



Welcome to [E-XFL.COM](http://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

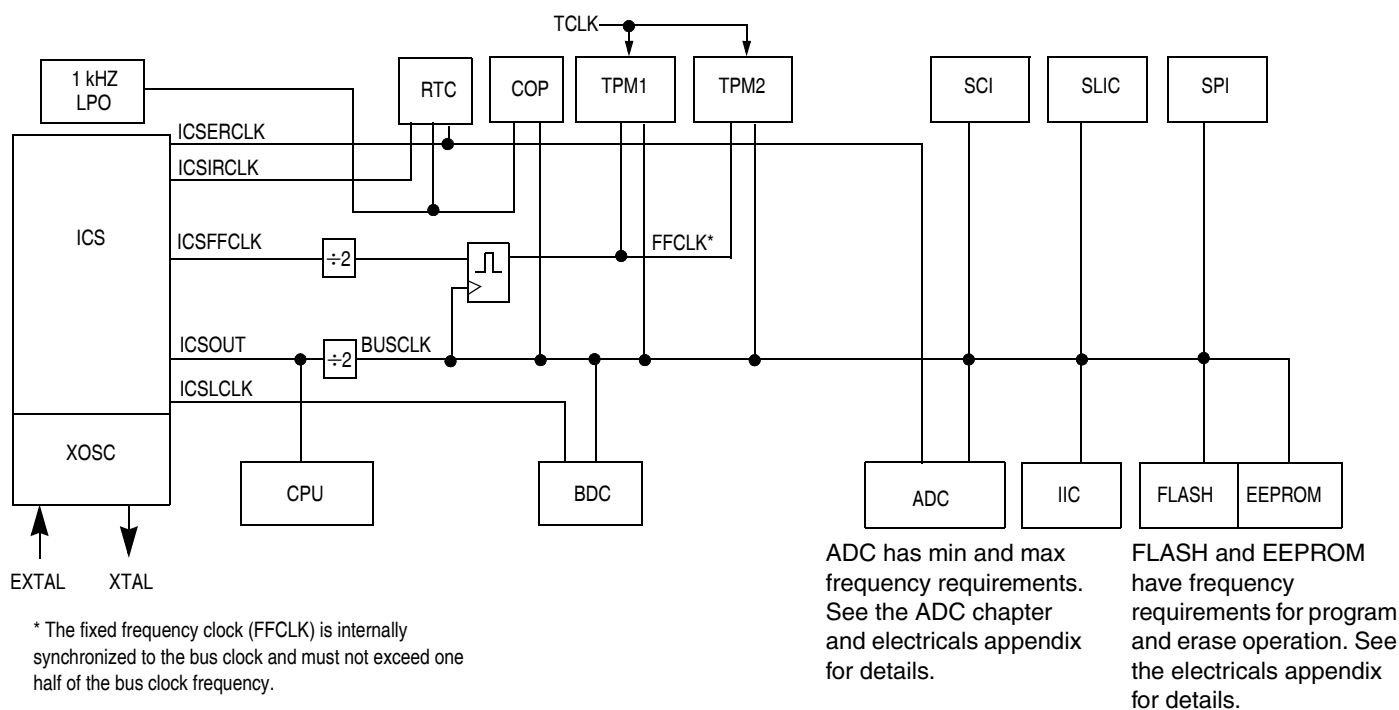
Product Status	Active
Core Processor	S08
Core Size	8-Bit
Speed	40MHz
Connectivity	I <sup>2</sup> C, LINbus, SCI, SPI
Peripherals	LVD, POR, PWM, WDT
Number of I/O	22
Program Memory Size	8KB (8K x 8)
Program Memory Type	FLASH
EEPROM Size	256 x 8
RAM Size	512 x 8
Voltage - Supply (Vcc/Vdd)	2.7V ~ 5.5V
Data Converters	A/D 16x10b
Oscillator Type	External
Operating Temperature	-40°C ~ 85°C (TA)
Mounting Type	Surface Mount
Package / Case	28-TSSOP (0.173", 4.40mm Width)
Supplier Device Package	28-TSSOP
Purchase URL	<a href="https://www.e-xfl.com/pro/item?MUrl=&amp;PartUrl=mc9s08sl8ctl">https://www.e-xfl.com/pro/item?MUrl=&amp;PartUrl=mc9s08sl8ctl</a>

## 1.3 System Clock Distribution

Figure 1-3 shows a simplified clock connection diagram. Some modules in the MCU have selectable clock inputs as shown. The clock inputs to the modules indicate the clock(s) that are used to drive the module function.

