cycle of an indivisible read-modify-write operation separately. $\overline{\text{RMC}}$ remains asserted during the entire retry sequence. The MCU will not relinquish the bus while $\overline{\text{RMC}}$ is asserted. 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 remain negated). The bus error handler software should examine the read-modify-write bit in the special status word and take the appropriate action to resolve this type of fault when it occurs. Refer to the ***SIM Reference Manual*** (SIMRM/AD) for additional information on read-modify-write and retry operations.

5.6.5.4 Halt Operation

When $\overline{\text{HALT}}$ is asserted while $\overline{\text{BERR}}$ is not asserted, the MCU halts external bus activity after negation of $\overline{\text{DSACK}}$. The MCU may complete the current word transfer in progress. For a long-word to byte transfer, this could be after S2 or S4. For a word to byte transfer, activity ceases after S2.

Negating and reasserting $\overline{\text{HALT}}$ according to timing requirements provides single-step (bus cycle to bus cycle) operation. The $\overline{\text{HALT}}$ signal affects external bus cycles only, so that a program that does not use the external bus can continue executing.

During dynamically-sized 8-bit transfers, external bus activity may not stop at the next cycle boundary. Occurrence of a bus error while $\overline{\text{HALT}}$ is asserted causes the CPU32 to initiate a retry sequence.

When the MCU completes a bus cycle while the HALT signal is asserted, the data bus goes into a high-impedance state and the AS and DS signals are driven to their inactive states. Address, function code, size, and read/write signals remain in the same state.

The halt operation has no effect on bus arbitration. However, when external bus arbitration occurs while the MCU is halted, address and control signals go into a high-impedance state. If HALT is still asserted when the MCU regains control of the bus, address, function code, size, and read/write signals revert to the previous driven states. The MCU cannot service interrupt requests while halted.

5.6.6 External Bus Arbitration

The MCU bus design provides for a single bus master at any one time. Either the MCU or an external device can be master. Bus arbitration protocols determine when an external device can become bus master. Bus arbitration requests are recognized during normal processing, HALT assertion, and when the CPU32 has halted due to a double bus fault.

The bus controller in the MCU manages bus arbitration signals so that the MCU has the lowest priority. External devices that need to obtain the bus must assert 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 protocol sequence is:

1. An external device asserts the bus request signal (BR);
2. The MCU asserts the bus grant signal (BG) to indicate that the bus is available;
3. An external device asserts the bus grant acknowledge (BGACK) signal to indicate that it has assumed bus mastership.

BR can be asserted during a bus cycle or between cycles. BG is asserted in response to BR. To guarantee operand coherency, BG is only asserted at the end of operand transfer. Additionally, BG is not asserted until the end of an indivisible read-modify-write operation (when RMC is negated).

If more than one external device can be bus master, required external arbitration must begin when a requesting device receives BG. An external device must assert BGACK when it assumes mastership, and must maintain BGACK assertion as long as it is bus master.

Two conditions must be met for an external device to assume bus mastership. The device must receive BG through the arbitration process, and BGACK must be inactive, indicating that no other bus master is active. This technique allows the processing of bus requests during data transfer cycles.

BG is negated a few clock cycles after BGACK transition. However, if bus requests are still pending after BG is negated, the MCU asserts BG again within a few clock cycles.

This additional $\overline{BG}$ assertion allows external arbitration circuitry to select the next bus master before the current master has released the bus.

Refer to **Figure 5-15**, which shows bus arbitration for a single device. The flowchart shows $\overline{BR}$ negated at the same time $\overline{BGACK}$ is asserted.

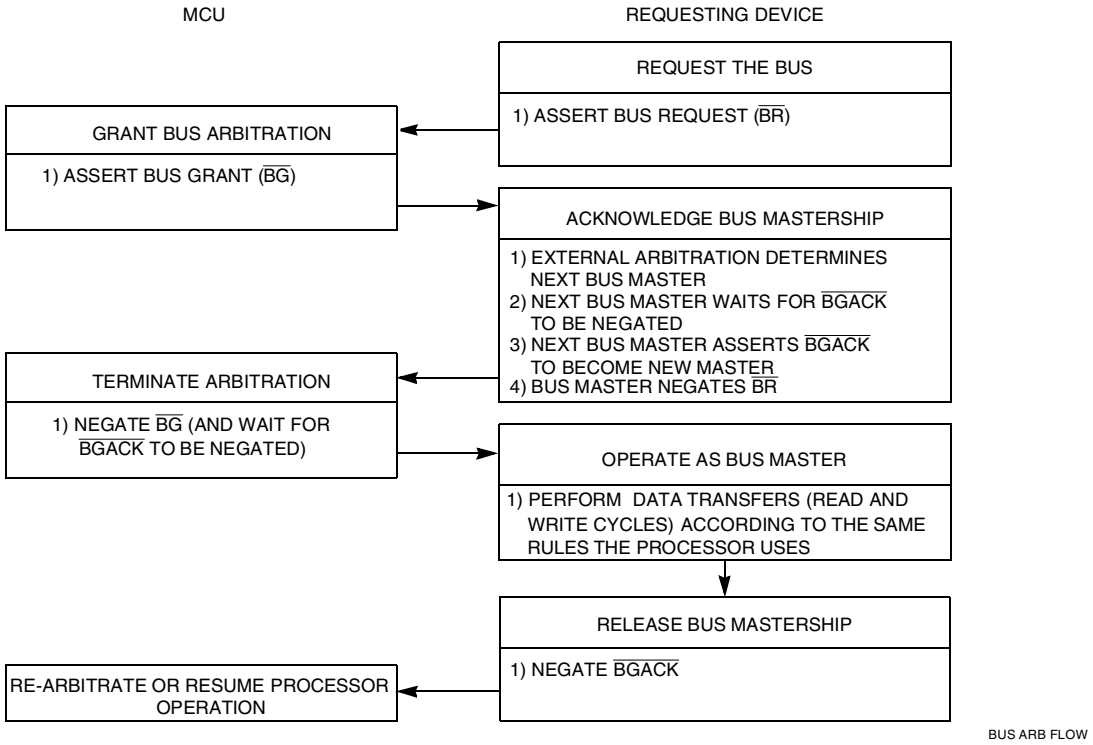


Figure 5-15 Bus Arbitration Flowchart for Single Request

5.6.6.1 Show Cycles

The MCU normally performs internal data transfers without affecting the external bus, but it is possible to show these transfers during debugging. $\overline{AS}$ is not asserted externally during show cycles.

Show cycles are controlled by SHEN[1:0] in SIMCR. This field is set to %00 by reset. When show cycles are disabled, the address bus, function codes, size, and read/write signals reflect internal bus activity, but $\overline{AS}$ and $\overline{DS}$ are not asserted externally and external data bus pins are in high-impedance state during internal accesses. Refer to **5.2.3 Show Internal Cycles** and the ***SIM Reference Manual*** (SIMRM/AD) for more information.

When show cycles are enabled, $\overline{DS}$ is asserted externally during internal cycles, and internal data is driven out on the external data bus. Because internal cycles normally continue to run when the external bus is granted, one SHEN encoding halts internal bus activity while there is an external master.

SIZ[1:0] signals reflect bus allocation during show cycles. Only the appropriate portion of the data bus is valid during the cycle. During a byte write to an internal address, the portion of the bus that represents the byte that is not written reflects internal bus conditions, and is indeterminate. During a byte write to an external address, the data multiplexer in the SIM causes the value of the byte that is written to be driven out on both bytes of the data bus.

5.7 Reset

Reset occurs when an active low logic level on the $\overline{\text{RESET}}$ pin is clocked into the SIM. The $\overline{\text{RESET}}$ input is synchronized to the system clock. If there is no clock when $\overline{\text{RESET}}$ is asserted, reset does not occur until the clock starts. Resets are clocked to allow completion of write cycles in progress at the time $\overline{\text{RESET}}$ is asserted.

Reset procedures handle system initialization and recovery from catastrophic failure. The MCU performs resets with a combination of hardware and software. The SIM determines whether a reset is valid, asserts control signals, performs basic system configuration and boot ROM selection based on hardware mode-select inputs, then passes control to the CPU32.

5.7.1 Reset Exception Processing

The CPU32 processes resets as a type of asynchronous exception. An exception is an event that preempts normal processing, and can be caused by internal or external events. Exception processing makes the transition from normal instruction execution to execution of a routine that deals with an exception. Each exception has an assigned vector that points to an associated handler routine. These vectors are stored in the exception vector table. The exception vector table consists of 256 four-byte vectors and occupies 1024 bytes of address space. The exception vector table can be relocated in memory by changing its base address in the vector base register (VBR). The CPU32 uses vector numbers to calculate displacement into the table. Refer to **4.9 Exception Processing** for more information.

Reset is the highest-priority CPU32 exception. Unlike all other exceptions, a reset occurs at the end of a bus cycle, and not at an instruction boundary. Handling resets in this way prevents write cycles in progress at the time the reset signal is asserted from being corrupted. However, any processing in progress is aborted by the reset exception and cannot be restarted. Only essential reset tasks are performed during exception processing. Other initialization tasks must be accomplished by the exception handler routine. Refer to **5.7.9 Reset Processing Summary** for details on exception processing.

