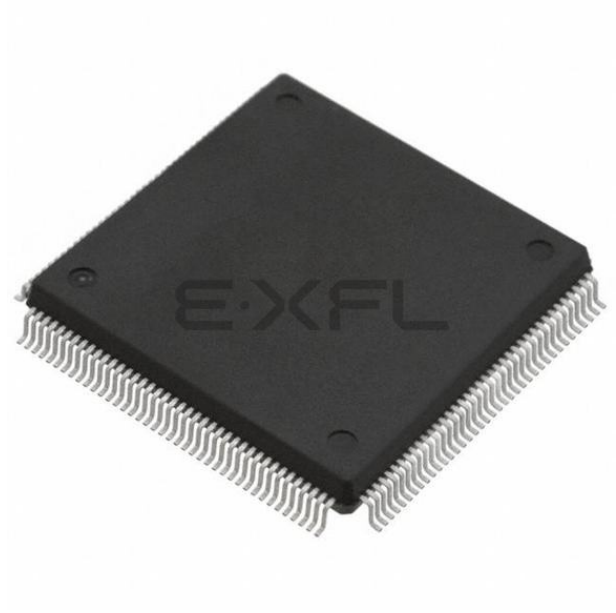

Welcome to E-XFL.COM

What is "Embedded - Microcontrollers"?

"Embedded - Microcontrollers" refer to small, integrated circuits designed to perform specific tasks within larger systems. These microcontrollers are essentially compact computers on a single chip, containing a processor core, memory, and programmable input/output peripherals. They are called "embedded" because they are embedded within electronic devices to control various functions, rather than serving as standalone computers. Microcontrollers are crucial in modern electronics, providing the intelligence and control needed for a wide range of applications.

Applications of "Embedded - Microcontrollers"

Details	
Product Status	Obsolete
Core Processor	CPU32
Core Size	32-Bit Single-Core
Speed	20MHz
Connectivity	CANbus, EBI/EMI, SCI, SPI
Peripherals	POR, PWM, WDT
Number of I/O	18
Program Memory Size	-
Program Memory Type	ROMless
EEPROM Size	-
RAM Size	7.5K x 8
Voltage - Supply (Vcc/Vdd)	4.75V ~ 5.25V
Data Converters	A/D 16x10b
Oscillator Type	External
Operating Temperature	-40°C ~ 85°C (TA)
Mounting Type	Surface Mount
Package / Case	160-BQFP
Supplier Device Package	160-QFP (28x28)
Purchase URL	https://www.e-xfl.com/product-detail/nxp-semiconductors/mc68376bacft20

TABLE OF CONTENTS
(Continued)

Paragraph **Title** **Page**

FCSM12CNT	—	CTM4 FCSM12 Counter Register
FCSM12SIC	—	CTM4 FCSM12 Status/Interrupt/Control Register
HSQR[0:1]	—	TPU Host Sequence Registers [0:1]
HSRR[0:1]	—	TPU Host Service Request Registers [0:1]
LJSRR[0:27]	—	QADC Left-Justified Signed Result Registers [0:27]
LJURR[0:27]	—	QADC Left-Justified Unsigned Result Registers [0:27]
LR	—	Link Register
MCSM[2]/[11]CNT	—	CTM4 MCSM Counter Registers [2]/[11]
MCSM[2]/[11]ML	—	CTM4 MCSM Modulus Latch Registers [2]/[11]
MCSM[2]/[11]SIC	—	CTM4 MCSM Status/Interrupt/Control Registers [2]/[11]
MRMCR	—	Masked ROM Module Configuration Register
PEPAR	—	SIM Port E Pin Assignment Register
PFPAR	—	SIM Port F Pin Assignment Register
PICR	—	SIM Periodic Interrupt Control Register
PITR	—	SIM Periodic Interrupt Timer Register
PORTC	—	SIM Port C Data Register
PORTE	—	SIM Port E Data Register
PORTF	—	SIM Port F Data Register
PORTQA	—	QADC Port A Data Register
PORTQB	—	QADC Port B Data Register
PORTQS	—	QSM Port QS Data Register
PQSPAR	—	QSM Port QS Pin Assignment Register
PRESDIV	—	TouCAN Prescaler Divide Register
PWM[5:8]C	—	CTM4 PWMSM Counter Registers [5:8]
PWM[5:8]A	—	CTM4 PWMSM Period Registers [5:8]
PWM[5:8]B	—	CTM4 PWMSM Pulse Width Registers [5:8]
PWM[5:8]SIC	—	CTM4 PWMSM Status/Interrupt/Control Registers [5:8]
QACR[0:1]	—	QADC Control Registers [0:2]
QADCINT	—	QADC Interrupt Register
QADCMCR	—	QADC Module Configuration Register
QADCTEST	—	QADC Test Register
QASR	—	QADC Status Register
QILR	—	QSM Interrupt Level Register
QIVR	—	QSM Interrupt Vector Register
QSMCR	—	QSM Module Configuration Register
QTEST	—	QSM Test Register
RAMBAH	—	RAM Base Address High Register

SECTION 3 OVERVIEW

This section contains information about the entire MC68336/376 modular microcontroller. It lists the features of each module, shows device functional divisions and pin assignments, summarizes signal and pin functions, discusses the intermodule bus, and provides system memory maps. Timing and electrical specifications for the entire microcontroller and for individual modules are provided in **APPENDIX A ELECTRICAL CHARACTERISTICS**. Comprehensive module register descriptions and memory maps are provided in **APPENDIX D REGISTER SUMMARY**.

