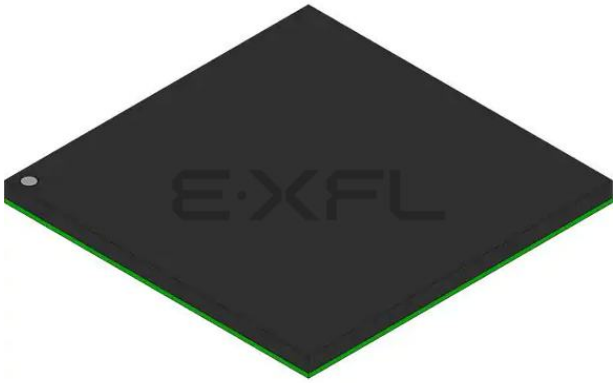


Welcome to E-XFL.COM

Understanding Embedded - Microprocessors

Embedded microprocessors are specialized computing chips designed to perform specific tasks within an embedded system. Unlike general-purpose microprocessors found in personal computers, embedded microprocessors are tailored for dedicated functions within larger systems, offering optimized performance, efficiency, and reliability. These microprocessors are integral to the operation of countless electronic devices, providing the computational power necessary for controlling processes, handling data, and managing communications.

Applications of Embedded - Microprocessors

Embedded microprocessors are utilized across a broad spectrum of applications, making them indispensable in

Details	
Product Status	Active
Core Processor	CPU32+
Number of Cores/Bus Width	1 Core, 32-Bit
Speed	33MHz
Co-Processors/DSP	Communications; CPM
RAM Controllers	DRAM
Graphics Acceleration	No
Display & Interface Controllers	-
Ethernet	10Mbps (1)
SATA	-
USB	-
Voltage - I/O	5.0V
Operating Temperature	0°C ~ 70°C (TA)
Security Features	-
Package / Case	357-BBGA
Supplier Device Package	357-PBGA (25x25)
Purchase URL	https://www.e-xfl.com/pro/item?MUrl=&PartUrl=mc68360zq33l

aligned on word or long-word boundaries, respectively. The QUICC IDMAs, when used, reduce the misalignment overhead to a minimum.

4.1 BUS TRANSFER SIGNALS

The bus transfers information between the QUICC and external memory or a peripheral device. External devices can accept or provide 8, 16, or 32 bits in parallel and must follow the handshake protocol described in this section. The maximum number of bits accepted or provided during a bus transfer is defined as the port width. The QUICC contains an address bus that specifies the address for the transfer and a data bus that transfers the data. Control signals indicate the beginning and type of the cycle as well as the address space and size of the transfer. The selected device then controls the length of the cycle with the signal(s) used to terminate the cycle. Strobe signals, one for the address bus and another for the data bus, indicate the validity of the address and provide timing information for the data.

Both asynchronous and synchronous operation is possible for any port width. In asynchronous operation, the bus and control input signals are internally synchronized to the QUICC clock, introducing a delay. This delay is the time required for the QUICC to sample an input signal, synchronize the input to the internal clocks, and determine whether it is high or low. In synchronous mode, the bus and control input signals must be timed to setup and hold times. Since no synchronization is needed, bus cycles can be completed in three clock cycles in this mode. Additionally, using the fast-termination option of the chip-select signals, two-clock operation is possible.

Furthermore, for all inputs, the QUICC latches the level of the input during a sample window around the falling edge of the clock signal. This window is illustrated in Figure 4-1, where t_{su} and t_h are the input setup and hold times, respectively. To ensure that an input signal is recognized on a specific falling edge of the clock, that input must be stable during the sample window. If an input makes a transition during the window time period, the level recognized by the QUICC is not predictable; however, the QUICC always resolves the latched level to either a logic high or low before using it. In addition to meeting input setup and hold times for deterministic operation, all input signals must obey the protocols described in this section.

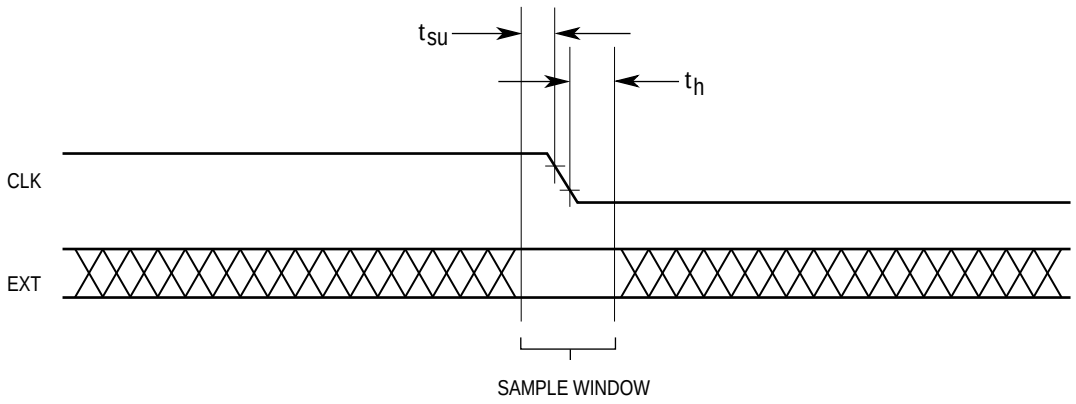


Figure 4-1. Input Sample Window

State 1—One-half clock later, in state 1 (S1), the QUICC asserts $\overline{AS}$ indicating a valid address on the address bus. The QUICC also asserts $\overline{DS}$ and $\overline{OE}$ during S1. The selected device uses R/$\overline{W}$, SIZ1 or SIZ0, A0, A1, $\overline{DS}$, and $\overline{OE}$ to place its information on the data bus. Any or all of the bytes (D31–D24, D23–D16, D15–D8, and D7–D0) are selected by SIZ1, SIZ0, A1, and A0. Concurrently, the selected device asserts $\overline{DSACKx}$.

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

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

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

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

4.3.2 Write Cycle

During a write cycle, the QUICC transfers data to memory or a peripheral device. Figure 4-19 is a flowchart of a write cycle operation for a long-word transfer. Figure 4-20 shows the functional write cycle timing diagram specified in clock periods for two write cycles (between two read cycles with no idle time) for a 32-bit port.

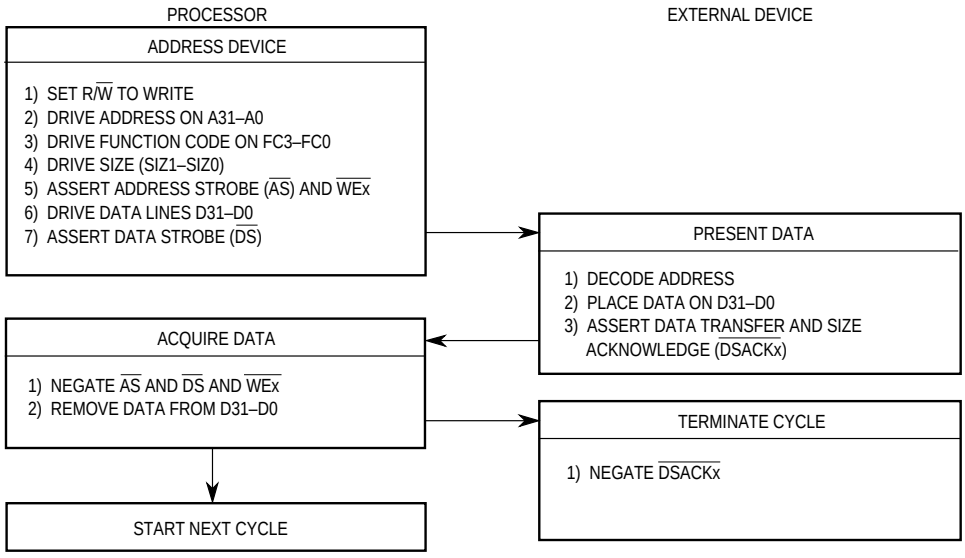


Figure 4-19. Write Cycle Flowchart

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

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

5.5 EXCEPTION PROCESSING

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

5.5.1 Exception Vectors

An exception vector is the address of a routine that handles an exception. The VBR contains the base address of a 1024-byte exception vector table, which consists of 256 exception vectors. Sixty-four vectors are defined by the processor, and 192 vectors are reserved for user definition as interrupt vectors. Except for the reset vector, which is two long words, each vector in the table is one long word. Refer to Table 5-16 for information on vector assignment.