5.7.2 Reset Control Logic

SIM reset control logic determines the cause of a reset, synchronizes reset assertion if necessary to the completion of the current bus cycle, and asserts the appropriate reset lines. Reset control logic can drive four different internal signals:

1. XTRST (external reset) drives the external reset pin.
2. CLKRST (clock reset) resets the clock module.
3. MSTRST (master reset) goes to all other internal circuits.
4. SYSRST (system reset) indicates to internal circuits that the CPU32 has executed a RESET instruction.

All resets are gated by CLKOUT. Resets are classified as synchronous or asynchronous. An asynchronous reset can occur on any CLKOUT edge. Reset sources that cause an asynchronous reset usually indicate a catastrophic failure. As a result, the reset control logic responds by asserting reset to the system immediately. (A system reset, however, caused by the CPU32 RESET instruction, is asynchronous but does not indicate any type of catastrophic failure).

Synchronous resets are timed to occur at the end of bus cycles. The SIM bus monitor is automatically enabled for synchronous resets. When a bus cycle does not terminate normally, the bus monitor terminates it.

Refer to **Table 5-14** for a summary of reset sources.

Table 5-14 Reset Source Summary

Type	Source	Timing	Cause	Reset Lines Asserted by Controller		
External	External	Synch	$\overline{\text{RESET}}$ pin	MSTRST	CLKRST	EXTRST
Power up	EBI	Asynch	V_{DD}	MSTRST	CLKRST	EXTRST
Software watchdog	Monitor	Asynch	Time out	MSTRST	CLKRST	EXTRST
$\overline{\text{HALT}}$	Monitor	Asynch	Internal $\overline{\text{HALT}}$ assertion (e.g. double bus fault)	MSTRST	CLKRST	EXTRST
Loss of clock	Clock	Synch	Loss of reference	MSTRST	CLKRST	EXTRST
Test	Test	Synch	Test mode	MSTRST	—	EXTRST
System	CPU32	Asynch	RESET instruction	—	—	EXTRST

Internal 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 as shown in **Figure 5-16**.

5.7.3 Reset Mode Selection

The logic states of certain data bus pins during reset determine SIM operating configuration. In addition, the state of the MODCLK pin determines system clock source and the state of the $\overline{\text{BKPT}}$ pin determines what happens during subsequent breakpoint assertions. **Table 5-15** is a summary of reset mode selection options.

Table 5-15 Reset Mode Selection

Mode Select Pin	Default Function (Pin Left High)	Alternate Function (Pin Pulled Low)
DATA0	$\overline{\text{CSBOOT}}$ 16-bit	$\overline{\text{CSBOOT}}$ 8-bit
DATA1	$\overline{\text{CS0}}$ $\overline{\text{CS1}}$ $\overline{\text{CS2}}$	$\overline{\text{BR}}$ $\overline{\text{BG}}$ $\overline{\text{BGACK}}$
DATA2	$\overline{\text{CS3}}$ $\overline{\text{CS4}}$ $\overline{\text{CS5}}$	FC0 FC1 FC2
DATA3 DATA4 DATA5 DATA6 DATA7	$\overline{\text{CS6}}$ $\overline{\text{CS}}$[7:6] $\overline{\text{CS}}$[8:6] $\overline{\text{CS}}$[9:6] $\overline{\text{CS}}$[10:6]	ADDR19 ADDR[20:19] ADDR[21:19] ADDR[22:19] ADDR[23:19]
DATA8	$\overline{\text{DSACK}}$[1:0], $\overline{\text{AVEC}}$, $\overline{\text{DS}}$, $\overline{\text{AS}}$, $\overline{\text{SIZ}}$[1:0]	PORTE
DATA9	$\overline{\text{IRQ}}$[7:1] MODCLK	PORTF
DATA11	Normal operation[1]	Reserved
MODCLK	VCO = System clock	EXTAL = System clock
$\overline{\text{BKPT}}$	Background mode disabled	Background mode enabled

NOTES:
1. The DATA11 bus must remain high during reset to ensure normal operation.

5.7.3.1 Data Bus Mode Selection

All data lines have weak internal pull-up drivers. When pins are held high by the internal drivers, the MCU uses a default operating configuration. However, specific lines can be held low externally during reset to achieve an alternate configuration.

NOTE

External bus loading can overcome the weak internal pull-up drivers on data bus lines and hold pins low during reset.

Use an active device to hold data bus lines low. Data bus configuration logic must release the bus before the first bus cycle after reset to prevent conflict with external memory devices. The first bus cycle occurs ten CLKOUT cycles after $\overline{\text{RESET}}$ is released. If external mode selection logic causes a conflict of this type, an isolation resistor on the driven lines may be required. **Figure 5-16** shows a recommended method for conditioning the mode select signals.

The mode configuration drivers are conditioned with R/$\overline{\text{W}}$ and $\overline{\text{DS}}$ to prevent conflicts between external devices and the MCU when reset is asserted. If external $\overline{\text{RESET}}$ is asserted during an external write cycle, R/$\overline{\text{W}}$ conditioning (as shown in **Figure 5-16**) prevents corruption of the data during the write. Similarly, $\overline{\text{DS}}$ conditions the mode configuration drivers so that external reads are not corrupted when $\overline{\text{RESET}}$ is asserted during an external read cycle.

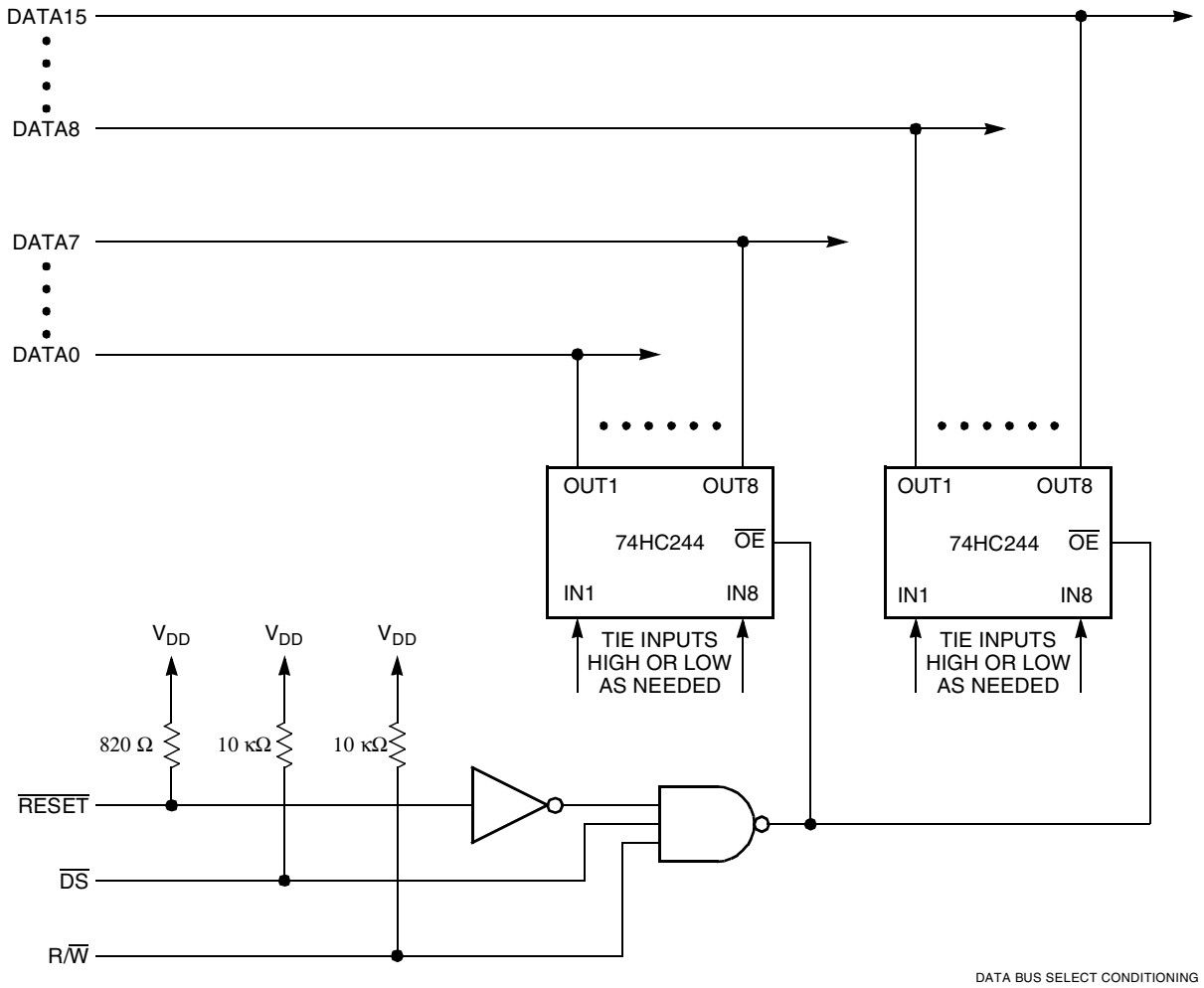


Figure 5-16 Preferred Circuit for Data Bus Mode Select Conditioning

Alternate methods can be used for driving data bus pins low during reset. **Figure 5-17** shows two of these options. The simplest is to connect a resistor in series with a diode from the data bus pin to the $\overline{\text{RESET}}$ line. A bipolar transistor can be used for the same purpose, but an additional current limiting resistor must be connected between the base of the transistor and the $\overline{\text{RESET}}$ pin. If a MOSFET is substituted for the bipolar transistor, only the 1 kΩ isolation resistor is required. These simpler circuits do not offer the protection from potential memory corruption during $\overline{\text{RESET}}$ assertion as does the circuit shown in **Figure 5-16**.