3.1 MCU Features

The following paragraphs highlight capabilities of each of the microcontroller modules. Each module is discussed separately in a subsequent section of this user's manual.

3.1.1 Central Processing Unit (CPU32)

- 32-bit architecture
- Virtual memory implementation
- Table look-up and interpolate instruction
- Improved exception handling for controller applications
- High level language support
- Background debug mode
- Fully static operation

3.1.2 System Integration Module (SIM)

- External bus support
- Programmable chip select outputs
- System protection logic
- Watchdog timer, clock monitor and bus monitor
- Two 8-bit dual function input/output ports
- One 7-bit dual function output port
- Phase-locked loop (PLL) clock system

3.1.3 Standby RAM Module (SRAM)

- 4-Kbytes of static RAM
- No standby supply

3.1.4 Masked ROM Module (MRM)

- 8-Kbyte array, accessible as bytes or words
- User selectable default base address
- User selectable bootstrap ROM function
- User selectable ROM verification code

4.4 Virtual Memory

The full addressing range of the CPU32 on the MC68336/376 is 16 Mbytes in each of eight address spaces. Even though most systems implement a smaller physical memory, the system can be made to appear to have a full 16 Mbytes of memory available to each user program by using virtual memory techniques.

A system that supports virtual memory has a limited amount of high-speed physical memory that can be accessed directly by the processor and maintains an image of a much larger virtual memory on a secondary storage device. When the processor attempts to access a location in the virtual memory map that is not resident in physical memory, a page fault occurs. The access to that location is temporarily suspended while the necessary data is fetched from secondary storage and placed in physical memory. The suspended access is then restarted or continued.

The CPU32 uses instruction restart, which requires that only a small portion of the internal machine state be saved. After correcting the fault, the machine state is restored, and the instruction is fetched and started again. This process is completely transparent to the application program.

4.5 Addressing Modes

Addressing in the CPU32 is register-oriented. Most instructions allow the results of the specified operation to be placed either in a register or directly in memory. There is no need for extra instructions to store register contents in memory.

There are seven basic addressing modes:

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

The register indirect addressing modes include postincrement, predecrement, and offset capability. The program counter indirect mode also has index and offset capabilities. In addition to these addressing modes, many instructions implicitly specify the use of the status register, stack pointer, and/or program counter.

4.6 Processing States

The processor is always in one of four processing states: normal, exception, halted, or background. The normal processing state is associated with instruction execution; the bus is used to fetch instructions and operands and to store results.

The exception processing state is associated with interrupts, trap instructions, tracing, and other exception conditions. The exception may be internally generated explicitly by an instruction or by an unusual condition arising during the execution of an instruction. Exception processing can be forced externally by an interrupt, a bus error, or a reset.

The halted processing state is an indication of catastrophic hardware failure. For example, if during the exception processing of a bus error another bus error occurs, the processor assumes that the system is unusable and halts.

The background processing state is initiated by breakpoints, execution of special instructions, or a double bus fault. Background processing is enabled by pulling $\overline{\text{BKPT}}$ low during $\overline{\text{RESET}}$. Background processing allows interactive debugging of the system via a simple serial interface.

4.7 Privilege Levels

The processor operates at one of two levels of privilege: user or supervisor. Not all instructions are permitted to execute at the user level, but all instructions are available at the supervisor level. Effective use of privilege level can protect system resources from uncontrolled access. The state of the S bit in the status register determines the privilege level and whether the user stack pointer (USP) or supervisor stack pointer (SSP) is used for stack operations.

4.8 Instructions

The CPU32 instruction set is summarized in **Table 4-2**. The instruction set of the CPU32 is very similar to that of the MC68020. Two new instructions have been added to facilitate controller applications: low-power stop (LPSTOP) and table lookup and interpolate (TBLS, TBLSN, TBLU, TBLUN).

Table 4-1 shows the MC68020 instructions that are not implemented on the CPU32.

