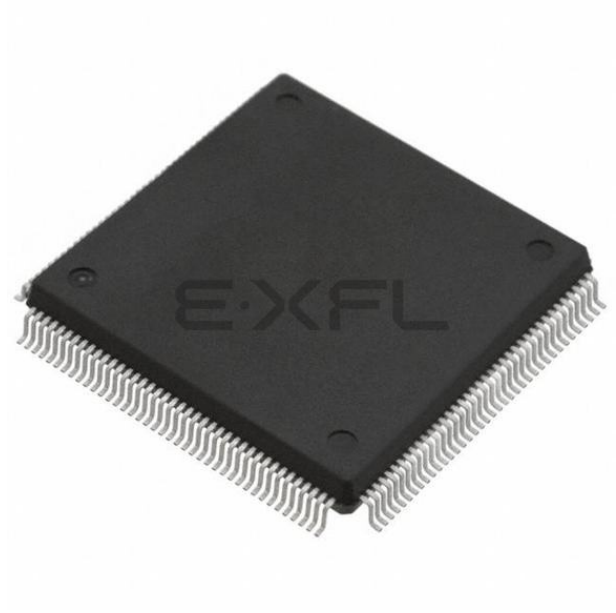


Welcome to E-XFL.COM

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/mc68376bgcft20

LIST OF ILLUSTRATIONS
(Continued)

Figure	Title	Page
A-6	Fast Termination Read Cycle Timing Diagram	A-13
A-7	Fast Termination Write Cycle Timing Diagram	A-14
A-8	Bus Arbitration Timing Diagram — Active Bus Case	A-15
A-9	Bus Arbitration Timing Diagram — Idle Bus Case	A-16
A-10	Show Cycle Timing Diagram	A-17
A-11	Chip-Select Timing Diagram	A-18
A-12	Reset and Mode Select Timing Diagram	A-18
A-13	Background Debugging Mode Timing — Serial Communication	A-20
A-14	Background Debugging Mode Timing — Freeze Assertion	A-20
A-15	ECLK Timing Diagram	A-22
A-16	QSPI Timing — Master, CPHA = 0	A-24
A-17	QSPI Timing — Master, CPHA = 1	A-24
A-18	QSPI Timing — Slave, CPHA = 0	A-25
A-19	QSPI Timing — Slave, CPHA = 1	A-25
A-20	TPU Timing Diagram	A-26
B-1	MC68336 Pin Assignments for 160-Pin Package	B-1
B-2	MC68376 Pin Assignments for 160-Pin Package	B-2
B-3	160-Pin Package Dimensions	B-3
D-1	User Programming Model	D-2
D-2	Supervisor Programming Model Supplement	D-3
D-3	TouCAN Message Buffer Address Map	D-85