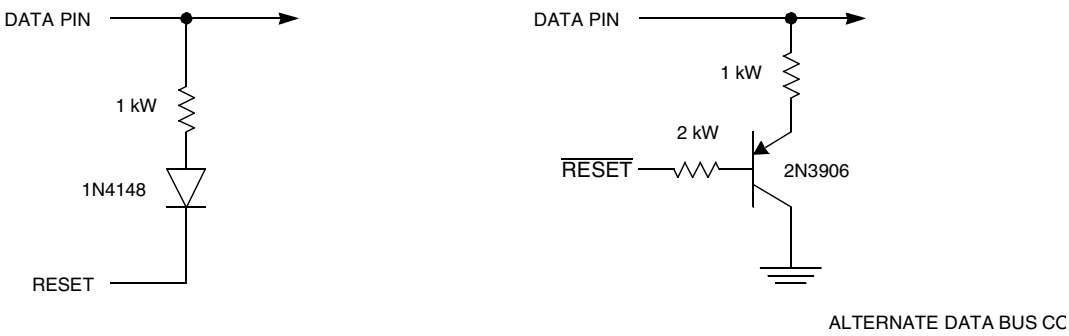


Figure 5-17 Alternate Circuit for Data Bus Mode Select Conditioning

Data bus mode select current is specified in **Table A-5**. Do not confuse pin function with pin electrical state. Refer to **5.7.5 Pin States During Reset** for more information.

Unlike other chip-select signals, the boot ROM chip-select ($\overline{\text{CSBOOT}}$) is active at the release of $\overline{\text{RESET}}$. During reset exception processing, the MCU fetches initialization vectors beginning at address $000000 in supervisor program space. An external memory device containing vectors located at these addresses can be enabled by $\overline{\text{CSBOOT}}$ after a reset.

The logic level of DATA0 during reset selects boot ROM port size for dynamic bus allocation. When DATA0 is held low, port size is eight bits; when DATA0 is held high, either by the weak internal pull-up driver or by an external pull-up, port size is 16 bits. Refer to **5.9.4 Chip-Select Reset Operation** for more information.

DATA1 and DATA2 determine the functions of $\overline{\text{CS}}$[2:0] and $\overline{\text{CS}}$[5:3], respectively. DATA[7:3] determine the functions of an associated chip-select and all lower-numbered chip-selects down through $\overline{\text{CS6}}$. For example, if DATA5 is pulled low during reset, $\overline{\text{CS}}$[8:6] are assigned alternate function as ADDR[21:19], and $\overline{\text{CS}}$[10:9] remain chip-selects. Refer to **5.9.4 Chip-Select Reset Operation** for more information.

DATA8 determines the function of the $\overline{\text{DSACK}}$[1:0], $\overline{\text{AVEC}}$, $\overline{\text{DS}}$, $\overline{\text{AS}}$, and SIZE pins. If DATA8 is held low during reset, these pins are assigned to I/O port E.

DATA9 determines the function of interrupt request pins $\overline{\text{IRQ}}$[7:1] and the clock mode select pin (MODCLK). When DATA9 is held low during reset, these pins are assigned to I/O port F.

5.7.3.2 Clock Mode Selection

The state of the clock mode (MODCLK) pin during reset determines what clock source the MCU uses. When MODCLK is held high during reset, the clock signal is generated from a reference frequency using the clock synthesizer. When MODCLK is held low during reset, the clock synthesizer is disabled, and an external system clock signal must be applied. Refer to **5.3 System Clock** for more information.

NOTE

The MODCLK pin can also be used as parallel I/O pin PF0. To prevent inadvertent clock mode selection by logic connected to port F, use an active device to drive MODCLK during reset.

5.7.3.3 Breakpoint Mode Selection

Background debug mode (BDM) is enabled when the breakpoint ($\overline{\text{BKPT}}$) pin is sampled at a logic level zero at the release of $\overline{\text{RESET}}$. Subsequent assertion of the $\overline{\text{BKPT}}$ pin or the internal breakpoint signal (for instance, the execution of the CPU32 BKPT instruction) will place the CPU32 in BDM.

If $\overline{\text{BKPT}}$ is sampled at a logic level one at the rising edge of $\overline{\text{RESET}}$, BDM is disabled. Assertion of the $\overline{\text{BKPT}}$ pin or execution of the execution of the BKPT instruction will result in normal breakpoint exception processing.

BDM remains enabled until the next system reset. $\overline{\text{BKPT}}$ is relatched on each rising transition of $\overline{\text{RESET}}$. $\overline{\text{BKPT}}$ is internally synchronized and must be held low for at least two clock cycles prior to $\overline{\text{RESET}}$ negation for BDM to be enabled. $\overline{\text{BKPT}}$ assertion logic must be designed with special care. If $\overline{\text{BKPT}}$ assertion extends into the first bus cycle following the release of $\overline{\text{RESET}}$, the bus cycle could inadvertently be tagged with a breakpoint.

Refer to **4.10.2 Background Debug Mode** and the ***CPU32 Reference Manual*** (CPU32RM/AD) for more information on background debug mode. Refer to the ***SIM Reference Manual*** (SIMRM/AD) and **APPENDIX A ELECTRICAL CHARACTERISTICS** for more information concerning BKPT signal timing.

5.7.4 MCU Module Pin Function During Reset

Usually, module pins default to port functions and input/output ports are set to the input state. This is accomplished by disabling pin functions in the appropriate control registers, and by clearing the appropriate port data direction registers. Refer to individual module sections in this manual for more information. **Table 5-16** is a summary of module pin function out of reset.

Table 5-16 Module Pin Functions During Reset

Module	Pin Mnemonic	Function
CPU32	DSI/$\overline{\text{IFETCH}}$	DSI/$\overline{\text{IFETCH}}$
	DSO/$\overline{\text{IPIPE}}$	DSO/$\overline{\text{IPIPE}}$
	$\overline{\text{BKPT}}$/DSCLK	$\overline{\text{BKPT}}$/DSCLK
CTM4	CPWM[8:5]	Discrete output
	CTD[10:9]/[4:3]	Discrete input
	CTM4C	Discrete input
QADC	PQA[7:5]/AN[59:57]	Discrete input
	PQA[4:3]/AN[56:55]/ETRIG[2:1]	Discrete input
	PQA[2:0]/AN[54:52]/MA[2:0]	Discrete input
	PQB[7:4]/AN[51:48]	Discrete input
	PQB[3:0]/AN[z, y, x, w]/AN[3:0]	Discrete input
QSM	PQS0/MISO	Discrete input
	PQS1/MOSI	Discrete input
	PQS2/SCK	Discrete input
	PQS3/PCS0/$\overline{\text{SS}}$	Discrete input
	PQS[6:4]/PCS[3:1]	RXD
	PQS7/TXD	Discrete input
	RXD	Discrete input
TouCAN (MC68376 only)	CANRX0	TouCAN receive
	CANTX0	TouCAN transmit
TPU	TPUCH[15:0]	TPU input
	T2CLK	TCR2 clock

5.7.5 Pin States During Reset

It is important to keep the distinction between pin function and pin electrical state clear. Although control register values and mode select inputs determine pin function, a pin driver can be active, inactive or in high-impedance state while reset occurs. During power-on reset, pin state is subject to the constraints discussed in **5.7.7 Power-On Reset**.

NOTE

Pins that are not used should either be configured as outputs, or (if configured as inputs) pulled to the appropriate inactive state. This decreases additional I_{DD} caused by digital inputs floating near mid-supply level.

5.7.5.1 Reset States of SIM Pins

Generally, while $\overline{\text{RESET}}$ is asserted, SIM pins either go to an inactive high-impedance state or are driven to their inactive states. After $\overline{\text{RESET}}$ is released, mode selection occurs and reset exception processing begins. Pins configured as inputs must be driven to the desired active state. Pull-up or pull-down circuitry may be necessary. Pins

configured as outputs begin to function after $\overline{\text{RESET}}$ is released. **Table 5-17** is a summary of SIM pin states during reset.