All exception vectors, except the reset vector, are located in supervisor data space. The reset vector is located in supervisor program space. Only the initial reset vector is fixed in the processor memory map. When initialization is complete, there are no fixed assignments. Since the VBR stores the vector table base address, the table can be located anywhere in memory. It can also be dynamically relocated for each task executed by an operating system.

Each vector is assigned an 8-bit number. Vector numbers for some exceptions are obtained from an external device; others are supplied by the processor. The processor multiplies the vector number by 4 to calculate vector offset, then adds the offset to the contents of the VBR. The sum is the memory address of the vector.

5.5.1.1 TYPES OF EXCEPTIONS. An exception can be caused by internal or external events.

An internal exception can be generated by an instruction or by an error. The TRAP, TRAPcc, TRAPV, BKPT, CHK, CHK2, RTE, and DIV instructions can cause exceptions during normal execution. Illegal instructions, instruction fetches from odd addresses, word or long-word operand accesses from odd addresses, and privilege violations also cause internal exceptions.

Sources of external exception include interrupts, breakpoints, bus errors, and reset requests. Interrupts are peripheral device requests for processor action. Breakpoints are used to support development equipment. Bus error and reset are used for access control and processor restart.

The vector number for the TRAP instruction is internally generated—part of the number comes from the instruction itself. The trap vector number, PC value, and a copy of the SR are saved on the supervisor stack. The saved PC value is the address of the instruction that follows the instruction that generated the trap. For all instruction traps other than TRAP, a pointer to the instruction causing the trap is also saved in the fifth and sixth words of the exception stack frame.

5.5.2.5 SOFTWARE BREAKPOINTS. To support hardware emulation, the CPU32+ must provide a means of inserting breakpoints into target code and of announcing when a breakpoint is reached.

The MC68000 and MC68008 can detect an illegal instruction inserted at a breakpoint when the processor fetches from the illegal instruction exception vector location. Since the VBR on the CPU32+ allows relocation of exception vectors, the exception vector address is not a reliable indication of a breakpoint. CPU32+ breakpoint support is provided by extending the function of a set of illegal instructions ($4848–$484F).

When a breakpoint instruction is executed, the CPU32+ performs a read from CPU space $0, at a location corresponding to the breakpoint number. If this bus cycle is terminated by $\overline{\text{BERR}}$, the processor performs illegal instruction exception processing. If the bus cycle is terminated by $\overline{\text{DSACK}}$x, the processor uses the data returned to replace the breakpoint in the instruction pipeline and begins execution of that instruction. See Section 4 Bus Operation for a description of CPU space operations.

5.5.2.6 HARDWARE BREAKPOINTS. The CPU32+ recognizes hardware breakpoint requests. Hardware breakpoint requests do not force immediate exception processing, but are left pending. An instruction breakpoint is not made pending until the instruction corresponding to the request is executed.

A pending breakpoint can be acknowledged between instructions or at the end of exception processing. To acknowledge a breakpoint, the CPU performs a read from CPU space $0 at location $1E (see Section 4 Bus Operation).

If the bus cycle terminates normally, instruction execution continues with the next instruction as if no breakpoint request occurred. If the bus cycle is terminated by $\overline{\text{BERR}}$, the CPU begins exception processing. Data returned during this bus cycle is ignored.

Exception processing follows the regular sequence. Vector number 12 (offset $30) is internally generated. The PC of the executing instruction, the PC of the next instruction to be executed, and a copy of the SR are saved on the supervisor stack.

5.5.2.7 FORMAT ERROR. The processor checks certain data values for control operations. The validity of the stack format code and, in the case of a bus cycle fault format, the version number of the processor that generated the frame are checked during execution of the RTE instruction. This check ensures that the program does not make erroneous assumptions about information in the stack frame.

If the format of the control data is improper, the processor generates a format error exception. This exception saves a four-word format exception frame and then vectors through vec-

BDM operation is enabled when BKPT is asserted (low) at the rising edge of RESET. BDM remains enabled until the next system reset. A high BKPT on the trailing edge of RESET disables BDM. BKPT is relatched on each rising transition of RESET. BKPT is synchronized internally and must be held low for at least two clock(four clocks for RESETS)cycles prior to negation of RESETH.

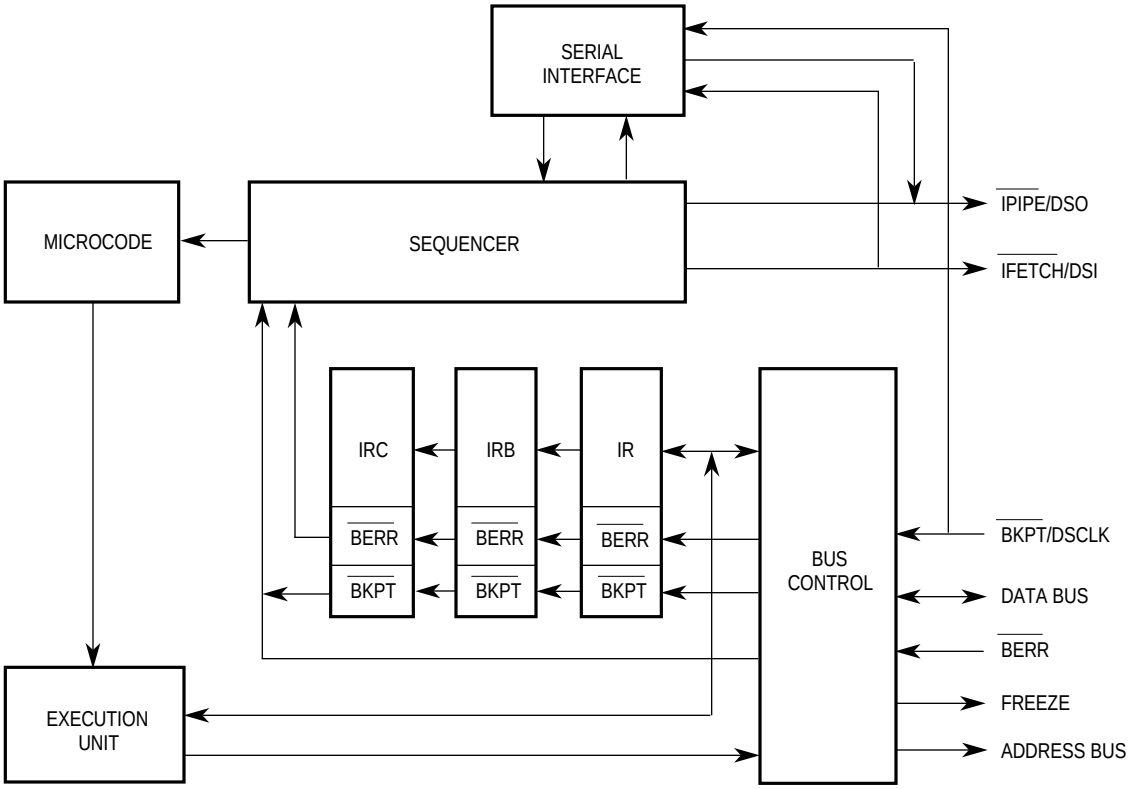


Figure 5-20. BDM Block Diagram

BDM enable logic must be designed with special care. If hold time on BKPT (after the trailing edge of RESET) extends into the first bus cycle following reset, this bus cycle could be tagged with a breakpoint. Refer to Section 4 Bus Operation for timing information.