Table 4-1 Unimplemented MC68020 Instructions

BFxx	—	Bit Field Instructions (BFCHG, BFCLR, BFEXTS, BFEXTU, BFFFO, BFINS, BFSET, BFTST)
CALLM, RTM	—	Call Module, Return Module
CAS, CAS2	—	Compare and Swap (Read-Modify-Write Instructions)
cpxxx	—	Coprocessor Instructions (cpBcc, cpDBcc, cpGEN)
PACK, UNPK	—	Pack, Unpack BCD Instructions
Memory	—	Memory Indirect Addressing Modes

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

Table 4-4 summarizes the processing of each source for both enabled and disabled cases. As shown in **Table 4-4**, the $\overline{\text{BKPT}}$ instruction never causes a transition into BDM.

Table 4-4 BDM Source Summary

Source	BDM Enabled	BDM Disabled
$\overline{\text{BKPT}}$	Background	Breakpoint Exception
Double Bus Fault	Background	Halted
BGND Instruction	Background	Illegal Instruction
$\overline{\text{BKPT}}$ Instruction	Opcode Substitution/ Illegal Instruction	Opcode Substitution/ Illegal Instruction

4.10.4.1 External $\overline{\text{BKPT}}$ Signal

Once enabled, BDM is initiated whenever assertion of $\overline{\text{BKPT}}$ is acknowledged. If BDM is disabled, a breakpoint exception (vector $0C) is acknowledged. The $\overline{\text{BKPT}}$ input has the same timing relationship to the data strobe trailing edge as does read cycle data. There is no breakpoint acknowledge bus cycle when BDM is entered.

4.10.4.2 BGND Instruction

An illegal instruction, $4AFA, is reserved for use by development tools. The CPU32 defines $4AFA (BGND) to be a BDM entry point when BDM is enabled. If BDM is disabled, an illegal instruction trap is acknowledged.

4.10.4.3 Double Bus Fault

The CPU32 normally treats a double bus fault, or two bus faults in succession, as a catastrophic system error, and halts. When this condition occurs during initial system debug (a fault in the reset logic), further debugging is impossible until the problem is corrected. In BDM, the fault can be temporarily bypassed, so that the origin of the fault can be isolated and eliminated.

4.10.4.4 Peripheral Breakpoints

CPU32 peripheral breakpoints are implemented in the same way as external breakpoints — peripherals request breakpoints by asserting the $\overline{\text{BKPT}}$ signal. Consult the appropriate peripheral user's manual for additional details on the generation of peripheral breakpoints.

4.10.5 Entering BDM

When the processor detects a breakpoint or a double bus fault, or decodes a BGND instruction, it suspends instruction execution and asserts the FREEZE output. This is the first indication that the processor has entered BDM. Once FREEZE has been asserted, the CPU enables the serial communication hardware and awaits a command.

The CPU writes a unique value indicating the source of BDM transition into temporary register A (ATEMP) as part of the process of entering BDM. A user can poll ATEMP and determine the source (refer to **Table 4-5**) by issuing a read system register command (RSREG). ATEMP is used in most debugger commands for temporary storage

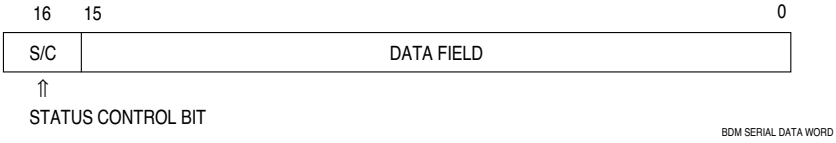

Figure 4-11 BDM Serial Data Word

Table 4-7 CPU Generated Message Encoding

Bit 16	Data	Message Type
0	XXXX	Valid Data Transfer
0	FFFF	Command Complete; Status OK
1	0000	Not Ready with Response; Come Again
1	0001	BERR Terminated Bus Cycle; Data Invalid
1	FFFF	Illegal Command

Command and data transfers initiated by the development system should clear bit 16. The current implementation ignores this bit; however, Motorola reserves the right to use this bit for future enhancements.

4.10.10 Recommended BDM Connection

In order to provide for use of development tools when an MCU is installed in a system, Motorola recommends that appropriate signal lines be routed to a male Berg connector or double-row header installed on the circuit board with the MCU, as shown in the following figure.

Figure 4-12 BDM Connector Pinout

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 ($\overline{\text{BERR}}$), and a spurious interrupt exception is taken.

Chip-select logic can also be used to generate internal $\overline{\text{AVEC}}$ or $\overline{\text{DSACK}}$ 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 $\overline{\text{AVEC}}$ or $\overline{\text{DSACK}}$ 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 $\overline{\text{IRQ}}$ 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.

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).