Table 3-1 MC68336/376 Pin Characteristics

Pin Mnemonic	Output Driver	Input Synchronized	Input Hysteresis	Discrete I/O	Port Designation
ADDR23/$\overline{CS10}$/ECLK	A	Yes	No	O	—
ADDR[22:19]/$\overline{CS[9:6]}$	A	Yes	No	O	PC[6:3]
ADDR[18:0]	A	Yes	No	—	—
AN[51:48]	—	Yes[1]	Yes	I	PQB[7:4]
AN[3:0]/AN[w, x, y, z]	—	Yes[1]	Yes	I	PQB[3:0]
AN[59:57]	Ba	Yes	Yes	I/O	PQA[7:5]
AN[56:55]/ETRIG[2:1]	Ba	Yes	Yes	I/O	PQA[4:3]
AN[54:52]/MA[2:0]	Ba	Yes	Yes	I/O	PQA[2:0]
$\overline{AS}$	B	Yes	Yes	I/O	PE5
$\overline{AVEC}$	B	Yes	No	I/O	PE2
$\overline{BERR}$	B	Yes	No	—	—
$\overline{BG}$/$\overline{CS1}$	B	—	—	—	—
$\overline{BGACK}$/$\overline{CS2}$	B	Yes	No	—	—
$\overline{BKPT}$/DSCLK	—	Yes	Yes	—	—
$\overline{BR}$/$\overline{CS0}$	B	Yes	No	O	—
CLKOUT	A	—	—	—	—
CANRX0 (MC68376 Only)	—	Yes	Yes	—	—
CANTX0 (MC68376 Only)	Bo	—	—	—	—
$\overline{CSBOOT}$	B	—	—	—	—
CTD[10:9]/[4:3]	A	Yes	Yes	I/O	—
CPWM[8:5]	A	—	—	O	—
CTM2C	—	Yes	Yes	I	—
DATA[15:0]	Aw	Yes[1]	No	—	—
$\overline{DS}$	B	Yes	Yes	I/O	PE4
$\overline{DSACK[1:0]}$	B	Yes	No	I/O	PE[1:0]
EXTAL[2]	—	—	Special	—	—
FC[2:0]/$\overline{CS[5:3]}$	A	Yes	No	O	PC[2:0]
FREEZE/QUOT	A	—	—	—	—
$\overline{IPIPE}$/DSO	A	—	—	O	—
$\overline{IFETCH}$/DSI	A	Yes	Yes	—	—
$\overline{HALT}$	Bo	Yes	No	—	—
$\overline{IRQ[7:1]}$	B	Yes	Yes	I/O	PF[7:1]
MISO	Bo	Yes[1]	Yes	I/O	PQS0
MODCLK	B	Yes[1]	Yes	I/O	PF0
MOSI	Bo	Yes[1]	Yes	I/O	PQS1
PCS0/$\overline{SS}$	Bo	Yes[1]	Yes	I/O	PQS3
PCS[3:1]	Bo	Yes[1]	Yes	I/O	PQS[6:4]
R/$\overline{W}$	A	Yes	No	—	—
$\overline{RESET}$	Bo	Yes	Yes	—	—
$\overline{RMC}$	B	Yes	Yes	I/O	PE3
RXD	—	No	Yes	—	—
SCK	Bo	Yes[1]	Yes	I/O	PQS2

SECTION 4 CENTRAL PROCESSOR UNIT

The CPU32, the instruction processing module of the M68300 family, is based on the industry-standard MC68000 processor. It has many features of the MC68010 and MC68020, as well as unique features suited for high-performance controller applications. This section is an overview of the CPU32. For detailed information concerning CPU operation, refer to the *CPU32 Reference Manual* (CPU32RM/AD).

4.1 General

Ease of programming is an important consideration in using a microcontroller. The CPU32 instruction format reflects a philosophy emphasizing register-memory interaction. There are eight multifunction data registers and seven general-purpose addressing registers.

All data resources are available to all operations requiring those resources. The data registers readily support 8-bit (byte), 16-bit (word), and 32-bit (long-word) operand lengths for all operations. Word and long-word operations support address manipulation. Although the program counter (PC) and stack pointers (SP) are special-purpose registers, they are also available for most data addressing activities. Ease of program checking and diagnosis is further enhanced by trace and trap capabilities at the instruction level.

A block diagram of the CPU32 is shown in **Figure 4-1**. The major blocks operate in a highly independent fashion that maximizes concurrency of operation while managing the essential synchronization of instruction execution and bus operation. The bus controller loads instructions from the data bus into the decode unit. The sequencer and control unit provide overall chip control, managing the internal buses, registers, and functions of the execution unit.

4.10.1 M68000 Family Development Support

All M68000 Family members include features to facilitate applications development. These features include the following:

Trace on Instruction Execution — M68000 Family processors include an instruction-by-instruction tracing facility as an aid to program development. The MC68020, MC68030, MC68040, and CPU32 also allow tracing only of those instructions causing a change in program flow. In the trace mode, a trace exception is generated after an instruction is executed, allowing a debugger program to monitor the execution of a program under test.

Breakpoint Instruction — An emulator may insert software breakpoints into the target code to indicate when a breakpoint has occurred. On the MC68010, MC68020, MC68030, and CPU32, this function is provided via illegal instructions, $4848–$484F, to serve as breakpoint instructions.

Unimplemented Instruction Emulation — During instruction execution, when an attempt is made to execute an illegal instruction, an illegal instruction exception occurs. Unimplemented instructions (F-line, A-line, . . .) utilize separate exception vectors to permit efficient emulation of unimplemented instructions in software.

4.10.2 Background Debug Mode

Microcomputer systems generally provide a debugger, implemented in software, for system analysis at the lowest level. The background debug mode (BDM) on the CPU32 is unique in that the debugger has been implemented in CPU microcode.

BDM incorporates a full set of debugging options: registers can be viewed or altered, memory can be read or written to, and test features can be invoked.