Table 5-17 SIM Pin Reset States

Pin(s)	Pin State While $\overline{\text{RESET}}$ Asserted	Pin State After $\overline{\text{RESET}}$ Released			
		Default Function		Alternate Function	
		Pin Function	Pin State	Pin Function	Pin State
$\overline{\text{CS10}}$/ADDR23/ECLK	V_{DD}	$\overline{\text{CS10}}$	V_{DD}	ADDR23	Unknown
$\overline{\text{CS}}$[9:6]/ADDR[22:19]/PC[6:3]	V_{DD}	$\overline{\text{CS}}$[9:6]	V_{DD}	ADDR[22:19]	Unknown
ADDR[18:0]	High-Z	ADDR[18:0]	Unknown	ADDR[18:0]	Unknown
$\overline{\text{AS}}$/PE5	High-Z	$\overline{\text{AS}}$	Output	PE5	Input
$\overline{\text{AVEC}}$/PE2	High-Z	$\overline{\text{AVEC}}$	Input	PE2	Input
$\overline{\text{BERR}}$	High-Z	$\overline{\text{BERR}}$	Input	$\overline{\text{BERR}}$	Input
$\overline{\text{CS1}}$/$\overline{\text{BG}}$	V_{DD}	$\overline{\text{CS1}}$	V_{DD}	$\overline{\text{BG}}$	V_{DD}
$\overline{\text{CS2}}$/$\overline{\text{BGACK}}$	V_{DD}	$\overline{\text{CS2}}$	V_{DD}	$\overline{\text{BGACK}}$	Input
$\overline{\text{CS0}}$/$\overline{\text{BR}}$	V_{DD}	$\overline{\text{CS0}}$	V_{DD}	$\overline{\text{BR}}$	Input
CLKOUT	Output	CLKOUT	Output	CLKOUT	Output
$\overline{\text{CSBOOT}}$	V_{DD}	$\overline{\text{CSBOOT}}$	V_{SS}	$\overline{\text{CSBOOT}}$	V_{SS}
DATA[15:0]	Mode select	DATA[15:0]	Input	DATA[15:0]	Input
$\overline{\text{DS}}$/PE4	High-Z	$\overline{\text{DS}}$	Output	PE4	Input
$\overline{\text{DSACK0}}$/PE0	High-Z	$\overline{\text{DSACK0}}$	Input	PE0	Input
$\overline{\text{DSACK1}}$/PE1	High-Z	$\overline{\text{DSACK1}}$	Input	PE1	Input
$\overline{\text{CS}}$[5:3]/FC[2:0]/PC[2:0]	V_{DD}	$\overline{\text{CS}}$[5:3]	V_{DD}	FC[2:0]	Unknown
$\overline{\text{HALT}}$	High-Z	$\overline{\text{HALT}}$	Input	$\overline{\text{HALT}}$	Input
$\overline{\text{IRQ}}$[7:1]/PF[7:1]	High-Z	$\overline{\text{IRQ}}$[7:1]	Input	PF[7:1]	Input
MODCLK/PF0	Mode Select	MODCLK	Input	PF0	Input
R/$\overline{\text{W}}$	High-Z	R/$\overline{\text{W}}$	Output	R/$\overline{\text{W}}$	Output
$\overline{\text{RESET}}$	Asserted	$\overline{\text{RESET}}$	Input	$\overline{\text{RESET}}$	Input
$\overline{\text{RMC}}$/PE3	High-Z	$\overline{\text{RMC}}$	Output	PE3	Input
SIZ[1:0]/PE[7:6]	High-Z	SIZ[1:0]	Unknown	PE[7:6]	Input
$\overline{\text{TSTME}}$/TSC	Mode select	TSC	Input	TSC	Input

5.7.5.2 Reset States of Pins Assigned to Other MCU Modules

As a rule, module pins that are assigned to general-purpose I/O ports go into a high-impedance state following reset. Other pin states are determined by individual module control register settings. Refer to sections concerning modules for details. However, during power-on reset, module port pins may be in an indeterminate state for a short period. Refer to **5.7.7 Power-On Reset** for more information.

5.7.6 Reset Timing

The $\overline{\text{RESET}}$ input must be asserted for a specified minimum period for reset to occur. External $\overline{\text{RESET}}$ assertion can be delayed internally for a period equal to the longest bus cycle time (or the bus monitor time-out period) in order to protect write cycles from

being aborted by reset. While $\overline{\text{RESET}}$ is asserted, SIM pins are either in an inactive, high-impedance state or are driven to their inactive states.

When an external device asserts $\overline{\text{RESET}}$ for the proper period, reset control logic clocks the signal into an internal latch. The control logic drives the $\overline{\text{RESET}}$ pin low for an additional 512 CLKOUT cycles after it detects that the $\overline{\text{RESET}}$ signal is no longer being externally driven to guarantee this length of reset to the entire system.

If an internal source asserts a reset signal, the reset control logic asserts the $\overline{\text{RESET}}$ pin for a minimum of 512 cycles. If the reset signal is still asserted at the end of 512 cycles, the control logic continues to assert the $\overline{\text{RESET}}$ pin until the internal reset signal is negated.

After 512 cycles have elapsed, the $\overline{\text{RESET}}$ pin goes to an inactive, high-impedance state for ten cycles. At the end of this 10-cycle period, the $\overline{\text{RESET}}$ input is tested. When the input is at logic level one, reset exception processing begins. If, however, the $\overline{\text{RESET}}$ input is at logic level zero, reset control logic drives the pin low for another 512 cycles. At the end of this period, the pin again goes to high-impedance state for ten cycles, then it is tested again. The process repeats until $\overline{\text{RESET}}$ is released.

5.7.7 Power-On Reset

When the SIM clock synthesizer is used to generate system clocks, power-on reset involves special circumstances related to application of system and clock synthesizer power. Regardless of clock source, voltage must be applied to the clock synthesizer power input pin V_{DDSYN} for the MCU to operate. The following discussion assumes that V_{DDSYN} is applied before and during reset, which minimizes crystal start-up time. When V_{DDSYN} is applied at power-on, start-up time is affected by specific crystal parameters and by oscillator circuit design. V_{DD} ramp-up time also affects pin state during reset. Refer to **APPENDIX A ELECTRICAL CHARACTERISTICS** for voltage and timing specifications.

During power-on reset, an internal circuit in the SIM drives the IMB internal (MSTRST) and external (EXTRST) reset lines. The power-on reset circuit releases the internal reset line as V_{DD} ramps up to the minimum operating voltage, and SIM pins are initialized to the values shown in **Table 5-17**. When V_{DD} reaches the minimum operating voltage, the clock synthesizer VCO begins operation. Clock frequency ramps up to specified limp mode frequency (f_{limp}). The external $\overline{\text{RESET}}$ line remains asserted until the clock synthesizer PLL locks and 512 CLKOUT cycles elapse.

The SIM clock synthesizer provides clock signals to the other MCU modules. After the clock is running and MSTRST is asserted for at least four clock cycles, these modules reset. V_{DD} ramp time and VCO frequency ramp time determine how long the four cycles take. Worst case is approximately 15 milliseconds. During this period, module port pins may be in an indeterminate state. While input-only pins can be put in a known state by external pull-up resistors, external logic on input/output or output-only pins during this time must condition the lines. Active drivers require high-impedance buffers or isolation resistors to prevent conflict.

Figure 5-18 is a timing diagram for power-on reset. It shows the relationships between $\overline{\text{RESET}}$, V_{DD}, and bus signals.

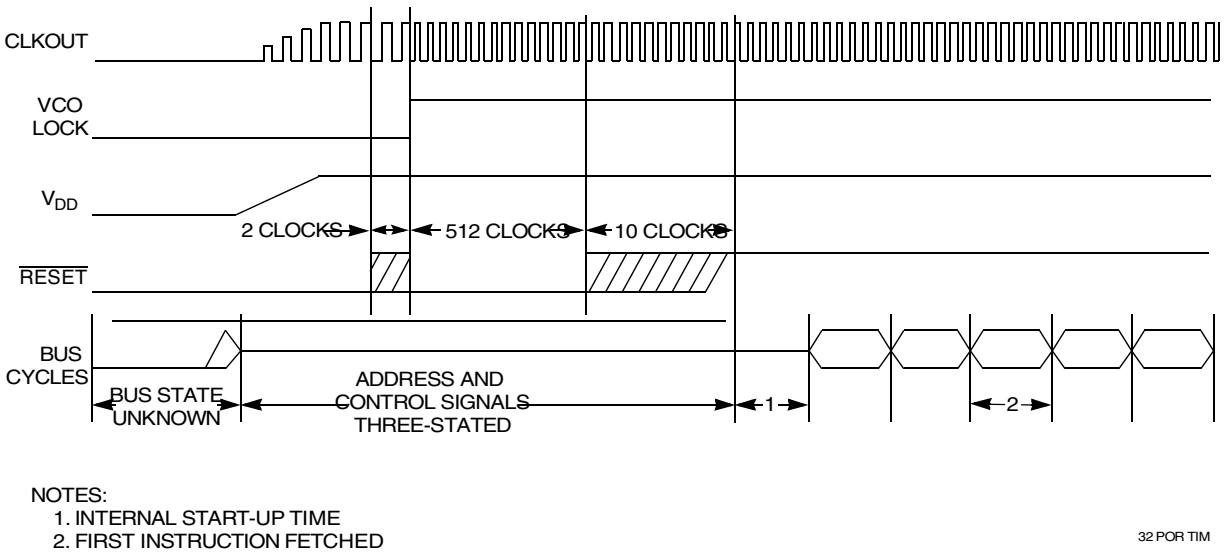


Figure 5-18 Power-On Reset

5.7.8 Use of the Three-State Control Pin

Asserting the three-state control (TSC) input causes the MCU to put all output drivers in a disabled, high-impedance state. The signal must remain asserted for approximately ten clock cycles in order for drivers to change state.