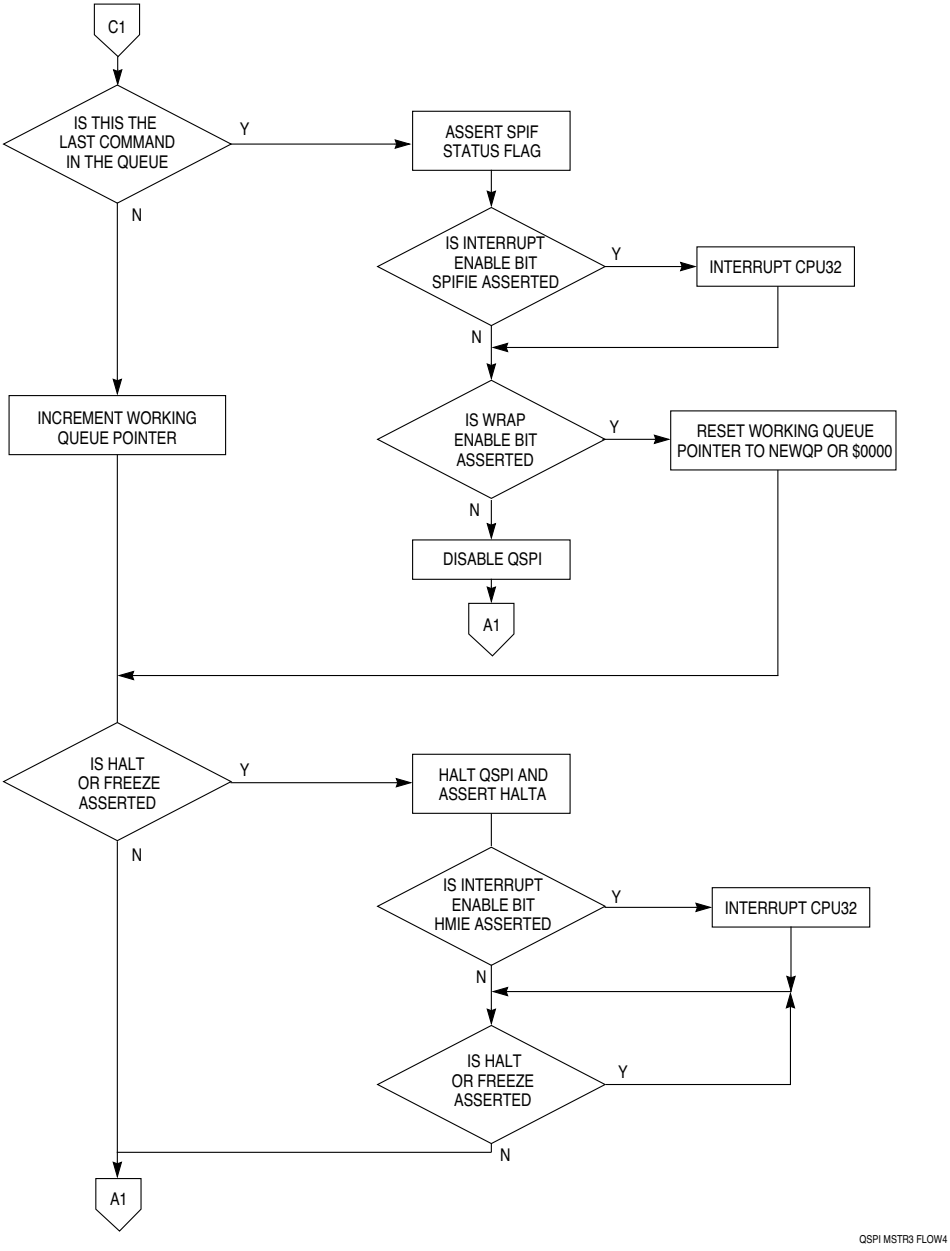

Figure 9-7 Flowchart of QSPI Master Operation (Part 3)