A resident debugger simplifies implementation of an in-circuit emulator. In a common setup (refer to **Figure 4-8**), emulator hardware replaces the target system processor. A complex, expensive pod-and-cable interface provides a communication path between the target system and the emulator.

By contrast, an integrated debugger supports use of a bus state analyzer (BSA) for incircuit emulation. The processor remains in the target system (refer to **Figure 4-9**) and the interface is simplified. The BSA monitors target processor operation and the on-chip debugger controls the operating environment. Emulation is much “closer” to target hardware, and many interfacing problems (for example, limitations on high-frequency operation, AC and DC parametric mismatches, and restrictions on cable length) are minimized.

— it is imperative that the RSREG command be the first command issued after transition into BDM.

Table 4-5 Polling the BDM Entry Source

Source	ATEMP[31:16]	ATEMP[15:0]
Double Bus Fault	SSW[1]	$FFFF
BGND Instruction	$0000	$0001
Hardware Breakpoint	$0000	$0000

NOTES:
1. Special status word (SSW) is described in detail in the *CPU32 Reference Manual* (CPU32RM/AD).

A double bus fault during initial stack pointer/program counter (SP/PC) fetch sequence is distinguished by a value of $FFFFFFFF in the current instruction PC. At no other time will the processor write an odd value into this register.

4.10.6 BDM Commands

BDM commands consist of one 16-bit operation word and can include one or more 16-bit extension words. Each incoming word is read as it is assembled by the serial interface. The microcode routine corresponding to a command is executed as soon as the command is complete. Result operands are loaded into the output shift register to be shifted out as the next command is read. This process is repeated for each command until the CPU returns to normal operating mode. **Table 4-6** is a summary of background mode commands.

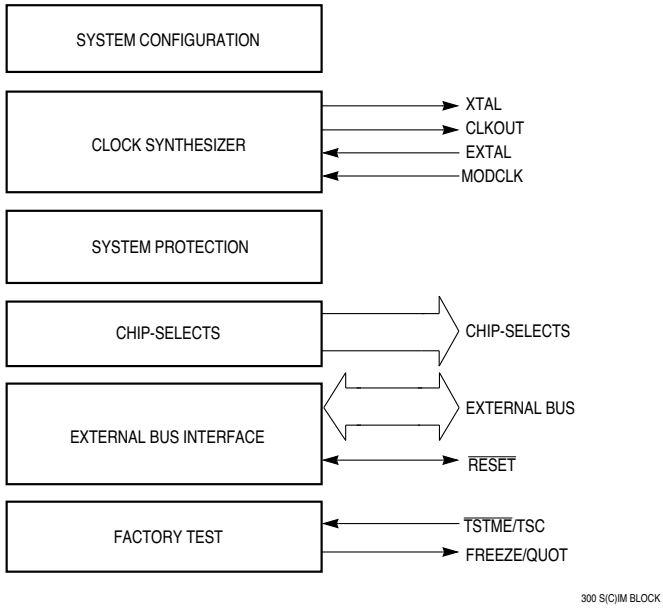


Figure 5-1 System Integration Module Block Diagram

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.

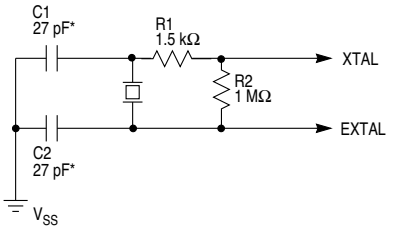
5.2.1 Module Mapping

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.

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.

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 $\overline{\text{BERR}}$, and the CPU32 generates the spurious interrupt vector number.
   2. The dominant interrupt source (external or internal) supplies a vector number and $\overline{\text{DSACK}}$ signals appropriate to the access. The CPU32 acquires the vector number.
   3. The $\overline{\text{AVEC}}$ 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 $\overline{\text{BERR}}$ 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 $\overline{\text{AVEC}}$ or $\overline{\text{DSACK}}$, 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.

7.5 Low-Power Stop Mode Operation

Low-power stop mode minimizes MCU power consumption. Setting the STOP bit in MRMCR places the MRM in low-power stop mode. In low-power stop mode, the array cannot be accessed. The reset state of STOP is the complement of the logic state of DATA14 during reset. Low-power stop mode is exited by clearing STOP.

7.6 ROM Signature

Signature registers RSIGHI and RSIGLO contain a user-specified mask-programmed signature pattern. A special signature algorithm allows the user to verify ROM array content.