5.6.2.2 BDM SOURCES. When BDM is enabled, any of several sources can cause the transition from normal mode to BDM. These sources include external BKPT hardware, the BGND instruction, a double bus fault, and internal peripheral breakpoints. If BDM is not enabled when an exception condition occurs, the exception is processed normally. Table 5-19 summarizes the processing of each source for both enabled and disabled cases. Note that the BKPT instruction never causes a transition into BDM.

Table 5-19. BDM Source Summary

Source	BDM Enabled	BDM Disabled
BKPT	Background	Breakpoint Exception
Double Bus Fault	Background	Halted
BGND Instruction	Background	Illegal Instruction
BKPT Instruction	Opcode Substitution/ Illegal Instruction	Opcode Substitution/ Illegal Instruction

Operand Data:

None

Result Data:

Requested data is returned as either a word or long word. Byte data is returned in the least significant byte of a word result. Word results return 16 bits of significant data; long-word results return 32 bits. Status of the read operation is returned as in the READ command: $0xxxx for success, $10001 for bus or address errors.

5.6.2.8.11 Fill Memory Block (FILL). FILL is used in conjunction with the WRITE command to fill large blocks of memory. An initial WRITE is executed to set up the starting address of the block and to supply the first operand. Subsequent operands are written with the FILL command. The initial address is incremented by the operand size (1, 2, or 4) and is saved in a temporary register. Subsequent FILL commands use this address, increment it by the current operand size, and store the updated address back in the temporary register.

NOTE

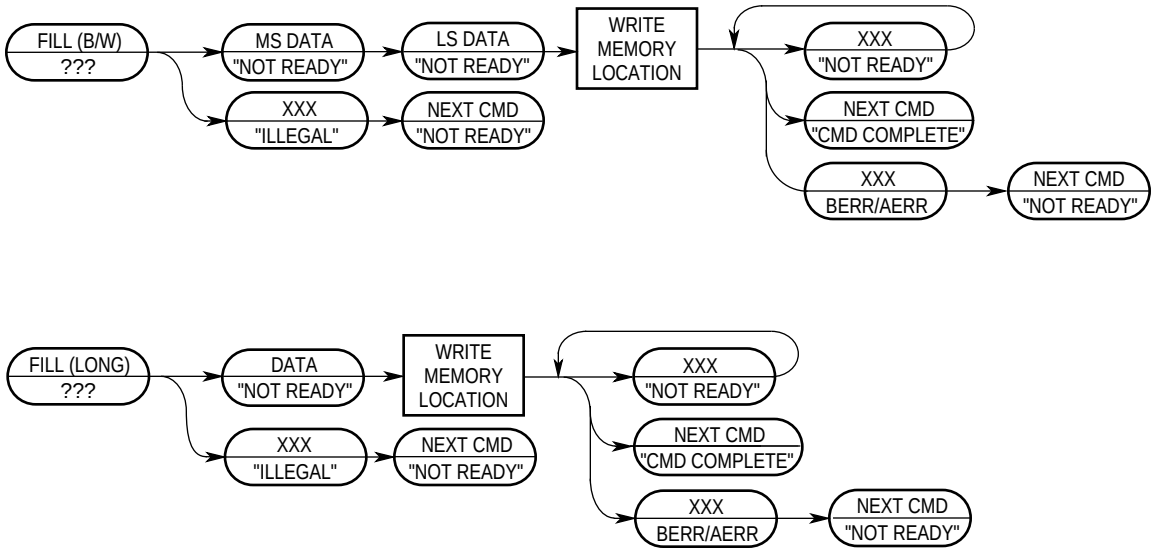
The FILL command does not check for a valid address in the temporary register—FILL is a valid command only when preceded by another FILL or by a WRITE command. Otherwise, the results are undefined. The NOP command can be used for intercommand padding without corrupting the address pointer.

The size field is examined each time a FILL command is given, allowing the operand size to be altered dynamically.

Command Format:

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

Command Sequence:



7.6.4 IDMA Operation

Every IDMA operation involves the following steps: IDMA channel initialization, data transfer, and block termination. In the initialization phase, the core (or external processor) loads the registers with control information, initializes the IDMA BDs (if auto buffer or buffer chaining is used), and then starts the channel. In the transfer phase, the IDMA accepts requests for operand transfers and provides addressing and bus control for the transfers. The termination phase occurs when the operation is complete and the IDMA interrupts the core if interrupts are enabled.

To initialize a block transfer operation, the user must initialize the IDMA registers. For the auto buffer and buffer chaining modes, the IDMA BDs must be initialized with information describing the data block, device type, request generation method, and other special control options. See 7.6.2 IDMA Registers and 7.6.4.2.3 IDMA Commands (INIT_IDMA) for further details.

7.6.4.1 SINGLE BUFFER. The single buffer mode is used to transfer only one buffer of data. When the buffer has been completely transferred (transfer count exhausted or $\overline{\text{DONE}}$x is asserted), the IDMA channel operation is terminated, STR is cleared, and a maskable interrupt is generated by the DONE bit in the CSR.

7.6.4.2 AUTO BUFFER AND BUFFER CHAINING. The auto buffer and the buffer chaining modes are supported with the RISC controller by setting the RCI bit in the CMR. The host processor should initialize the IDMA BD ring (see Figure 7-9) with the appropriate buffer handling mode, source address, destination address, and block length. The user then sets the STR bit in the CMR. All transfer modes described in 7.6.4.4.4 External Cycle Steal are still valid. The function codes for the source and destination addresses are programmed as described in 7.5.2.5 Timer Capture Registers (TCR1, TCR2, TCR3, TCR4).

pin. Thus, DACKx is the acknowledgment of the original cycle steal request given on the DREQx pin.

It is possible to cause the IDMA to perform back-to-back cycle steal requests. To achieve this, DREQx should be asserted to generate the first request, negated, and reasserted during the access to the device. If the IDMA detects that DREQx is reasserted prior to the S3 falling edge of the bus cycle to the device (i.e., bus cycle when DACKx is asserted), then another back-to-back cycle steal request will be performed. Otherwise, the bus is relinquished. If DREQx was not reasserted soon enough, a new request will be made to the IDMA, but the bus will be relinquished and re-requested by the IDMA.

NOTE

To generate back-to-back cycle steal requests, DREQx should be reasserted after DACKx is asserted, but before the S3 falling edge. Instead of saying before the S3 falling edge, one could also say before or with the assertion of DSACKx because the DSACKx pins are also sampled at falling S3 to determine the end of the bus cycle.