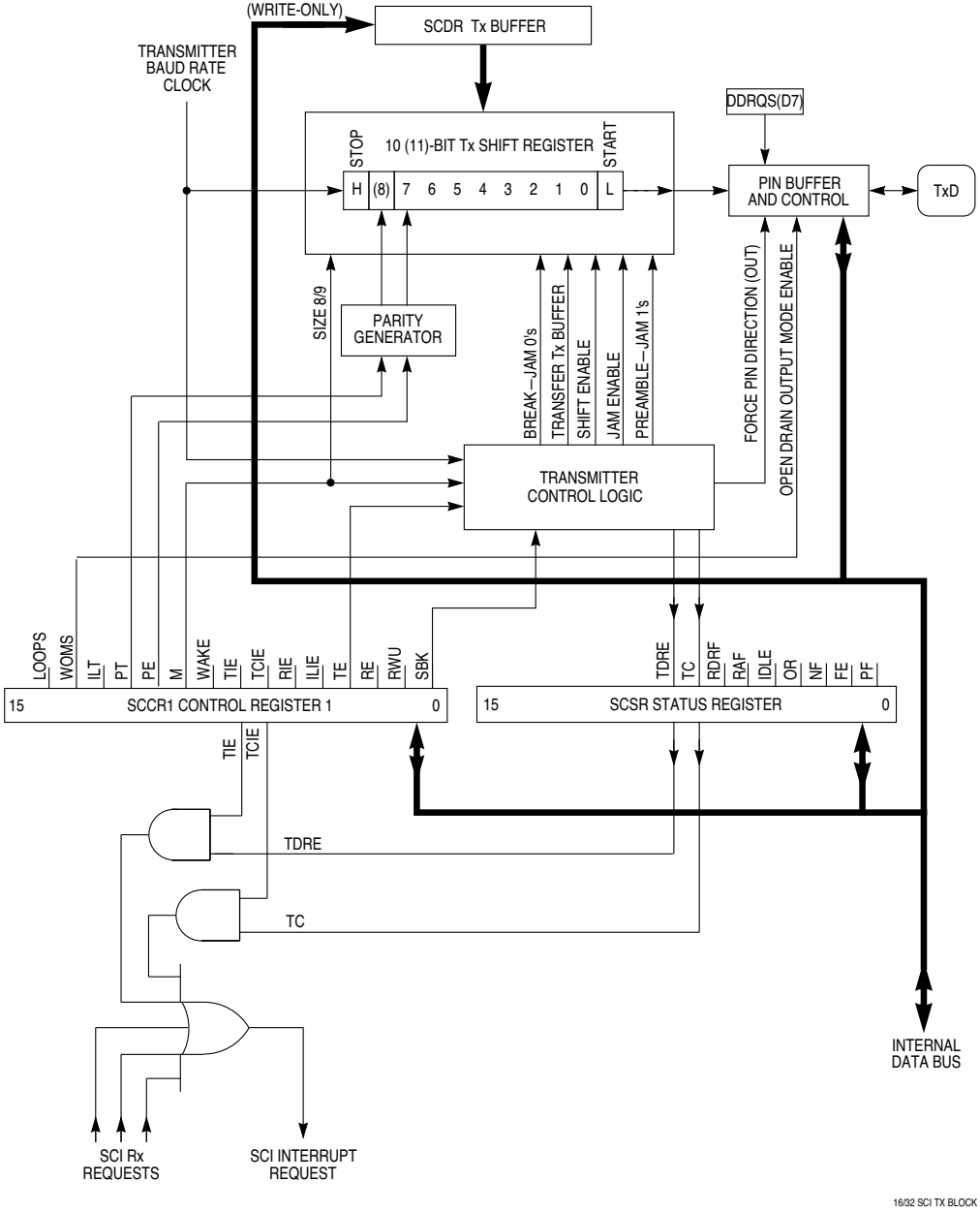

Figure 9-10 SCI Transmitter Block Diagram

- Software initiated single-scan mode
  - — Software can initiate the execution of a scan sequence for queue 1 or 2 by selecting this mode, and setting the single-scan enable bit in QACR1 or QACR2. A trigger event is generated internally and the QADC immediately begins execution of the first CCW in the queue. If a pause is encountered, queue execution ceases momentarily while another trigger event is generated internally, and then execution continues. While the time to internally generate and act on a trigger event is very short, software can momentarily read the status conditions, indicating that the queue is paused.
  - — The QADC automatically performs the conversions in the queue until an end-of-queue condition is encountered. The queue remains idle until software again sets the single-scan enable bit. The trigger overrun flag is never set while in this mode.
- External trigger rising or falling edge single-scan mode
  - — This mode is a variation of the external trigger continuous-scan mode. It is available for both queue 1 and queue 2. Software programs the external trigger to be either a rising or a falling edge. Software must also set the single-scan enable bit for the queue in order for the scan to take place. The first external trigger edge causes the queue to be executed one time. Each CCW is read and the indicated conversions are performed until an end-of-queue condition is encountered. After the queue scan is complete, the QADC clears the single-scan enable bit. Software may set the single-scan enable bit again to allow another scan of the queue to be initiated by the next external trigger edge.
- Interval timer single-scan mode
  - — In addition to the above modes, queue 2 can also be programmed for the interval timer single-scan mode. The queue operating mode for queue 2 is selected by the MQ2 field in QACR2.
  - — When this mode is selected and software sets the single-scan enable bit in QACR2, the periodic/interval timer begins counting. The timer interval can range from 2^7 to 2^{17} QCLK cycles in binary multiples. When the time interval expires, a trigger event is generated internally to start the queue. The timer is reloaded and begins counting again. Meanwhile, the QADC begins execution with the first CCW in queue 2.
  - — The QADC automatically performs the conversions in the queue until a pause or an end-of-queue condition is encountered. When a pause is encountered, queue execution stops until the timer interval expires again; queue execution then continues. When an end of queue condition is encountered, the timer is held in reset and the single-scan enable bit is cleared.
  - — Software may set the single-scan enable bit again, allowing another scan of the queue to be initiated by the interval timer. The interval timer generates a trigger event whenever the time interval elapses. The trigger event may cause queue execution to continue following a pause, or may be considered a trigger overrun if the queue is currently executing.