7.7 Reset

The state of the MRM following reset is determined by the default values programmed into the MRMCR $\overline{\text{BOOT}}$, LOCK, ASPC[1:0], and WAIT[1:0] bits. The default array base address is determined by the values programmed into ROMBAL and ROMBAH.

When the mask programmed value of the MRMCR $\overline{\text{BOOT}}$ bit is zero, the contents of MRM bootstrap words ROMBS[0:3] are used as reset vectors. When the mask programmed value of the MRMCR $\overline{\text{BOOT}}$ bit is one, reset vectors are fetched from external memory, and system integration module chip-select logic is used to assert the boot ROM select signal $\overline{\text{CSBOOT}}$. Refer to **5.9.4 Chip-Select Reset Operation** for more information concerning external boot ROM selection.

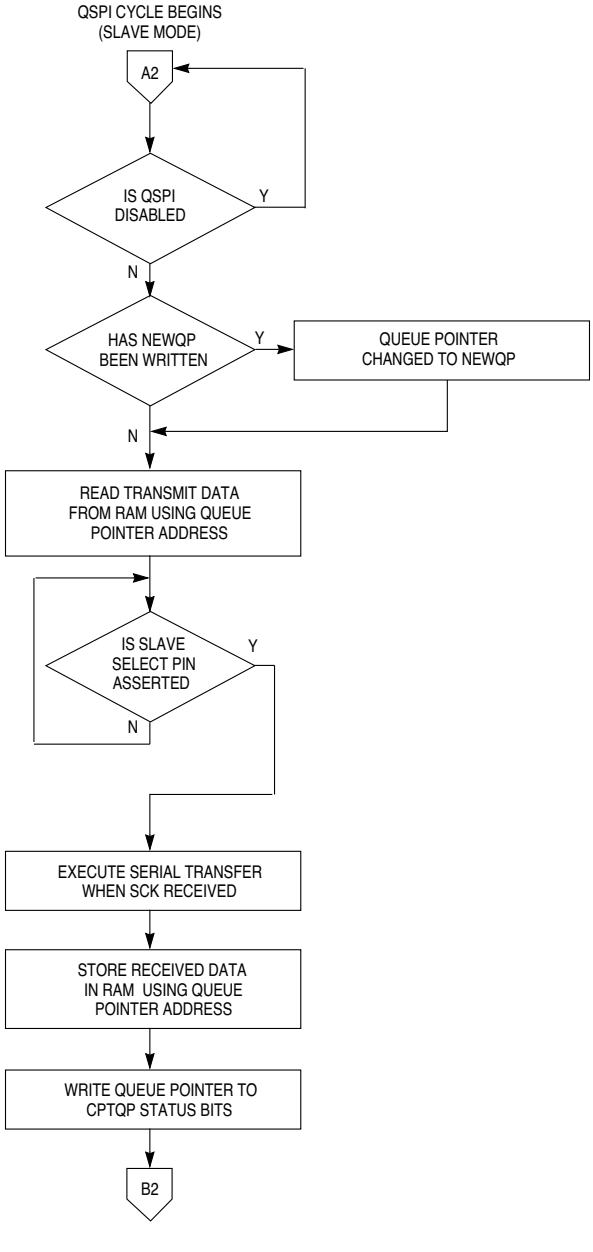


Figure 9-8 Flowchart of QSPI Slave Operation (Part 1)

Delay after transfer can be used to provide a peripheral deselect interval. A delay can also be inserted between consecutive transfers to allow serial A/D converters to complete conversion. Writing a value to DTL[7:0] in SPCR1 specifies a delay period. The DT bit in each command RAM byte determines whether the standard delay period (DT = 0) or the specified delay period (DT = 1) is used. The following expression is used to calculate the delay:

$$\text{Delay after Transfer} = \frac{32 \times \text{DTL[7:0]}}{\text{System Clock}}$$

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

A zero value for DTL[7:0] causes a delay-after-transfer value of 8192/System Clock.

$$\text{Standard Delay after Transfer} = \frac{17}{\text{System Clock}}$$

Adequate delay between transfers must be specified for long data streams because the QSPI requires time to load a transmit RAM entry for transfer. Receiving devices need at least the standard delay between successive transfers. If the system clock is operating at a slower rate, the delay between transfers must be increased proportionately.