The previous paragraphs discuss the general rules; however, important special cases are discussed in the following points:

1. The sample point at the S3 falling edge means the last S3 before the S4 edge that completes the cycle. Thus, if wait states are inserted in the bus cycle, the sample point is later in the cycle.
2. The sample point at S3 assumes that the required setup time is met, as defined in Section 10 Electrical Characteristics.
3. If SRM is cleared in the CMR (default condition), then DREQx is synchronized internally before it is used; therefore, DREQx must be reasserted one clock earlier than the S3 falling edge to be recognized on that cycle and generate a back-to-back request.

7.6.4.5 IDMA BUS ARBITRATION. Once the IDMA receives a request for a transfer, it begins arbitrating for the IMB. (The four request types are internal maximum rate, internal limited rate, external burst, and external cycle steal.)

On the QUICC, the IDMAs, SDMAs, and DRAM refresh controller, called internal masters, have the capability to become bus master. To determine the relative priority of these masters, each is given an arbitration ID. The user programs the arbitration ID (a value between 0 and 7) of the IDMAs into the ICCR. The arbitration IDs of the two IDMAs must be different by a value of 2 (e.g., IDMA1 ID = 2 and IDMA2 ID = 0). These values are used to determine the relative priority of the IDMA channel and the other internal bus masters.

NOTE

Typically, the IDMA IDs are configured by the user to be the lowest of the internal bus masters.

7.10.19.4.2 PSMR Programming. The PSMR programming sequence is as follows:

1. The NOF bits should be set to 0001 (binary) giving two flags before frames (one opening flag, plus one additional flag).
2. The CRC should be set to 16-bit CRC-CCITT.
3. The DRT bit should be set.
4. All other bits should be set to zero or to their default condition.

7.10.19.4.3 TODR Programming. To expedite a transmit frame, the transmit on demand register (TODR) may be used.

7.10.19.4.4 AppleTalk Controller Example. Except for the previously discussed register programming, the HDLC Example #1 may be followed.

7.10.20 BISYNC Controller

The byte-oriented binary synchronous communication (BISYNC) protocol was originated by IBM for use in networking products. The three classes of BISYNC frames are transparent, non-transparent with header, and non-transparent without header (see Figure 7-63). The transparent mode in BISYNC allows full binary data to be transmitted with any possible character pattern. Each class of frame starts with a standard two-octet synchronization pattern and ends with a block check code (BCC). The end of text character (ETX) is used to separate the text and BCC fields.

NOTE

The transparent frame type in BISYNC is not related to the totally transparent protocol supported by the QUICC. See 7.10.21 Transparent Controller for details.

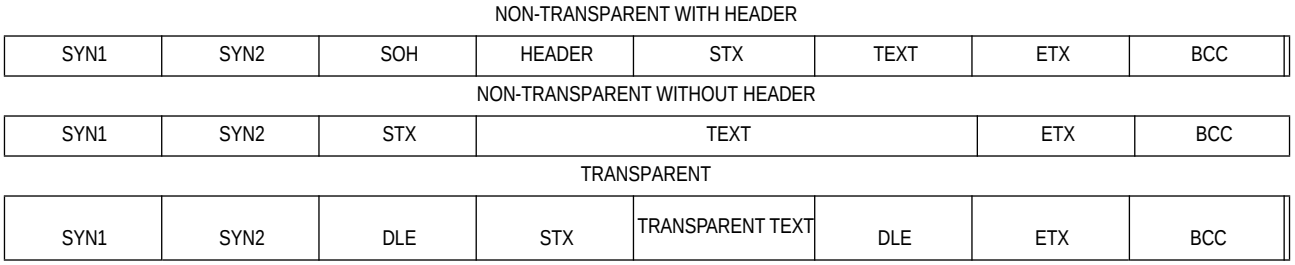


Figure 7-63. Typical BISYNC Frames

The bulk of the frame is divided into fields whose meaning depends on the frame type. The BCC is a 16-bit CRC (CRC16) format if 8-bit characters are used; it is a longitudinal check (a sum check) in combination with vertical redundancy check (parity) if 7-bit characters are used. In transparent operation, to allow the BISYNC control characters to be present in the frame as valid text data, a special character (DLE) is defined, which informs the receiver that the character following the DLE is a text character, not a control character. If a DLE is transmitted as valid data, it must be preceded by a DLE character. This technique is sometimes called byte-stuffing.

7.11 SERIAL MANAGEMENT CONTROLLERS (SMCS)

The SMC key features are as follows:

- Each SMC can implement the UART protocol on its own pins.
- Each SMC can implement a totally transparent protocol on a multiplexed line or on a nonmultiplexed line. This mode can also be used for a fast connection between QUICCs.
- Each SMC channel fully supports the C/I and Monitor channels of the GCI (IOM-2) in ISDN applications.
- Two SMCs fully support the two sets of C/I and Monitor channels in the SCIT channel 0 and channel. 1
- Full-Duplex operation.
- Local Loopback and Echo Capability for testing.

7.11.1 SMC Overview

The SMCs are two full-duplex ports that may be independently configured to support any one of three protocols: UART, transparent, or GCI.

The SMCs can support simple UART operation for such purposes as providing a debug/monitor port in an application, allowing the four SCCs to be free for another purpose. The UART functionality of the SMCs is reduced as compared to the SCCs. The SMC clock can be derived from one of the four internal baud rate generators or from an external clock pin. The clock provided to the SMC should be a 16x clock.

The SMCs can also support totally transparent operation. In this mode, the SMC may be connected to a TDM channel (such as a T1 line) or directly to its own set of pins. The receive and transmit clocks can be derived from the TDM channel, the internal baud rate generators, or from an external clock. In either case, the clock provided to the SMCs should be a 1x clock. The transparent protocol also allows the use of an external synchronization pin for the transmitter and receiver. The transparent functionality of the SMCs is reduced as compared to the SCCs.

Finally, each SMC can support the C/I and monitor channels of the GCI bus (IOM-2). In this case, the SMC is connected to a TDM channel in the SI. See 7.8 Serial Interface with Time Slot Assigner for the details of configuring the GCI interfaces.

The SMCs support loopback and echo modes for testing.

NOTE

In the MC68302, the SMCs also provide support for the A and M bits of the IDL definition. Since the IDL definition has been modified to eliminate the A and M bits, the QUICC does not provide special SMC support for IDL; however, the A and M bits may still be routed to the SMC using the TSA, if desired. The SMC would be configured into transparent mode for this operation.

7.11.10.9 SMC TRANSPARENT ERROR-HANDLING PROCEDURE. The SMC reports message reception and transmission error conditions using the channel BDs and the SMC event register.

7.11.10.9.1 Transmission Error (Underrun). When this error occurs, the channel terminates buffer transmission, closes the buffer, sets the UN bit in the BD, and generates the TXE interrupt if it is enabled. The channel resumes transmission after the reception of the RESTART TRANSMIT command. Underrun cannot occur between frames.

7.11.10.9.2 Reception Error (Overrun). The SMC maintains an internal FIFO for receiving data (shift register plus data register). The CP begins programming the SDMA channel (if the data buffer is in external memory) when the first character is received into the FIFO. If a FIFO overrun occurs, the SMC writes the received data character to the internal FIFO over the previously received character. The previous character and its status bits are lost. Following this, the channel closes the buffer, sets the OV bit in the BD, and generates the RX interrupt if it is enabled. Reception then continues normally.