To prepare the QADC for a scan sequence, the desired channel conversions are written to the CCW table. Software establishes the criteria for initiating the queue execution by programming queue operating mode. The queue operating mode determines what type of trigger event initiates queue execution.

A scan sequence may be initiated by the following trigger events:

- A software command
- Expiration of the periodic/interval timer
- An external trigger signal

Software also specifies whether the QADC is to perform a single pass through the queue or is to scan continuously. When a single-scan mode is selected, queue execution begins when software sets the single-scan enable bit. When a continuous-scan mode is selected, the queue remains active in the selected queue operating mode after the QADC completes each queue scan sequence.

During queue execution, the QADC reads each CCW from the active queue and executes conversions in four stages:

1. Initial sample
2. Transfer
3. Final sample
4. Resolution

During initial sample, the selected input channel is connected to the sample capacitor at the input of the sample buffer amplifier.

During the transfer period, the sample capacitor is disconnected from the multiplexer, and the stored voltage is buffered and transferred to the RC DAC array.

During the final sample period, the sample capacitor and amplifier are bypassed, and the multiplexer input charges the RC DAC array directly. Each CCW specifies a final input sample time of 2, 4, 8, or 16 QCLK cycles. When an analog-to-digital conversion is complete, the result is written to the corresponding location in the result word table. The QADC continues to sequentially execute each CCW in the queue until the end of the queue is detected or a pause bit is found in a CCW.

When the pause bit is set in the current CCW, the QADC stops execution of the queue until a new trigger event occurs. The pause status flag bit is set, which may generate an interrupt request to notify software that the queue has reached the pause state. When the next trigger event occurs, the paused state ends, and the QADC continues to execute each CCW in the queue until another pause is encountered or the end of the queue is detected.

An end-of-queue condition is indicated as follows:

- The CCW channel field is programmed with 63 ($3F) to specify the end of the queue.
- The end of queue 1 is implied by the beginning of queue 2, which is specified in the BQ2 field in QACR2.
- The physical end of the queue RAM space defines the end of either queue.

When a match or input capture event requiring service occurs, the affected channel generates a service request to the scheduler. The scheduler determines the priority of the request and assigns the channel to the microengine at the first available time. The microengine performs the function defined by the content of the control store or emulation RAM, using parameters from the parameter RAM.

11.3.1 Event Timing

Match and capture events are handled by independent channel hardware. This provides an event accuracy of one time-base clock period, regardless of the number of channels that are active. An event normally causes a channel to request service. The time needed to respond to and service an event is determined by which channels and the number of channels requesting service, the relative priorities of the channels requesting service, and the microcode execution time of the active functions. Worst-case event service time (latency) determines TPU performance in a given application. Latency can be closely estimated. For more information, refer to the *TPU Reference Manual* (TPURM/AD)

11.3.2 Channel Orthogonality

Most timer systems are limited by the fixed number of functions assigned to each pin. All TPU channels contain identical hardware and are functionally equivalent in operation, so that any channel can be configured to perform any time function. Any function can operate on the calling channel, and, under program control, on another channel determined by the program or by a parameter. The user controls the combination of time functions.

11.3.3 Interchannel Communication

The autonomy of the TPU is enhanced by the ability of a channel to affect the operation of one or more other channels without CPU32 intervention. Interchannel communication can be accomplished by issuing a link service request to another channel, by controlling another channel directly, or by accessing the parameter RAM of another channel.

11.3.4 Programmable Channel Service Priority

The TPU provides a programmable service priority level to each channel. Three priority levels are available. When more than one channel of a given priority requests service at the same time, arbitration is accomplished according to channel number. To prevent a single high-priority channel from permanently blocking other functions, other service requests of the same priority are performed in channel order after the lowest-numbered, highest-priority channel is serviced.

11.3.5 Coherency

For data to be coherent, all available portions of the data must be identical in age, or must be logically related. As an example, consider a 32-bit counter value that is read and written as two 16-bit words. The 32-bit value is read-coherent only if both 16-bit portions are updated at the same time, and write-coherent only if both portions take

2. Initialize IARB[3:0] to a non-zero value in CANMCR.
3. Set the required mask bits in the IMASK register (for all message buffer interrupts), in CANCTRL0 (for bus off and error interrupts), and in CANMCR for the WAKE interrupt.

E. Negate the HALT bit in the module configuration register
1. At this point, the TouCAN will attempt to synchronize with the CAN bus.

NOTE

In both the transmit and receive processes, the first action in preparing a message buffer should be to deactivate the buffer by setting its code field to the proper value. This requirement is mandatory to assure data coherency.