Operation is initiated by setting the SPE bit in SPCR1. Shortly after SPE is set, the QSPI executes the command at the command RAM address pointed to by NEWQP. Data at the pointer address in transmit RAM is loaded into the data serializer and transmitted. Data that is simultaneously received is stored at the pointer address in receive RAM.

When the proper number of bits have been transferred, the QSPI stores the working queue pointer value in CPTQP, increments the working queue pointer, and loads the next data for transfer from transmit RAM. The command pointed to by the incremented working queue pointer is executed next, unless a new value has been written to NEWQP. If a new queue pointer value is written while a transfer is in progress, that transfer is completed normally.

When the CONT bit in a command RAM byte is set, PCS pins are continuously driven in specified states during and between transfers. If the chip-select pattern changes during or between transfers, the original pattern is driven until execution of the following transfer begins. When CONT is cleared, the data in register PORTQS is driven between transfers. The data in PORTQS must match the inactive states of SCK and any peripheral chip-selects used.

When the QSPI reaches the end of the queue, it sets the SPIF flag. If the SPIFIE bit in SPCR2 is set, an interrupt request is generated when SPIF is asserted. At this point, the QSPI clears SPE and stops unless wrap-around mode is enabled.

9.4.3.9 Internal Loop

The LOOPS bit in SCCR1 controls a feedback path in the data serial shifter. When LOOPS is set, the SCI transmitter output is fed back into the receive serial shifter. TXD is asserted (idle line). Both transmitter and receiver must be enabled before entering loop mode.

9.5 QSM Initialization

After reset, the QSM remains in an idle state until initialized. A general guide for initialization follows.

A. Global
   1. Configuration QSMCR
      a.Write an interrupt arbitration priority value into the IARB field.
      b. Clear the FREEZE and/or STOP bits for normal operation.
   2. Configure QIVR and QILR
      a. Write QSPI/SCI interrupt vector number into QIVR.
      b. Write QSPI (ILSPI) and SCI (ILSCI) interrupt priorities into QILR.
   3. Configure PORTQS and DDRQS
      a. Write a data word to PORTQS.
      b. Set the direction of QSM pins used for I/O by writing to DDRQS.
   4. Assign pin functions by writing to the pin assignment register PQSPAR

B. Queued Serial Peripheral Interface
   1. Write appropriate values to QSPI command RAM and transmit RAM.
   2. Set up the SPCR0
      a. Set the bit in with the BR field.
      b. Determine clock phase (CPHA), and clock polarity (CPOL).
      c. Determine number of bits to be transferred in a serial operation (BITS[3:0]).
      d. Select master or slave operating mode (MSTR).
      e. Enable or disable wired-OR operation (WOMQ).
   3. Set up SPCR1
      a. Establish a delay following serial transfer by writing to the DTL field.
      b. Establish a delay before serial transfer by writing to the DSCKL field.
   4. Set up SPCR2
      a. Write an initial queue pointer value into the NEWQP field.
      b. Write a final queue pointer value into the ENDQP field.
      c. Enable or disable queue wrap-around (WREN).
      d. Set wrap-around address if enabled (WRTO).
      e. Enable or disable QSPI interrupt (SPIFIE).
   5. Set up SPCR3
      a. Enable or disable halt at end of queue (HALT).
      b. Enable or disable halt and mode fault interrupts (HMIE).
      c. Enable or disable loopback (LOOPQ).
   6. To enable the QSPI, set the SPE bit in SPCR1.

C. Serial Communication Interface
   1. Set up SCCR0
      a. Set the baud with the SCBR field.

2. Set up SCCR1
   a. Select serial mode (M)
   b. Enable use (PE) and type (PT) of parity check.
   c. Select use (RWU) and type (WAKE) of receiver wake-up.
   d. Enable idle-line detection (ILT) and interrupt (ILIE).
   e. Enable or disable wired-OR operation (WOMS).
   f. Enable or disable break transmission (SBK).
3. To receive:
   a. Set the receiver (RE) and receiver interrupt (RIE) bits in SCCR1.
4. To transmit:
   a. Set transmitter (TE) and transmitter interrupt (TIE) bits in SCCR1.
   b. Clear the TDRE and TC flags by reading SCSR and writing data to SCDR.

8.13 Interrupts

The QADC supports both polled and interrupt driven operation. Status bits in QASR reflect the operating condition of each queue and can optionally generate interrupts when enabled by the appropriate bits in QACR1 and/or QACR2.