7.11.10.10 SMC TRANSPARENT MODE REGISTER (SMCMR). The operating mode of an SMC is defined by the SMCMR. The SMCMR is a 16-bit, memory-mapped, read-write register. The register is cleared at reset. The function of bits 7–0 is common to each SMC protocol. The function of bits 15–8 varies according to the protocol selected by the SM bits.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
—	CLEN				—		REVD	—		SM		DM		TEN	REN

Bits 15, 10, 9, 7, 6—Reserved

CLEN—Character Length

CLEN is programmed with a value from 3 to 15 to obtain 4 to 16 bits per character. If the character length is less than 8 bits, the MSBs of the byte in buffer memory are not used on transmit and are written with zeros on receive. If the character length is more than 8 bits, but less than 16 bits, the MSBs of the word in buffer memory will not be used on transmit and will be written with zeros on receive.

NOTES

The values 0 to 2 should not be written to CLEN, or erratic behavior may result.

Larger character lengths increase the potential performance of the SMC channel and lower the performance impact on other channels. For instance, the use of 16-bit characters, rather than 8-bit characters, is encouraged if 16-bit characters are acceptable in the end application.

REVD—Reverse Data

0 = Normal mode
1 = Reverse the character bit order; the MSB is transmitted first.

Thus, to connect to QUICCs using this interface, connect the STBO pin of each QUICC to the STBI pin of the other and connect the desired data pins (either PB8–PB15 or PB0–PB15 are connected between QUICCs).

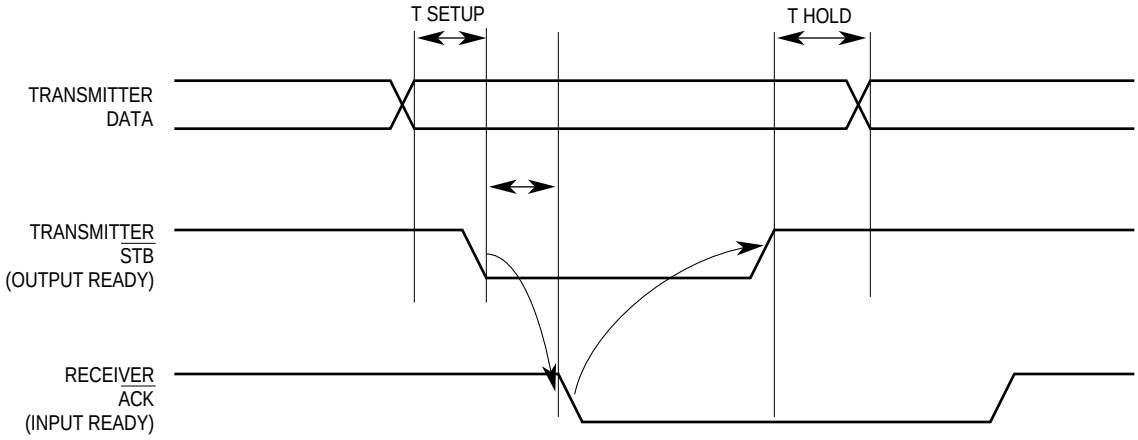


Figure 7-85. Interlock Handshake Mode

7.13.5 Pulsed Data Transfers

In the pulsed handshake mode, the PIP may be configured as a transmitter or a receiver. This configuration allows a Centronics-compatible interface to be implemented.

The pulsed handshake mode may be controlled by the RISC or the CPU32+ core. Operation using the RISC requires BDs and parameter RAM initialization very similar to the other serial channels. Data is then stored in the buffers using one of the SDMA channels (one of the available channels from SMC2). Operation by the CPU32+ core is performed by software-controlled reads and writes from/to the PIP data register upon interrupt request.

NOTE

At the time of writing, RISC operation of the PIP has not been fully defined. The user should use the CPU32+ core operation mode until as RISC microcode becomes available or the full PIP microcode is available in the RISC internal ROM. Please contact the local Motorola sales representative to obtain the current status of the PIP RISC microcode. In the following description, the RISC reads and writes of the data register are replaced by CPU32+ core reads and writes.

When configured as a transmitter, the STBO pin (PB16) is used as a strobe output ($\overline{STB}$) handshake control signal, and the STBI pin (PB17) is used as an acknowledge ($\overline{ACK}$) input. When configured as a receiver, the PIP generates the $\overline{ACK}$ signal on the STBO pin and inputs the $\overline{STB}$ signal on the STBI pin.

Bits PB16 and PB17 in the port B data direction register (PBDIR) and the port B data register (PBDAT) corresponding to STBO and STBI are not valid and are ignored by the PIP when the pulsed handshake mode is selected.

When configured as a transmitter, the PIP generates the $\overline{STB}$ signal when data is ready in the PIP's output latch and the previous transfer has been acknowledged (see Figure 7-86). The setup time and the strobe pulse width are user programmable. When configured as a receiver, the PIP uses the $\overline{STB}$ signal to latch the input data and acknowledges the transfer with the $\overline{ACK}$ signal. The timing of the $\overline{ACK}$ signal is user programmable.