13.5.3 Transmit Process

The transmit process includes preparing a message buffer for transmission, as well as the internal steps performed by the TouCAN to decide which message to transmit. For the user, this involves loading the message and ID to be transmitted into a message buffer and then activating that buffer as an active transmit buffer. Once this is done, the TouCAN will perform all additional steps necessary to transmit the message onto the CAN bus.

The user should prepare/change a message buffer for transmission by executing the following steps.

1. Write the control/status word to hold the transmit buffer inactive (code = %1000)
2. Write the ID_HIGH and ID_LOW words
3. Write the data bytes
4. Write the control/status word (active TX code, TX length)

NOTE

Steps 1 and 4 are mandatory to ensure data coherency while preparing a message buffer for transmission.

Once an active transmit code is written to a transmit message buffer, that buffer will begin participating in an internal arbitration process as soon as the CAN bus is sensed to be free by the receiver, or at the inter-frame space. If there are multiple messages awaiting transmission, this internal arbitration process selects the message buffer from which the next frame is transmitted.

When this process is over, and a message buffer is selected for transmission, the frame from that message buffer is transferred to the serial message buffer for transmission.

While transmitting, the TouCAN will transmit no more than eight data bytes, even if the transmit length contains a value greater than eight.

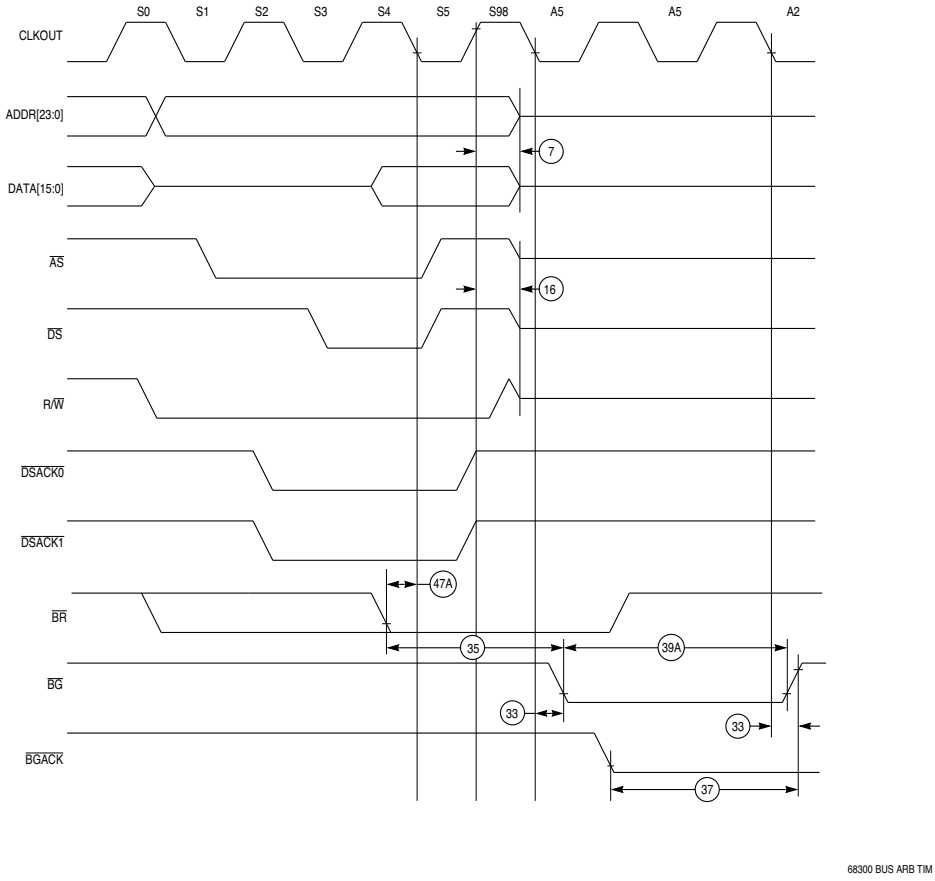

Figure A-8 Bus Arbitration Timing Diagram — Active Bus Case

Table B-2 MC68376 Ordering Information

Part Number	Package Type	Frequency (MHz)	TPU	Mask ROM	Temperature	Package Order Quantity	Order Number
MC68376	160-pin QFP	20.97 MHz	A	Blank	–40 to +85 °C	2	SPMC68376BACFT20
						24	MC68376BACFT20
						120	MC68376BACFT20B1
					–40 to +105 °C	2	SPMC68376BAVFT20
						24	MC68376BAVFT20
						120	MC68376BAVFT20B1
					–40 to +125 °C	2	SPMC68376BAMFT20
						24	MC68376BAMFT20
						120	MC68376BAMFT20B1
			G	Blank	–40 to +85 °C	2	SPMC68376BGCFT20
						24	MC68376BGCFT20
						120	MC68376BGCFT20B1
					–40 to +105 °C	2	SPMC68376BGVFT20
						24	MC68376BGVFT20
						120	MC68376BGVFT20B1
					–40 to +125 °C	2	SPMC68376BGMFT20
						24	MC68376BGMFT20
						120	MC68376BGMFT20B1

This field only affects the response of chip-selects and does not affect interrupt recognition by the CPU32.