8.13.1 Interrupt Sources

The QADC has four interrupt service sources, each of which is separately enabled. Each time the result is written for the last CCW in a queue, the completion flag for the corresponding queue is set, and when enabled, an interrupt request is generated. In the same way, each time the result is written for a CCW with the pause bit set, the queue pause flag is set, and when enabled, an interrupt request is generated.

Table 8-5 displays the status flag and interrupt enable bits which correspond to queue 1 and queue 2 activity.

Table 8-5 QADC Status Flags and Interrupt Sources

Queue	Queue Activity	Status Flag	Interrupt Enable Bit
Queue 1	Result written for the last CCW in queue 1	CF1	CIE1
	Result written for a CCW with pause bit set in queue 1	PF1	PIE1
Queue 2	Result written for the last CCW in queue 2	CF2	CIE2
	Result written for a CCW with pause bit set in queue 2	PF2	PIE2

Both polled and interrupt-driven QADC operations require that status flags must be cleared after an event occurs. Flags are cleared by first reading QASR with the appropriate flag bits set to one, then writing zeros to the flags that are to be cleared. A flag can be cleared only if the flag was a logic one at the time the register was read by the CPU. If a new event occurs between the time that the register is read and the time that it is written, the associated flag is not cleared.

8.13.2 Interrupt Register

The QADC interrupt register QADCINT specifies the priority level of QADC interrupt requests and the upper six bits of the vector number provided during an interrupt acknowledge cycle.

The values contained in the IRLQ1 and IRLQ2 fields in QADCINT determine the priority of QADC interrupt service requests. A value of %000 in either field disables the interrupts associated with that field. The interrupt levels for queue 1 and queue 2 may be different.

The IVB[7:2] bits specify the upper six bits of each QADC interrupt vector number. IVB[1:0] have fixed assignments for each of the four QADC interrupt sources. Refer to **8.13.3 Interrupt Vectors** for more information.

Arbitration is performed by means of serial assertion of IARB field bit values. The IARB of TPUMCR is initialized to $0 during reset.

When the TPU wins arbitration, it must respond to the CPU32 interrupt acknowledge cycle by placing an interrupt vector number on the data bus. The vector number is used to calculate displacement into the exception vector table. Vectors are formed by concatenating the 4-bit value of the CIBV field in TICR with the 4-bit number of the channel requesting interrupt service. Since the CIBV field has a reset value of $0, it must be assigned a value corresponding to the upper nibble of a block of 16 user-defined vector numbers before TPU interrupts are enabled. Otherwise, a TPU interrupt service request could cause the CPU32 to take one of the reserved vectors in the exception vector table.

For more information about the exception vector table, refer to **4.9 Exception Processing**. Refer to **5.8 Interrupts** for further information about interrupts.

11.4 A Mask Set Time Functions

The following paragraphs describe factory-programmed time functions implemented in the A mask set TPU microcode ROM. A complete description of the functions is beyond the scope of this manual. Refer to the *TPU Reference Manual* (TPURM/AD) for additional information.

11.4.1 Discrete Input/Output (DIO)

When a pin is used as a discrete input, a parameter indicates the current input level and the previous 15 levels of a pin. Bit 15, the most significant bit of the parameter, indicates the most recent state. Bit 14 indicates the next most recent state, and so on. The programmer can choose one of the three following conditions to update the parameter: 1) when a transition occurs, 2) when the CPU32 makes a request, or 3) when a rate specified in another parameter is matched. When a pin is used as a discrete output, it is set high or low only upon request by the CPU32.

Refer to TPU programming note *Discrete Input/Output (DIO) TPU Function* (TPUPN18/D) for more information.

11.4.2 Input Capture/Input Transition Counter (ITC)

Any channel of the TPU can capture the value of a specified TCR upon the occurrence of each transition or specified number of transitions and then generate an interrupt request to notify the CPU32. A channel can perform input captures continually, or a channel can detect a single transition or specified number of transitions, then cease channel activity until reinitialization. After each transition or specified number of transitions, the channel can generate a link to a sequential block of up to eight channels. The user specifies a starting channel of the block and the number of channels within the block. The generation of links depends on the mode of operation. In addition, after each transition or specified number of transitions, one byte of the parameter RAM (at an address specified by channel parameter) can be incremented and used as a flag to notify another channel of a transition.