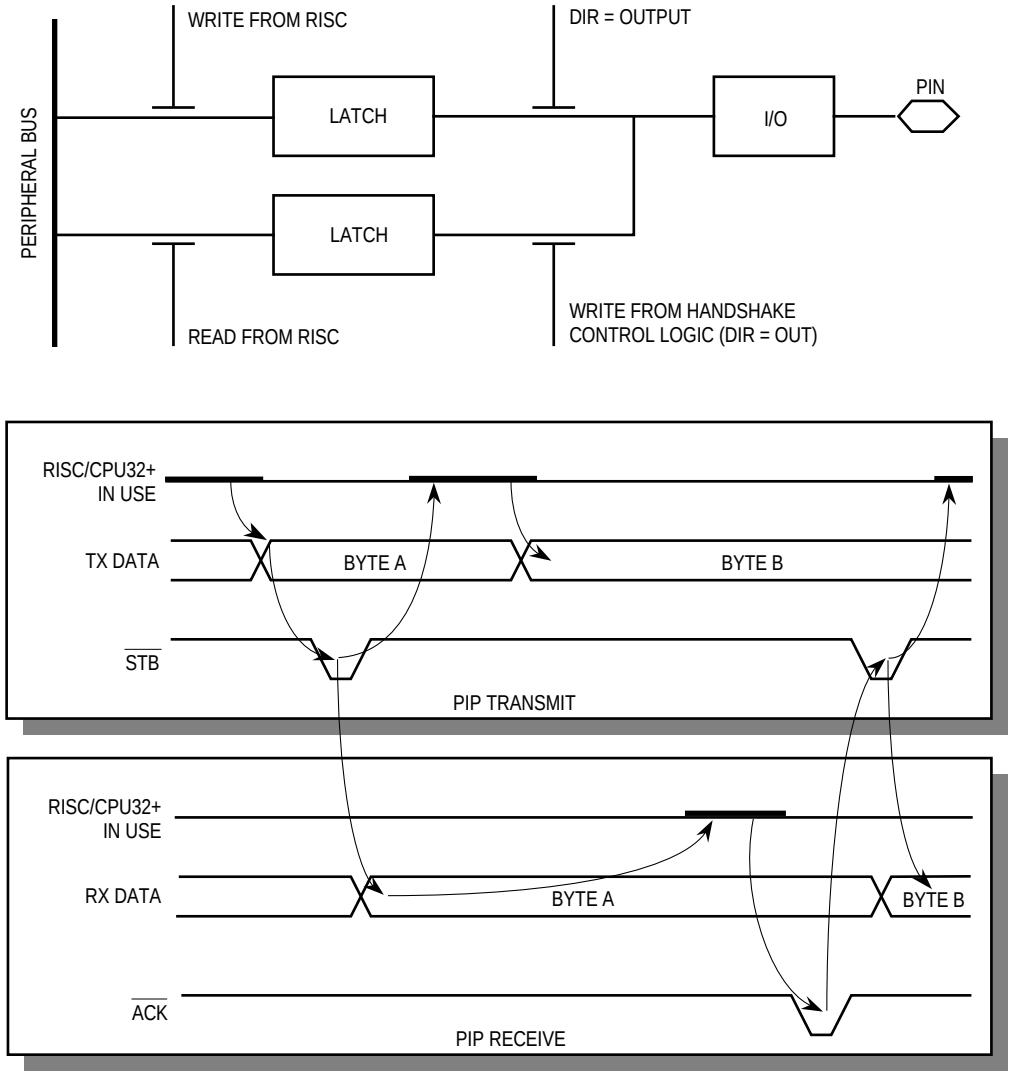


Figure 7-86. Pulsed Handshake Full Cycle

7.13.5.1 BUSY SIGNAL. In the pulsed handshake mode, the PIP receiver can generate an additional BUSY handshake signal, which is useful to implement the Centronics reception interface (see Figure 7-87). The BUSY signal is an output indication of a transfer in service. It is asserted by the Centronics receiver as soon as the data is latched into the PIP data register. The timing of BUSY negation in relation to the $\overline{ACK}$ signal is user programmable. Two bits in the PIP configuration register enable the assertion and negation of the BUSY signal via the host processor software.

4. Set the PCINT bit to determine which edges cause interrupts.
5. Write the corresponding CIMR bit with a one to allow interrupts to be generated to the CPU32+ core.
6. Read the pin value using the PCDAT.

NOTE

These steps correspond to the “software operation” mode of the SCM DIAG bits on the MC68302.

The port C lines associated with the $\overline{CDx}$ and $\overline{CTSx}$ pins have a mode of operation where the pin may be internally connected to the SCC but may also generate interrupts. Port C still detects changes on the $\overline{CTS}$ and $\overline{CD}$ pins and asserts the corresponding interrupt request, but the SCC simultaneously uses the $\overline{CTS}$ and/or $\overline{CD}$ pin to automatically control operation. This allows the user to fully implement protocols V.24, X.21, and X.21 bis (with the assistance of other general-purpose I/O lines).

To configure a port C pin as a $\overline{CTS}$ or $\overline{CD}$ pin that is connected to the SCC and also generates interrupts, use the following steps:

1. Write the corresponding PCPAR bit with a zero.
2. Write the corresponding PCDIR bit with a zero.
3. Write the corresponding PCSO bit with a one.
4. Set the PCINT bit to determine which edges cause interrupts.
5. Write the corresponding CIMR bit with a one to allow interrupts to be generated to the CPU32+ core.
6. The pin value may be read at any time using PCDAT.

NOTE

After connecting the $\overline{CTS}$ or $\overline{CD}$ pins to the SCC, the user must also choose the “normal operation” mode in DIAG bits of the general SCC mode register (GSMR) to enable and disable SCC transmission and reception with these pins.

7.14.10 Port C Registers

The user interfaces with port C via five registers. The port C interrupt control register (PCINT) indicates how changes on the pin cause interrupts when interrupts are generated with that pin. The port C special options register (PCSO) indicates whether certain port C pins have the ability to be connected to on-chip peripherals while simultaneously being able to generate an interrupt. The other three port C registers also exist on the other ports: PCDAT, PCDIR, and PCPAR. Since port C does not have open-drain capability, it does not contain an open-drain register.

former, see the DRAM multiplexing scheme in 9.4 Using the QUICC MC68040 Companion Mode.

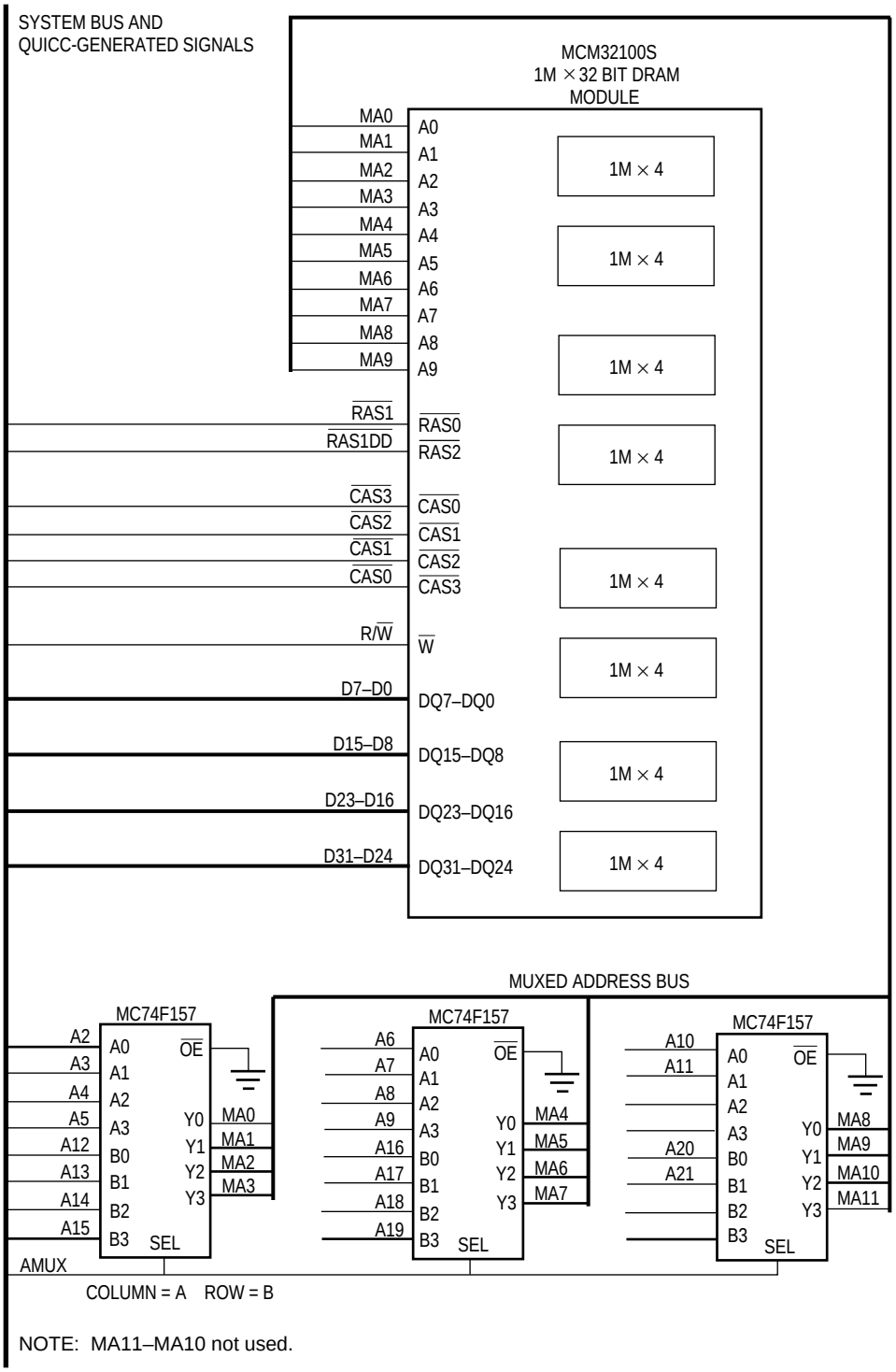
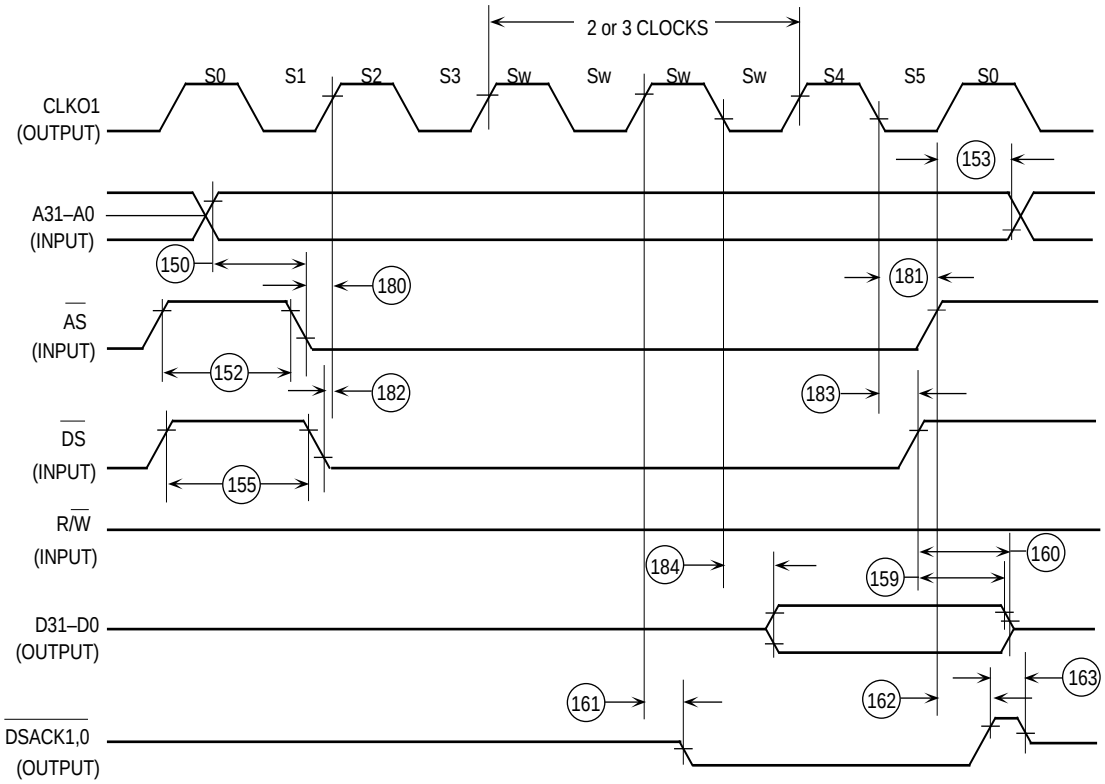