$\overline{\text{AVEC}}$ — Autovector Enable

This field selects one of two methods of acquiring an interrupt vector during an interrupt acknowledge cycle. It is not usually used with a chip-select pin.

0 = External interrupt vector enabled
1 = Autovector enabled

If the chip select is configured to trigger on an interrupt acknowledge cycle (SPACE[1:0] = %00) and the $\overline{\text{AVEC}}$ field is set to one, the chip-select automatically generates $\overline{\text{AVEC}}$ in response to the interrupt acknowledge cycle. Otherwise, the vector must be supplied by the requesting device.

D.2.22 Master Shift Registers

TSTMSRA — Master Shift Register A **$YFFA30**

Used for factory test only.

TSTMSRB — Master Shift Register B **$YFFA32**

Used for factory test only.

D.2.23 Test Module Shift Count Register

TSTSC — Test Module Shift Count **$YFFA34**

Used for factory test only.

D.2.24 Test Module Repetition Count Register

TSTRC — Test Module Repetition Count **$YFFA36**

Used for factory test only.

D.2.25 Test Submodule Control Register

CREG — Test Submodule Control Register **$YFFA38**

Used for factory test only.

D.2.26 Distributed Register

DREG — Distributed Register **$YFFA3A**

Used for factory test only.

$$\text{SCK Baud Rate} = \frac{f_{sys}}{2 \times \text{SPBR[7:0]}}$$

or

$$\text{SPBR[7:0]} = \frac{f_{sys}}{2 \times \text{SCK Baud Rate Desired}}$$

Giving SPBR[7:0] a value of zero or one disables the baud rate generator. SCK is disabled and assumes its inactive state value. No serial transfers occur. At reset, the SCK baud rate is initialized to one eighth of the system clock frequency.

D.6.12 QSPI Control Register 1

SPCR1 — QSPI Control Register 1 **$YFFC1A**

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

SPCR1 enables the QSPI and specified transfer delays. The CPU32 has read/write access to SPCR1, but the QSM has read access only to all bits except SPE. SPCR1 must be written last during initialization because it contains SPE. Writing a new value to SPCR1 while the QSPI is enabled disrupts operation.

SPE — QSPI Enable

0 = QSPI is disabled. QSPI pins can be used for general-purpose I/O.
1 = QSPI is enabled. Pins allocated by PQSPAR are controlled by the QSPI.

DSCKL[6:0] — Delay before SCK

When the DSCK bit is set in a command RAM byte, this field determines the length of the delay from PCS valid to SCK transition. PCS can be any of the four peripheral chip-select pins. The following equation determines the actual delay before SCK:

$$\text{PCS to SCK Delay} = \frac{\text{DSCKL[6:0]}}{f_{sys}}$$

where DSCKL[6:0] equals is in the range of 1 to 127.

When DSCK is zero in a command RAM byte, then DSCKL[6:0] is not used. Instead, the PCS valid to SCK transition is one-half the SCK period.

CH[15:0] — Encoded Host Sequence

The host sequence field selects the mode of operation for the time function selected on a given channel. The meaning of the host sequence bits depends on the time function specified.

D.8.9 Host Service Request Registers

HSSR0 — Host Service Request Register 0 **$YFFE18**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CH 15		CH 14		CH 13		CH 12		CH 11		CH 10		CH 9		CH 8	
RESET:															
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

HSSR1 — Host Service Request Register 1 **$YFFE1A**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CH 7		CH 6		CH 5		CH 4		CH 3		CH 2		CH 1		CH 0	
RESET:															
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

CH[15:0] — Encoded Type of Host Service

The host service request field selects the type of host service request for the time function selected on a given channel. The meaning of the host service request bits depends on the time function specified.

A host service request field cleared to %00 signals the host that service is completed by the microengine on that channel. The host can request service on a channel by writing the corresponding host service request field to one of three non-zero states. The CPU32 should monitor the host service request register until the TPU clears the service request to %00 before any parameters are changed or a new service request is issued to the channel.

D.8.10 Channel Priority Registers

CPR0 — Channel Priority Register 0 **$YFFE1C**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CH 15		CH 14		CH 13		CH 12		CH 11		CH 10		CH 9		CH 8	
RESET:															
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

CPR1 — Channel Priority Register 1 **$YFFE1E**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CH 7		CH 6		CH 5		CH 4		CH 3		CH 2		CH 1		CH 0	
RESET:															
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0