When the internal clock synthesizer is used (MODCLK held high during reset), synthesizer ramp-up time affects how long the ten cycles take. Worst case is approximately 20 milliseconds from TSC assertion.

When an external clock signal is applied (MODCLK held low during reset), pins go to high-impedance state as soon after TSC assertion as approximately ten clock pulses have been applied to the EXTAL pin.

NOTE

When TSC assertion takes effect, internal signals are forced to values that can cause inadvertent mode selection. Once the output drivers change state, the MCU must be powered down and restarted before normal operation can resume.

5.7.9 Reset Processing Summary

To prevent write cycles in progress from being corrupted, a reset is recognized at the end of a bus cycle instead of at an instruction boundary. Any processing in progress at the time a reset occurs is aborted. After SIM reset control logic has synchronized an internal or external reset request, the MSTRST signal is asserted.

The following events take place when MSTRST is asserted:

A. Instruction execution is aborted.
B. The status register is initialized.
   1. The T0 and T1 bits are cleared to disable tracing.
   2. The S bit is set to establish supervisor privilege level.
   3. The interrupt priority mask is set to $7, disabling all interrupts below priority 7.
C. The vector base register is initialized to $000000.

The following events take place when MSTRST is negated after assertion.

A. The CPU32 samples the $\overline{\text{BKPT}}$ input.
B. The CPU32 fetches the reset vector:
   1. The first long word of the vector is loaded into the interrupt stack pointer.
   2. The second long word of the vector is loaded into the program counter.
   3. Vectors can be fetched from external ROM enabled by the $\overline{\text{CSBOOT}}$ signal.
C. The CPU32 fetches and begins decoding the first instruction to be executed.

5.7.10 Reset Status Register

The reset status register (RSR) contains a bit for each reset source in the MCU. When a reset occurs, a bit corresponding to the reset type is set. When multiple causes of reset occur at the same time, only one bit in RSR may be set. The reset status register is updated by the reset control logic when the $\overline{\text{RESET}}$ signal is released. Refer to **D.2.4 Reset Status Register** for more information.

5.8 Interrupts

Interrupt recognition and servicing involve complex interaction between the SIM, the CPU32, and a device or module requesting interrupt service.

The following paragraphs provide an overview of the entire interrupt process. Chip-select logic can also be used to terminate the IACK cycle with either $\overline{\text{AVEC}}$ or $\overline{\text{DSACK}}$. Refer to **5.9 Chip-Selects** for more information.

5.8.1 Interrupt Exception Processing

The CPU32 processes interrupts as a type of asynchronous exception. An exception is an event that preempts normal processing. Each exception has an assigned vector in an exception vector table that points to an associated handler routine. The CPU32 uses vector numbers to calculate displacement into the table. During exception processing, the CPU fetches the appropriate vector and executes the exception handler routine to which the vector points.

At the release of reset, the exception vector table is located beginning at address $000000. This value can be changed by programming the vector base register (VBR) with a new value. Multiple vector tables can be used. Refer to **4.9 Exception Processing** for more information.

5.8.2 Interrupt Priority and Recognition

The CPU32 provides seven levels of interrupt priority (1-7), seven automatic interrupt vectors, and 200 assignable interrupt vectors. All interrupts with priorities less than seven can be masked by the interrupt priority (IP) field in status register.

NOTE

Exceptions such as “address error” are not interrupts and have no “level” associated. Exceptions cannot ever be masked.

There are seven interrupt request signals ($\overline{\text{IRQ}}$[7:1]). These signals are used internally on the IMB, and have corresponding pins for external interrupt service requests. The CPU32 treats all interrupt requests as though they come from internal modules; external interrupt requests are treated as interrupt service requests from the SIM. Each of the interrupt request signals corresponds to an interrupt priority. $\overline{\text{IRQ1}}$ has the lowest priority and $\overline{\text{IRQ7}}$ the highest.

Interrupt recognition is determined by interrupt priority level and interrupt priority (IP) mask value. The interrupt priority mask consists of three bits in the CPU32 status register. Binary values %000 to %111 provide eight priority masks. Masks prevent an interrupt request of a priority less than or equal to the mask value from being recognized and processed. $\overline{\text{IRQ7}}$, however, is always recognized, even if the mask value is %111.

$\overline{\text{IRQ}}$[7:1] are active-low level-sensitive inputs. The low on the pin must remain asserted until an interrupt acknowledge cycle corresponding to that level is detected.

$\overline{\text{IRQ7}}$ is transition-sensitive as well as level-sensitive: a level-7 interrupt is not detected unless a falling edge transition is detected on the $\overline{\text{IRQ7}}$ line. This prevents redundant servicing and stack overflow. A non-maskable interrupt is generated each time $\overline{\text{IRQ7}}$ is asserted as well as each time the priority mask is written while $\overline{\text{IRQ7}}$ is asserted. If $\overline{\text{IRQ7}}$ is asserted and the IP mask is written to any new value (including %111), $\overline{\text{IRQ7}}$ will be recognized as a new $\overline{\text{IRQ7}}$.

Interrupt requests are sampled on consecutive falling edges of the system clock. Interrupt request input circuitry has hysteresis. To be valid, a request signal must be asserted for at least two consecutive clock periods. Valid requests do not cause immediate exception processing, but are left pending. Pending requests are processed at instruction boundaries or when exception processing of higher-priority interrupts is complete.

The CPU32 does not latch the priority of a pending interrupt request. If an interrupt source of higher priority makes a service request while a lower priority request is pending, the higher priority request is serviced. If an interrupt request with a priority equal to or lower than the current IP mask value is made, the CPU32 does not recognize the occurrence of the request. If simultaneous interrupt requests of different priorities are made, and both have a priority greater than the mask value, the CPU32 recognizes the higher-level request.

5.8.3 Interrupt Acknowledge and Arbitration

When the CPU32 detects one or more interrupt requests of a priority higher than the interrupt priority mask value, it places the interrupt request level on the address bus and initiates a CPU space read cycle. The request level serves two purposes: it is decoded by modules or external devices that have requested interrupt service, to determine whether the current interrupt acknowledge cycle pertains to them, and it is latched into the interrupt priority mask field in the CPU32 status register to preclude further interrupts of lower priority during interrupt service.

Modules or external devices that have requested interrupt service must decode the IP mask value placed on the address bus during the interrupt acknowledge cycle and respond if the priority of the service request corresponds to the mask value. However, before modules or external devices respond, interrupt arbitration takes place.

Arbitration is performed by means of serial contention between values stored in individual module interrupt arbitration (IARB) fields. Each module that can make an interrupt service request, including the SIM, has an IARB field in its configuration register. IARB fields can be assigned values from %0000 to %1111. In order to implement an arbitration scheme, each module that can request interrupt service must be assigned a unique, non-zero IARB field value during system initialization. Arbitration priorities range from %0001 (lowest) to %1111 (highest) — if the CPU recognizes an interrupt service request from a source that has an IARB field value of %0000, a spurious interrupt exception is processed.

WARNING

Do not assign the same arbitration priority to more than one module. When two or more IARB fields have the same nonzero value, the CPU32 interprets multiple vector numbers at the same time, with unpredictable consequences.

Because the EBI manages external interrupt requests, the SIM IARB value is used for arbitration between internal and external interrupt requests. The reset value of IARB for the SIM is %1111, and the reset IARB value for all other modules is %0000.

Although arbitration is intended to deal with simultaneous requests of the same interrupt level, it always takes place, even when a single source is requesting service. This is important for two reasons: the EBI does not transfer the interrupt acknowledge read cycle to the external bus unless the SIM wins contention, and failure to contend causes the interrupt acknowledge bus cycle to be terminated early by a bus error.

When arbitration is complete, the module with both the highest asserted interrupt level and the highest arbitration priority must terminate the bus cycle. Internal modules place an interrupt vector number on the data bus and generate appropriate internal cycle termination signals. In the case of an external interrupt request, after the interrupt acknowledge cycle is transferred to the external bus, the appropriate external device must respond with a vector number, then generate data and size acknowledge ($\overline{\text{DSACK}}$) termination signals, or it must assert the autovector ($\overline{\text{AVEC}}$) request signal.

If the device does not respond in time, the SIM bus monitor, if enabled, asserts the bus error signal ($\overline{BERR}$), and a spurious interrupt exception is taken.

Chip-select logic can also be used to generate internal $\overline{AVEC}$ or $\overline{DSACK}$ signals in response to interrupt requests from external devices. Refer to **5.9.3 Using Chip-Select Signals for Interrupt Acknowledge** for more information. Chip-select address match logic functions only after the EBI transfers an interrupt acknowledge cycle to the external bus following IARB contention. If an internal module makes an interrupt request of a certain priority, and the appropriate chip-select registers are programmed to generate $\overline{AVEC}$ or $\overline{DSACK}$ signals in response to an interrupt acknowledge cycle for that priority level, chip-select logic does not respond to the interrupt acknowledge cycle, and the internal module supplies a vector number and generates internal cycle termination signals.

For periodic timer interrupts, the PIRQ[2:0] field in the periodic interrupt control register (PICR) determines PIT priority level. A PIRQ[2:0] value of %000 means that PIT interrupts are inactive. By hardware convention, when the CPU32 receives simultaneous interrupt requests of the same level from more than one SIM source (including external devices), the periodic interrupt timer is given the highest priority, followed by the $\overline{IRQ}$ pins.