Table D-26 Queue 2 Operating Modes

MQ2[4:0]	Queue 2 Operating Mode
00000	Disabled mode, conversions do not occur
00001	Software triggered single-scan mode (started with SSE2)
00010	External trigger rising edge single-scan mode (on ETRIG2 pin)
00011	External trigger falling edge single-scan mode (on ETRIG2 pin)
00100	Interval timer single-scan mode: interval = QCLK period x 2^7
00101	Interval timer single-scan mode: interval = QCLK period x 2^8
00110	Interval timer single-scan mode: interval = QCLK period x 2^9
00111	Interval timer single-scan mode: interval = QCLK period x 2^{10}
01000	Interval timer single-scan mode: interval = QCLK period x 2^{11}
01001	Interval timer single-scan mode: interval = QCLK period x 2^{12}
01010	Interval timer single-scan mode: interval = QCLK period x 2^{13}
01011	Interval timer single-scan mode: interval = QCLK period x 2^{14}
01100	Interval timer single-scan mode: interval = QCLK period x 2^{15}
01101	Interval timer single-scan mode: interval = QCLK period x 2^{16}
01110	Interval timer single-scan mode: interval = QCLK period x 2^{17}
01111	Reserved mode
10000	Reserved mode
10001	Software triggered continuous-scan mode (started with SSE2)
10010	External trigger rising edge continuous-scan mode (on ETRIG2 pin)
10011	External trigger falling edge continuous-scan mode (on ETRIG2 pin)
10100	Periodic timer continuous-scan mode: period = QCLK period x 2^7
10101	Periodic timer continuous-scan mode: period = QCLK period x 2^8
10110	Periodic timer continuous-scan mode: period = QCLK period x 2^9
10111	Periodic timer continuous-scan mode: period = QCLK period x 2^{10}
11000	Periodic timer continuous-scan mode: period = QCLK period x 2^{11}
11001	Periodic timer continuous-scan mode: period = QCLK period x 2^{12}
11010	Periodic timer continuous-scan mode: period = QCLK period x 2^{13}
11011	Periodic timer continuous-scan mode: period = QCLK period x 2^{14}
11100	Periodic timer continuous-scan mode: period = QCLK period x 2^{15}
11101	Periodic timer continuous-scan mode: period = QCLK period x 2^{16}
11110	Periodic timer continuous-scan mode: period = QCLK period x 2^{17}
11111	Reserved mode

RES — Queue 2 Resume

RES selects the resumption point after queue 2 is suspended by queue 1. If RES is changed during execution of queue 2, the change is not recognized until an end-of-queue condition is reached, or the queue operating mode of queue 2 is changed.

0 = After suspension, begin execution with the first CCW in queue 2 or the current subqueue.

1 = After suspension, begin execution with the aborted CCW in queue 2.

BQ2[5:0] — Beginning of Queue 2

The BQ2 field indicates the location in the CCW table where queue 2 begins. The BQ2 field also indicates the end of queue 1 and thus creates an end-of-queue condition for queue 1.

D.5.7 QADC Status Register

QASR — Status Register **$YFFF210**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CF1	PF1	CF2	PF2	TOR1	TOR2	QS[3:0]				CWP[5:0]					
RESET:															
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

CF1 — Queue 1 Completion Flag

CF1 indicates that a queue 1 scan has been completed. CF1 is set by the QADC when the conversion is complete for the last CCW in queue 1, and the result is stored in the result table.

0 = Queue 1 scan is not complete.
1 = Queue 1 scan is complete.

PF1 — Queue 1 Pause Flag

PF1 indicates that a queue 1 scan has reached a pause. PF1 is set by the QADC when the current queue 1 CCW has the pause bit set, the selected input channel has been converted, and the result has been stored in the result table.

0 = Queue 1 has not reached a pause.
1 = Queue 1 has reached a pause.

CF2 — Queue 2 Completion Flag

CF2 indicates that a queue 2 scan has been completed. CF2 is set by the QADC when the conversion is complete for the last CCW in queue 2, and the result is stored in the result table.

0 = Queue 2 scan is not complete.
1 = Queue 2 scan is complete.

PF2 — Queue 2 Pause Flag

PF2 indicates that a queue 2 scan has reached a pause. PF2 is set by the QADC when the current queue 2 CCW has the pause bit set, the selected input channel has been converted, and the result has been stored in the result table.

0 = Queue 2 has not reached a pause.
1 = Queue 2 has reached a pause.