Figure 9-32. 4-Mbyte DRAM Bank—32 Bits Wide



NOTE: Two wait states are inserted when reading the SIM, dual-port RAM, and CPM. Three wait states are inserted when reading the SI RAM. Additional wait states may be inserted when the SHEN1–SHEN0 =10 and one of the internal masters is accessing an internal peripheral.

Figure 10-30. External MC68030/MC68360 Internal Synchronous Read Cycle Timing Diagram

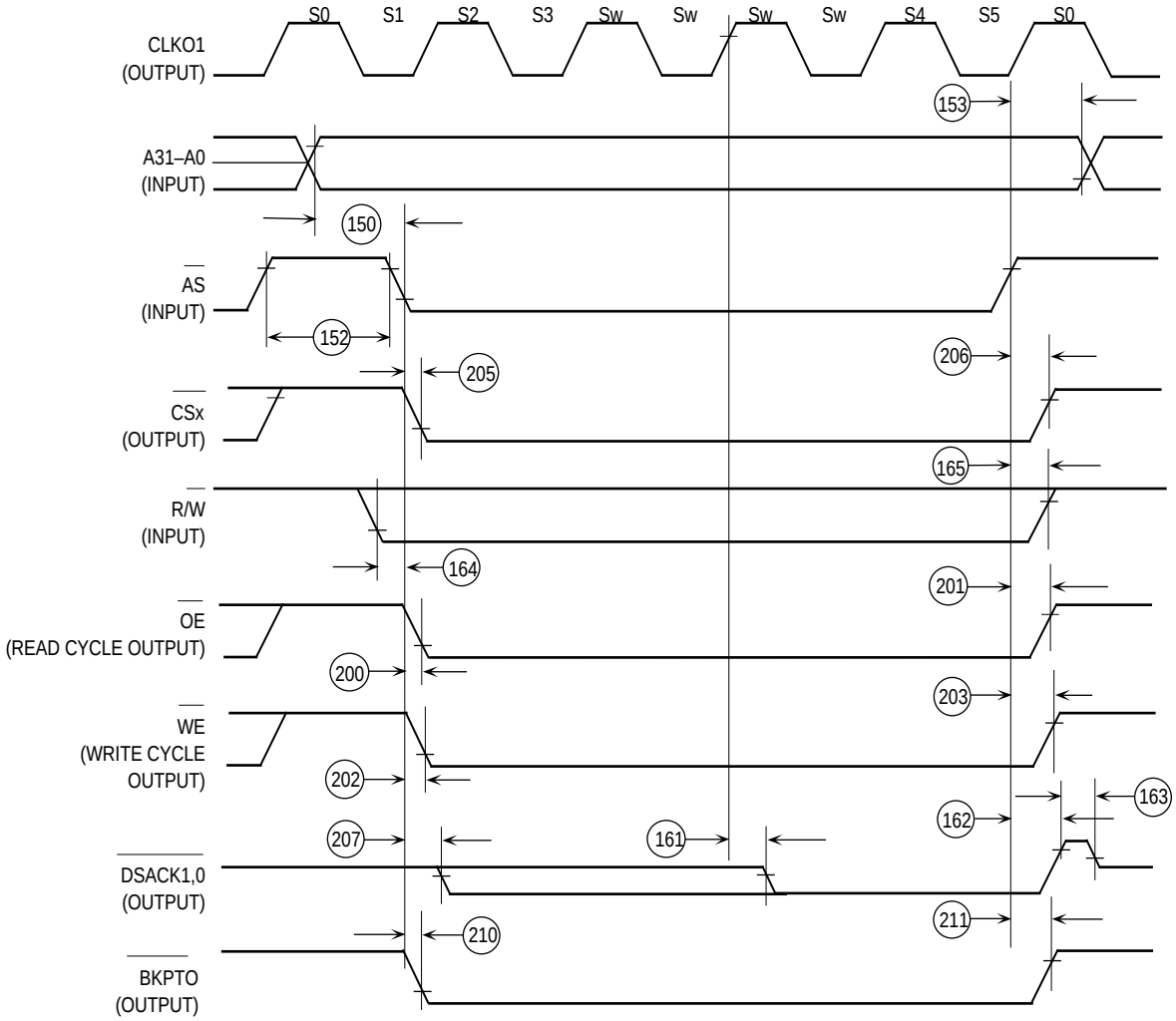


Figure 10-33. External MC68030/MC68360 SRAM Asynchronous Cycle Timing Diagram (BSTN = 0/1; SYNC = 0)

MC68160 EEST device. The LocalTalk connection is made through an RS-422 transceiver. Applications not needing LocalTalk can use the RS-422 to implement other proprietary LANs. The BDM output connector allows the QUADS board to control another device using background debug mode. All these connections are made through the slave QUICC to prevent master QUICC resources from being impacted.

If the user application includes a need for the RS-232 port, the system RS-232 port may be used for the application if the host computer is connected to the QUADS board via the parallel port.