5.8.4 Interrupt Processing Summary

A summary of the entire interrupt processing sequence follows. When the sequence begins, a valid interrupt service request has been detected and is pending.

A. The CPU32 finishes higher priority exception processing or reaches an instruction boundary.
B. The processor state is stacked. The S bit in the status register is set, establishing supervisor access level, and bits T1 and T0 are cleared, disabling tracing.
C. The interrupt acknowledge cycle begins:
    1. FC[2:0] are driven to %111 (CPU space) encoding.
    2. The address bus is driven as follows: ADDR[23:20] = %1111; ADDR[19:16] = %1111, which indicates that the cycle is an interrupt acknowledge CPU space cycle; ADDR[15:4] = %111111111111; ADDR[3:1] = the priority of the interrupt request being acknowledged; and ADDR0 = %1.
    3. The request level is latched from the address bus into the IP mask field in the status register.
D. Modules that have requested interrupt service decode the priority value on ADDR[3:1]. If request priority is the same as acknowledged priority, arbitration by IARB contention takes place.
E. After arbitration, the interrupt acknowledge cycle is completed in one of the following ways:
    1. When there is no contention (IARB = %0000), the spurious interrupt monitor asserts $\overline{BERR}$, and the CPU32 generates the spurious interrupt vector number.
    2. The dominant interrupt source (external or internal) supplies a vector number and $\overline{DSACK}$ signals appropriate to the access. The CPU32 acquires the

vector number.

3. The AVEC signal is asserted (the signal can be asserted by the dominant external interrupt source or the pin can be tied low), and the CPU32 generates an autovector number corresponding to interrupt priority.
4. The bus monitor asserts BERR and the CPU32 generates the spurious interrupt vector number.

F. The vector number is converted to a vector address.
G. The content of the vector address is loaded into the PC and the processor transfers control to the exception handler routine.

5.8.5 Interrupt Acknowledge Bus Cycles

Interrupt acknowledge bus cycles are CPU32 space cycles that are generated during exception processing. For further information about the types of interrupt acknowledge bus cycles determined by AVEC or DSACK, refer to **APPENDIX A ELECTRICAL CHARACTERISTICS** and the ***SIM Reference Manual*** (SIMRM/AD).

5.9 Chip-Selects

Typical microcontrollers require additional hardware to provide external chip-select and address decode signals. The MCU includes 12 programmable chip-select circuits that can provide 2 to 16 clock-cycle access to external memory and peripherals. Address block sizes of two Kbytes to one Mbyte can be selected. **Figure 5-19** is a diagram of a basic system that uses chip-selects.

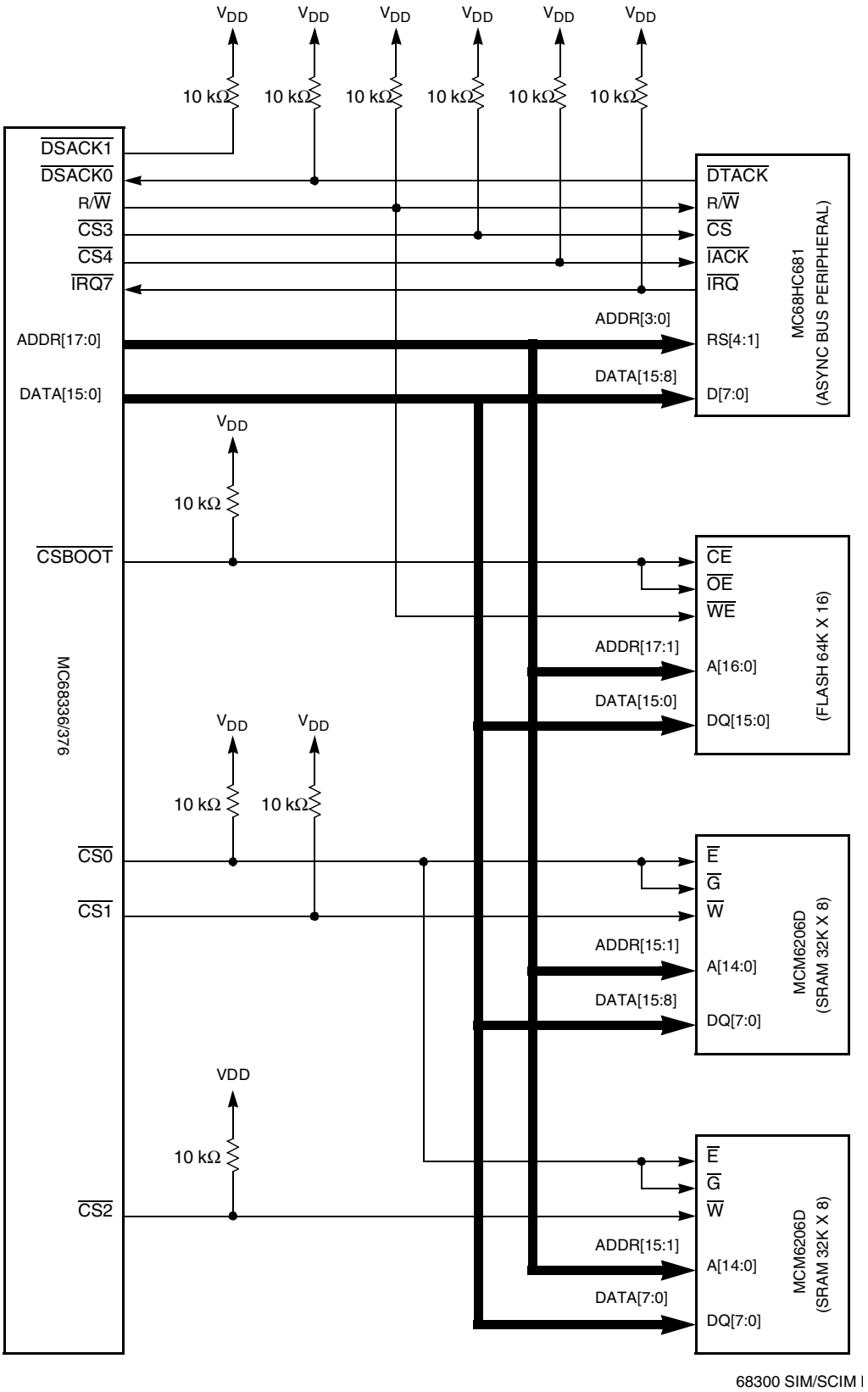


Figure 5-19 Basic MCU System

Chip-select assertion can be synchronized with bus control signals to provide output enable, read/write strobe, or interrupt acknowledge signals. Chip-select logic can also generate DSACK and AVEC signals internally. A single DSACK generator is shared by all chip-selects. Each signal can also be synchronized with the ECLK signal available on ADDR23.

When a memory access occurs, chip-select logic compares address space type, address, type of access, transfer size, and interrupt priority (in the case of interrupt acknowledge) to parameters stored in chip-select registers. If all parameters match, the appropriate chip-select signal is asserted. Select signals are active low.

If a chip-select function is given the same address as a microcontroller module or an internal memory array, an access to that address goes to the module or array, and the chip-select signal is not asserted. The external address and data buses do not reflect the internal access.

All chip-select circuits are configured for operation out of reset. However, all chip-select signals except CSBOOT are disabled, and cannot be asserted until the BYTE[1:0] field in the corresponding option register is programmed to a non-zero value to select a transfer size. The chip-select option register must not be written until a base address has been written to a proper base address register. Alternate functions for chip-select pins are enabled if appropriate data bus pins are held low at the release of RESET. Refer to **5.7.3.1 Data Bus Mode Selection** for more information. **Figure 5-20** is a functional diagram of a single chip-select circuit.

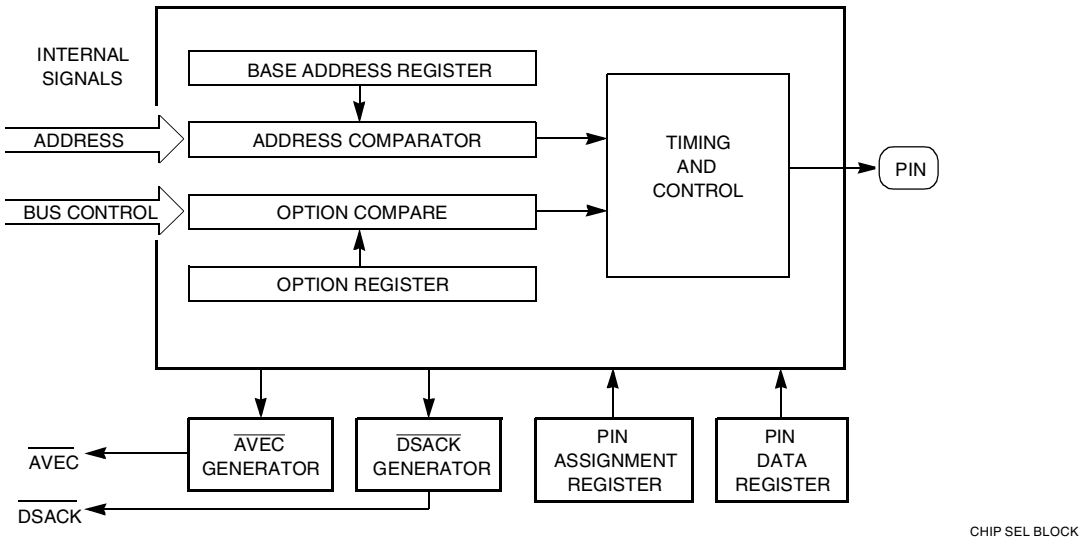


Figure 5-20 Chip-Select Circuit Block Diagram

5.9.1 Chip-Select Registers

Each chip-select pin can have one or more functions. Chip-select pin assignment registers CSPAR[0:1] determine functions of the pins. Pin assignment registers also

determine port size (8- or 16-bit) for dynamic bus allocation. A pin data register (PORTC) latches data for chip-select pins that are used for discrete output.