TOR1 — Queue 1 Trigger Overrun

TOR1 indicates that an unexpected queue 1 trigger event has occurred. TOR1 can be set only while queue 1 is active.

A trigger event generated by a transition on ETRIG1 may be recorded as a trigger overrun. TOR1 can only be set when using an external trigger mode. TOR1 cannot occur when the software initiated single-scan mode or the software initiated continuous-scan mode is selected.

STOP — Low-Power Stop Mode Enable

0 = QSM clock operates normally.

1 = QSM clock is stopped.

When STOP is set, the QSM enters low-power stop mode. The system clock input to the module is disabled. While STOP is set, only QSMCR reads are guaranteed to be valid, but writes to the QSPI RAM and other QSM registers are guaranteed valid. The SCI receiver and transmitter must be disabled before STOP is set. To stop the QSPI, set the HALT bit in SPCR3, wait until the HALTA flag is set, then set STOP.

FRZ1— FREEZE Assertion Response

FRZ1 determines what action is taken by the QSPI when the IMB FREEZE signal is asserted.

0 = Ignore the IMB FREEZE signal.

1 = Halt the QSPI on a transfer boundary.

FRZ0 — Not Implemented

Bits [12:8] — Not Implemented

SUPV — Supervisor/Unrestricted Data Space

The SUPV bit places the QSM registers in either supervisor or user data space.

0 = Registers with access controlled by the SUPV bit are accessible in either supervisor or user mode.

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

Bits [6:4] — Not Implemented

IARB[3:0] — Interrupt Arbitration ID

The IARB field is used to arbitrate between simultaneous interrupt requests of the same priority. Each module that can generate interrupt requests must be assigned a unique, non-zero IARB field value.

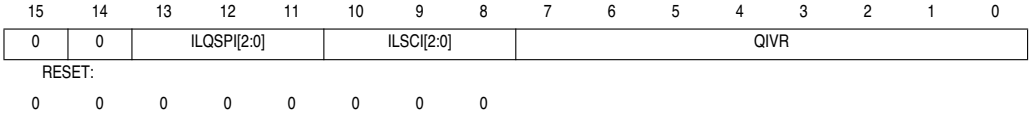
D.6.2 QSM Test Register

QTEST — QSM Test Register **$YFFC02**

Used for factory test only.

D.6.3 QSM Interrupt Level Register

QILR — QSM Interrupt Levels Register **$YFFC04**



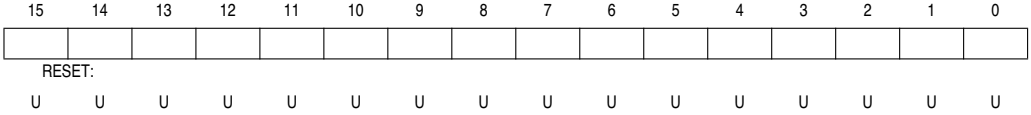
The values of ILQSPI[2:0] and ILSCI[2:0] in QILR determine the priority of QSPI and SCI interrupt requests.

Table D-47 DASMA Operations

Mode	DASMA Operation
DIS	DASMA can be accessed to prepare a value for a subsequent mode selection
IPWM	DASMA contains the captured value corresponding to the trailing edge of the measured pulse
IPM	DASMA contains the captured value corresponding to the most recently detected user-specified rising or falling edge
IC	DASMA contains the captured value corresponding to the most recently detected user-specified rising or falling edge
OCB	DASMA is loaded with the value corresponding to the leading edge of the pulse to be generated. Writing to DASMA in the OCB and OCAB modes also enables the corresponding channel A comparator until the next successful comparison.
OCAB	DASMA is loaded with the value corresponding to the leading edge of the pulse to be generated. Writing to DASMA in the OCB and OCAB modes also enables the corresponding channel A comparator until the next successful comparison.
OPWM	DASMA is loaded with the value corresponding to the leading edge of the PWM pulse to be generated.

D.7.13 DASM Data Register B

DASM3B — DASM3 Data Register B **$YFF41C**
DASM4B — DASM4 Data Register B **$YFF424**
DASM9B — DASM9 Data Register B **$YFF44C**
DASM10B — DASM10 Data Register B **$YFF454**



DASMB is the data register associated with channel B. **Table D-48** shows how DASMB is used with the different modes of operation. Depending on the mode selected, software access is to register B1 or register B2.