More detail of the QUADS system is shown in Figure B-3. Buffers are provided in this system so that the system functions do not affect the fanout of the expansion connector. In a real application, such buffers and the local bus arbiter are not necessarily required. The slave QUICC provides the system integration functions (such as chip selects) that would normally be provided by the QUICC's own SIM. Thus, the expansion connectors provide all the master QUICC resources without using these resources and provide a non-degraded QUICC electrical interface.

To assist with target board development, the QUADS board may be connected to the target board through the edge connectors or through a PGA connector using a flexible PC board supplied by a third party. (Adapters from PGA to PQFP are also available from third parties.) Prototype boards may therefore make use of signals available from the QUADS development board. Target board execution may be carried out with the QUICC executing from the QUADS board memory, from target board memory, or a combination of both.

The QUADS board also contains connections for use by a logic analyzer, allowing all QUICC pins to be examined. Connectors between the QUADS board and common logic analyzers are available from third parties.

To connect the QUADS board to a host computer, a parallel interface cable and host interface (ADI) board are supplied. Hosts include the IBM-PC and SUN-4 systems. Available software executing on the host allows uploading and downloading of files and data between the host and the QUADS board and allows control of the user interface module through a window-based interface program resident on the IBM-PC or SUN-4. Source-level debugging support through this interface and the serial interface is provided by third-party vendors.

NOTE

The ADI boards mentioned here are the same ADI boards used with the MC68302 ADS system. If the user already purchased an ADI board for a MC68302 development project, that same board may be used with the QUADS board as long as the ADI board supports the IBM-PC, or SUN-4. The user only has to obtain new host software (available on the BBS) to interface to the QUADS board.

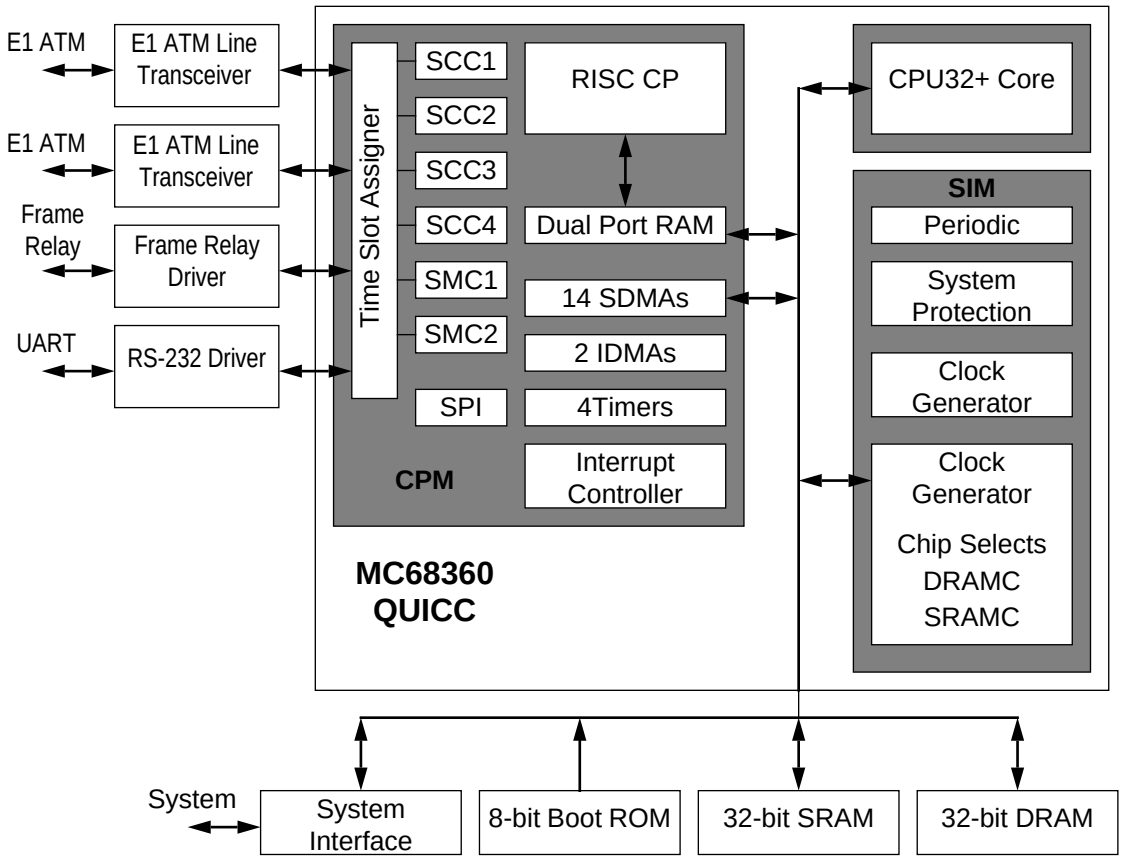


Figure C-2. ATM/Frame Relay Interwork System

- Empty cells transmitted when there are no pending data transfers.
- Empty cells and cells with non-matching headers are automatically discarded on receive.
- Scrambling option is provided utilizing the self-synchronizing $X^{43} + 1$ scrambling polynomial.
- Incoming cells with incorrect HECs are received and marked.
- Bandwidth reservation mechanism in the transmitter to allow mixing of data and isochronous services.
- CAM support on reception for handling many connections.

  —Consumes 1280 bytes of the QUICC's internal memory.

C.3.2 Performance

At 25 Mhz, an aggregate ATOM1 bandwidth of 10 Mbps divided among the 4 SCCs consumes 100% of the processing power of the RISC communications engine. If only a percentage of the total available ATOM1 bandwidth is used, the remaining RISC processing power can be used to run other protocols on other channels. Table C-3 shows the possible QUICC configuration.

D.2.8 Development Support

No new development tools will be needed for the QUICC32.

Only the replacement of actual silicon is needed. All Motorola and third party support tools will work as before. Motorola will provide simple device drivers for developers to have an easy start on the software.

D.2.9 Ordering Information

The following table identifies the packages and operating frequencies available for the MC68MH360.

Table D-1. MC68MH360 Package/Frequency Availability

Package Type	V_{CC}	Frequency (MHz)	Temperature	Order Number
Quad Flat Pack (FE Suffix) Quad Flat Pack (CFE Suffix)	5 V	0–25 0–25	0°C to 70°C –40°C to +85°C	MC68MH360FE25 MC68MH360CFE25
Pin Grid Array (RC Suffix) Pin Grid Array (CRC Suffix)	5 V	0–25 0–25	0°C to 70°C –40°C to +85°C	MC68MH360RC25 MC68MH360CRC25
Quad Flat Pack (FE Suffix	5 V	0–33	0°C to 70°C	MC68MH360FE33
Pin Grid Array (RC Suffix)	5 V	0–33	0°C to 70°C	MC68MH360RC33
Quad Flat Pack (FE Suffix	3.3 V	0–25	0°C to 70°C	MC68MH360FE25V
Pin Grid Array (RC Suffix)	3.3 V	0–25	0°C to 70°C	MC68MH360RC25V

The documents listed in the following table contain detailed information on the MC68360. These documents may be obtained from the Literature Distribution Centers at the addresses listed at the bottom of this page.

Table D-2. Documentation

Document Title	Order Number	Contents
MC68MH360 Specification Addendum	Contact local field sales office	Preliminary design information
M68000 Family Programmer's Reference Manual	M68000PM/AD	M68000 Family Instruction Set
The 68K Source	BR729/D	Independent vendor listing supporting software and development tools