Blocks of addresses are assigned to each chip-select function. Block sizes of two Kbytes to one Mbyte can be selected by writing values to the appropriate base address register (CSBAR[0:10] and CSBARBT). Multiple chip-selects assigned to the same block of addresses must have the same number of wait states. The base address register for a chip-select line should be written to a value that is an exact integer multiple of both the block size and the size of the memory device being selected.

Chip-select option registers (CSORBT and CSOR[0:10]) determine timing of and conditions for assertion of chip-select signals. Eight parameters, including operating mode, access size, synchronization, and wait state insertion can be specified.

Initialization software usually resides in a peripheral memory device controlled by the chip-select circuits. A set of special chip-select functions and registers (CSORBT and CSBARBT) is provided to support bootstrap operation.

Comprehensive address maps and register diagrams are provided in **APPENDIX D REGISTER SUMMARY**.

5.9.1.1 Chip-Select Pin Assignment Registers

The pin assignment registers contain twelve 2-bit fields that determine the functions of the chip-select pins. Each pin has two or three possible functions, as shown in **Table 5-18**.

Table 5-18 Chip-Select Pin Functions

Chip-Select	Alternate Function	Discrete Output
$\overline{\text{CSBOOT}}$	$\overline{\text{CSBOOT}}$	—
$\overline{\text{CS0}}$	$\overline{\text{BR}}$	—
$\overline{\text{CS1}}$	$\overline{\text{BG}}$	—
$\overline{\text{CS2}}$	$\overline{\text{BGACK}}$	—
$\overline{\text{CS3}}$	FC0	PC0
$\overline{\text{CS4}}$	FC1	PC1
$\overline{\text{CS5}}$	FC2	PC2
$\overline{\text{CS6}}$	ADDR19	PC3
$\overline{\text{CS7}}$	ADDR20	PC4
$\overline{\text{CS8}}$	ADDR21	PC5
$\overline{\text{CS9}}$	ADDR22	PC6
$\overline{\text{CS10}}$	ADDR23	ECLK

Table 5-19 shows pin assignment field encoding. Pins that have no discrete output function must not use the %00 encoding as this will cause the alternate function to be selected. For instance, %00 for $\overline{\text{CS0}}$/$\overline{\text{BR}}$ will cause the pin to perform the $\overline{\text{BR}}$ function.

Table 5-19 Pin Assignment Field Encoding

CSxPA[1:0]	Description
00	Discrete output
01	Alternate function
10	Chip-select (8-bit port)
11	Chip-select (16-bit port)

Port size determines the way in which bus transfers to an external address are allocated. Port size of eight bits or sixteen bits can be selected when a pin is assigned as a chip-select. Port size and transfer size affect how the chip-select signal is asserted. Refer to **5.9.1.3 Chip-Select Option Registers** for more information.

Out of reset, chip-select pin function is determined by the logic level on a corresponding data bus pin. The data bus pins have weak internal pull-up drivers, but can be held low by external devices. Refer to **5.7.3.1 Data Bus Mode Selection** for more information. Either 16-bit chip-select function (%11) or alternate function (%01) can be selected during reset. All pins except the boot ROM select pin ($\overline{\text{CSBOOT}}$) are disabled out of reset. There are twelve chip-select functions and only eight associated data bus pins. There is not a one-to-one correspondence. Refer to **5.9.4 Chip-Select Reset Operation** for more detailed information.

The $\overline{\text{CSBOOT}}$ signal is enabled out of reset. The state of the DATA0 line during reset determines what port width $\overline{\text{CSBOOT}}$ uses. If DATA0 is held high (either by the weak internal pull-up driver or by an external pull-up device), 16-bit port size is selected. If DATA0 is held low, 8-bit port size is selected.

A pin programmed as a discrete output drives an external signal to the value specified in the port C register. No discrete output function is available on pins $\overline{\text{CSBOOT}}$, $\overline{\text{BR}}$, $\overline{\text{BG}}$, or $\overline{\text{BGACK}}$. ADDR23 provides the ECLK output rather than a discrete output signal.

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

5.9.1.2 Chip-Select Base Address Registers

Each chip-select has an associated base address register. A base address is the lowest address in the block of addresses enabled by a chip-select. Block size is the extent of the address block above the base address. Block size is determined by the value contained in BLKSZ[2:0]. Multiple chip-selects assigned to the same block of addresses must have the same number of wait states.

BLKSZ[2:0] determines which bits in the base address field are compared to corresponding bits on the address bus during an access. Provided other constraints determined by option register fields are also satisfied, when a match occurs, the associated chip-select signal is asserted. **Table 5-20** shows BLKSZ[2:0] encoding.

Table 5-20 Block Size Encoding

BLKSZ[2:0]	Block Size	Address Lines Compared
000	2 Kbytes	ADDR[23:11]
001	8 Kbytes	ADDR[23:13]
010	16 Kbytes	ADDR[23:14]
011	64 Kbytes	ADDR[23:16]
100	128 Kbytes	ADDR[23:17]
101	256 Kbytes	ADDR[23:18]
110	512 Kbytes	ADDR[23:19]
111	1 Mbyte	ADDR[23:20]

The chip-select address compare logic uses only the most significant bits to match an address within a block. The value of the base address must be an integer multiple of the block size.

After reset, the MCU fetches the initialization routine from the address contained in the reset vector, located beginning at address $000000 of program space. To support bootstrap operation from reset, the base address field in the boot chip-select base address register (CSBARBT) has a reset value of $000, which corresponds to a base address of $000000 and a block size of one Mbyte. A memory device containing the reset vector and initialization routine can be automatically enabled by $\overline{\text{CSBOOT}}$ after a reset. Refer to **5.9.4 Chip-Select Reset Operation** for more information.

5.9.1.3 Chip-Select Option Registers

Option register fields determine timing of and conditions for assertion of chip-select signals. To assert a chip-select signal, and to provide $\overline{\text{DSACK}}$ or autovector support, other constraints set by fields in the option register and in the base address register must also be satisfied. The following paragraphs summarize option register functions. Refer to **D.2.21 Chip-Select Option Registers** for register and bit field information.

The MODE bit determines whether chip-select assertion simulates an asynchronous bus cycle, or is synchronized to the M6800-type bus clock signal ECLK available on ADDR23. Refer to **5.3 System Clock** for more information on ECLK.

BYTE[1:0] controls bus allocation for chip-select transfers. Port size, set when a chip-select is enabled by a pin assignment register, affects signal assertion. When an 8-bit port is assigned, any BYTE field value other than %00 enables the chip-select signal. When a 16-bit port is assigned, however, BYTE field value determines when the chip-select is enabled. The BYTE fields for $\overline{\text{CS}}$[10:0] are cleared during reset. However, both bits in the boot ROM chip-select option register (CSORBT) BYTE field are set (%11) when the $\overline{\text{RESET}}$ signal is released.

R/$\overline{\text{W}}$[1:0] causes a chip-select signal to be asserted only for a read, only for a write, or for both read and write. Use this field in conjunction with the STRB bit to generate asynchronous control signals for external devices.

The STRB bit controls the timing of a chip-select assertion in asynchronous mode. Selecting address strobe causes a chip-select signal to be asserted synchronized with the address strobe. Selecting data strobe causes a chip-select signal to be asserted synchronized with the data strobe. This bit has no effect in synchronous mode.

$\overline{\text{DSACK}}$[3:0] specifies the source of $\overline{\text{DSACK}}$ in asynchronous mode. It also allows the user to optimize bus speed in a particular application by controlling the number of wait states that are inserted.

NOTE

The external $\overline{\text{DSACK}}$ pins are always active.

SPACE[1:0] determines the address space in which a chip-select is asserted. An access must have the space type represented by the SPACE[1:0] encoding in order for a chip-select signal to be asserted.

IPL[2:0] contains an interrupt priority mask that is used when chip-select logic is set to trigger on external interrupt acknowledge cycles. When SPACE[1:0] is set to %00 (CPU space), interrupt priority (ADDR[3:1]) is compared to the IPL field. If the values are the same, and other option register constraints are satisfied, a chip-select signal is asserted. This field only affects the response of chip-selects and does not affect interrupt recognition by the CPU. Encoding %000 in the IPL field causes a chip-select signal to be asserted regardless of interrupt acknowledge cycle priority, provided all other constraints are met.

The $\overline{\text{AVEC}}$ bit is used to make a chip-select respond to an interrupt acknowledge cycle. If the $\overline{\text{AVEC}}$ bit is set, an autovector will be selected for the particular external interrupt being serviced. If $\overline{\text{AVEC}}$ is zero, the interrupt acknowledge cycle will be terminated with $\overline{\text{DSACK}}$, and an external vector number must be supplied by an external device.

5.9.1.4 Port C Data Register

The port C data register latches data for PORTC pins programmed as discrete outputs. When a pin is assigned as a discrete output, the value in this register appears at the output. PC[6:0] correspond to $\overline{\text{CS}}$[9:3]. Bit 7 is not used. Writing to this bit has no effect, and it always reads zero.

5.9.2 Chip-Select Operation

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

- Function codes to SPACE fields, and to the IPL field if the SPACE field encoding is not for CPU space.
- Appropriate address bus bits to base address fields.
- Read/write status to R/$\overline{\text{W}}$ fields.
- ADDR0 and/or SIZ[1:0] bits to BYTE fields (16-bit ports only).
- Priority of the interrupt being acknowledged (ADDR[3:1]) to IPL fields (when the access is an interrupt acknowledge cycle).

When a match occurs, the chip-select signal is asserted. Assertion occurs at the same time as AS or DS assertion in asynchronous mode. Assertion is synchronized with ECLK in synchronous mode. In asynchronous mode, the value of the DSACK field determines whether DSACK is generated internally. DSACK[3:0] also determines the number of wait states inserted before internal DSACK assertion.

The speed of an external device determines whether internal wait states are needed. Normally, wait states are inserted into the bus cycle during S3 until a peripheral asserts DSACK. If a peripheral does not generate DSACK, internal DSACK generation must be selected and a predetermined number of wait states can be programmed into the chip-select option register. Refer to the ***SIM Reference Manual*** (SIMRM/AD) for further information.

5.9.3 Using Chip-Select Signals for Interrupt Acknowledge

Ordinary bus cycles use supervisor or user space access, but interrupt acknowledge bus cycles use CPU space access. Refer to **5.6.4 CPU Space Cycles** and **5.8 Interrupts** for more information. There are no differences in flow for chip-selects in each type of space, but base and option registers must be properly programmed for each type of external bus cycle.

During a CPU space cycle, bits [15:3] of the appropriate base register must be configured to match ADDR[23:11], as the address is compared to an address generated by the CPU.

Figure 5-21 shows CPU space encoding for an interrupt acknowledge cycle. FC[2:0] are set to %111, designating CPU space access. ADDR[3:1] indicate interrupt priority, and the space type field (ADDR[19:16]) is set to %1111, the interrupt acknowledge code. The rest of the address lines are set to one.

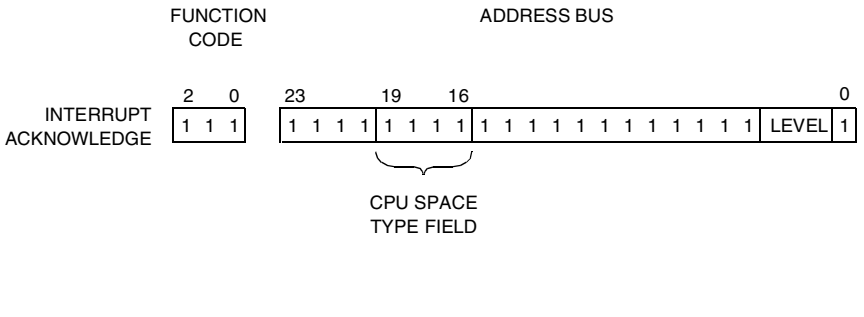


Figure 5-21 CPU Space Encoding for Interrupt Acknowledge

Because address match logic functions only after the EBI transfers an interrupt acknowledge cycle to the external address bus following IARB contention, chip-select logic generates AVEC or DSACK signals only in response to interrupt requests from external IRQ pins. If an internal module makes an interrupt request of a certain priority,

and the chip-select base address and option registers are programmed to generate AVEC or DSACK signals in response to an interrupt acknowledge cycle for that priority level, chip-select logic does not respond to the interrupt acknowledge cycle, and the internal module supplies a vector number and generates an internal DSACK signal to terminate the cycle.

Perform the following operations before using a chip-select to generate an interrupt acknowledge signal:

1. Program the base address field to all ones.
2. Program block size to no more than 64 Kbytes, so that the address comparator checks ADDR[19:16] against the corresponding bits in the base address register. (The CPU32 places the CPU space bus cycle type on ADDR[19:16].)
3. Set the R/W field to read only. An interrupt acknowledge cycle is performed as a read cycle.
4. Set the BYTE field to lower byte when using a 16-bit port, as the external vector for a 16-bit port is fetched from the lower byte. Set the BYTE field to upper byte when using an 8-bit port.

If an interrupting device does not provide a vector number, an autovector acknowledge must be generated, either by asserting the AVEC pin or by generating AVEC internally using the chip-select option register. This terminates the bus cycle.

5.9.4 Chip-Select Reset Operation

The least significant bit of each of the 2-bit chip-select pin assignment fields in CSPAR0 and CSPAR1 each have a reset value of one. The reset values of the most significant bits of each field are determined by the states of DATA[7:1] during reset. There are weak internal pull-up drivers for each of the data lines so that chip-select operation is selected by default out of reset. However, the internal pull-up drivers can be overcome by bus loading effects.

To ensure a particular configuration out of reset, use an active device to put the data lines in a known state during reset. The base address fields in chip-select base address registers CSBAR[0:10] and chip-select option registers CSOR[0:10] have the reset values shown in **Table 5-21**. The BYTE fields of CSOR[0:10] have a reset value of “disable”, so that a chip-select signal cannot be asserted until the base and option registers are initialized.

Table 5-21 Chip-Select Base and Option Register Reset Values

Fields	Reset Values
Base address	$000000
Block size	2 Kbyte
Async/sync mode	Asynchronous mode
Upper/lower byte	Disabled
Read/write	Disabled
$\overline{AS}$/$\overline{DS}$	$\overline{AS}$
$\overline{DSACK}$	No wait states
Address space	CPU space
IPL	Any level
Autovector	External interrupt vector

Following reset, the MCU fetches the initial stack pointer and program counter values from the exception vector table, beginning at $000000 in supervisor program space. The $\overline{CSBOOT}$ chip-select signal is used to select an external boot device mapped to a base address of $000000.

The MSB of the CSBTPA field in CSPAR0 has a reset value of one, so that chip-select function is selected by default out of reset. The BYTE field in chip-select option register CSORBT has a reset value of “both bytes” so that the select signal is enabled out of reset. The LSB of the $\overline{CSBOOT}$ field, determined by the logic level of DATA0 during reset, selects the boot ROM port size. When DATA0 is held low during reset, port size is eight bits. When DATA0 is held high during reset, port size is 16 bits. DATA0 has a weak internal pull-up driver, so that a 16-bit port is selected by default out of reset. However, the internal pull-up driver can be overcome by bus loading effects. To ensure a particular configuration out of reset, use an active device to put DATA0 in a known state during reset.

The base address field in the boot chip-select base address register CSBARBT has a reset value of all zeros, so that when the initial access to address $000000 is made, an address match occurs, and the $\overline{CSBOOT}$ signal is asserted. The block size field in CSBARBT has a reset value of one Mbyte. **Table 5-22** shows $\overline{CSBOOT}$ reset values.

Table 5-22 $\overline{\text{CSBOOT}}$ Base and Option Register Reset Values

Fields	Reset Values
Base address	$000000
Block size	1 Mbyte
Async/sync mode	Asynchronous mode
Upper/lower byte	Both bytes
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[1]	Any level
Autovector	Interrupt vector externally

NOTES:
1. These fields are not used unless “Address space” is set to CPU space.

5.10 Parallel Input/Output Ports

Sixteen SIM pins can be configured for general-purpose discrete input and output. Although these pins are organized into two ports, port E and port F, function assignment is by individual pin. Pin assignment registers, data direction registers, and data registers are used to implement discrete I/O.

5.10.1 Pin Assignment Registers

Bits in the port E and port F pin assignment registers (PEPAR and PFPAR) control the functions of the pins on each port. Any bit set to one defines the corresponding pin as a bus control signal. Any bit cleared to zero defines the corresponding pin as an I/O pin.

5.10.2 Data Direction Registers

Bits in the port E and port F data direction registers (DDRE and DDRF) control the direction of the pin drivers when the pins are configured as I/O. Any bit in a register set to one configures the corresponding pin as an output. Any bit in a register cleared to zero configures the corresponding pin as an input. These registers can be read or written at any time.

5.10.3 Data Registers

A write to the port E and port F data registers (PORTE[0:1] and PORTF[0:1]) is stored in an internal data latch, and if any pin in the corresponding port is configured as an output, the value stored for that bit is driven out on the pin. A read of a data register 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 port data register. Both data registers can be accessed in two locations and can be read or written at any time.

5.11 Factory Test

The test submodule supports scan-based testing of the various MCU modules. It is integrated into the SIM to support production test. Test submodule registers are intended for Motorola use only. Register names and addresses are provided in **D.2.2 System Integration Test Register** and **D.2.5 System Integration Test Register (ECLK)** to show the user that these addresses are occupied. The QUOT pin is also used for factory test.