The following defines the clocks used in this MCU:

- **BUSCLK** — The frequency of the bus is always half of ICSOUT.
- **ICSOUT** — Primary output of the ICS and is twice the bus frequency.
- **ICSLCLK** — Development tools can select this clock source to speed up BDC communications in systems where the bus clock is configured to run at a very slow frequency.
- **ICSERCLK** — External reference clock can be selected as the RTC clock source and as the alternate clock for the ADC module.
- **ICSIRCLK** — Internal reference clock can be selected as the RTC clock source.
- **ICSFFCLK** — Fixed frequency clock can be selected as clock source for the TPM1 and TPM2 modules.
- **LPO** — Independent 1-kHz clock that can be selected as the source for the COP and RTC modules.
- **TCLK** — External input clock source for TPM1 and TPM2 and is referenced as TPMCLK in TPM chapters.



**Figure 1-3. System Clock Distribution Diagram**



## 4.3 Register Addresses and Bit Assignments

The registers in the MC9S08EL32 Series and MC9S08SL16 Series are divided into these groups:

- Direct-page registers are located in the first 128 locations in the memory map; these are accessible with efficient direct addressing mode instructions.
- High-page registers are used much less often, so they are located above 0x1800 in the memory map. This leaves more room in the direct page for more frequently used registers and RAM.
- The nonvolatile register area consists of a block of 16 locations in FLASH memory at 0xFFB0–0xFFBF. Nonvolatile register locations include:
  - NVPROT and NVOPT which are loaded into working registers at reset
  - An 8-byte backdoor comparison key that optionally allows a user to gain controlled access to secure memory

Because the nonvolatile register locations are FLASH memory, they must be erased and programmed like other FLASH memory locations.

Direct-page registers can be accessed with efficient direct addressing mode instructions. Bit manipulation instructions can be used to access any bit in any direct-page register. [Table 4-2](#) is a summary of all user-accessible direct-page registers and control bits.

The direct page registers in [Table 4-2](#) can use the more efficient direct addressing mode, which requires only the lower byte of the address. Because of this, the lower byte of the address in column one is shown in bold text. In [Table 4-3](#) and [Table 4-4](#), the whole address in column one is shown in bold. In [Table 4-2](#), [Table 4-3](#), and [Table 4-4](#), the register names in column two are shown in bold to set them apart from the bit names to the right. Cells that are not associated with named bits are shaded. A shaded cell with a 0 indicates this unused bit always reads as a 0. Shaded cells with dashes indicate unused or reserved bit locations that could read as 1s or 0s.

### 4.5.5 Sector Erase Abort

The sector erase abort operation will terminate the active sector erase operation so that other sectors are available for read and program operations without waiting for the sector erase operation to complete.

The sector erase abort command write sequence is as follows:

1. Write to any FLASH or EEPROM address to start the command write sequence for the sector erase abort command. The address and data written are ignored.
2. Write the sector erase abort command, 0x47, to the FCMD register.
3. Clear the FCBEF flag in the FSTAT register by writing a “1” to FCBEF to launch the sector erase abort command.

If the sector erase abort command is launched resulting in the early termination of an active sector erase operation, the FACCERR flag will set once the operation completes as indicated by the FCCF flag being set. The FACCERR flag sets to inform the user that the FLASH sector may not be fully erased and a new sector erase command must be launched before programming any location in that specific sector.

If the sector erase abort command is launched but the active sector erase operation completes normally, the FACCERR flag will not set upon completion of the operation as indicated by the FCCF flag being set. Therefore, if the FACCERR flag is not set after the sector erase abort command has completed, a sector being erased when the abort command was launched will be fully erased.

A flowchart to execute the sector erase abort operation is shown in [Figure 4-4](#).

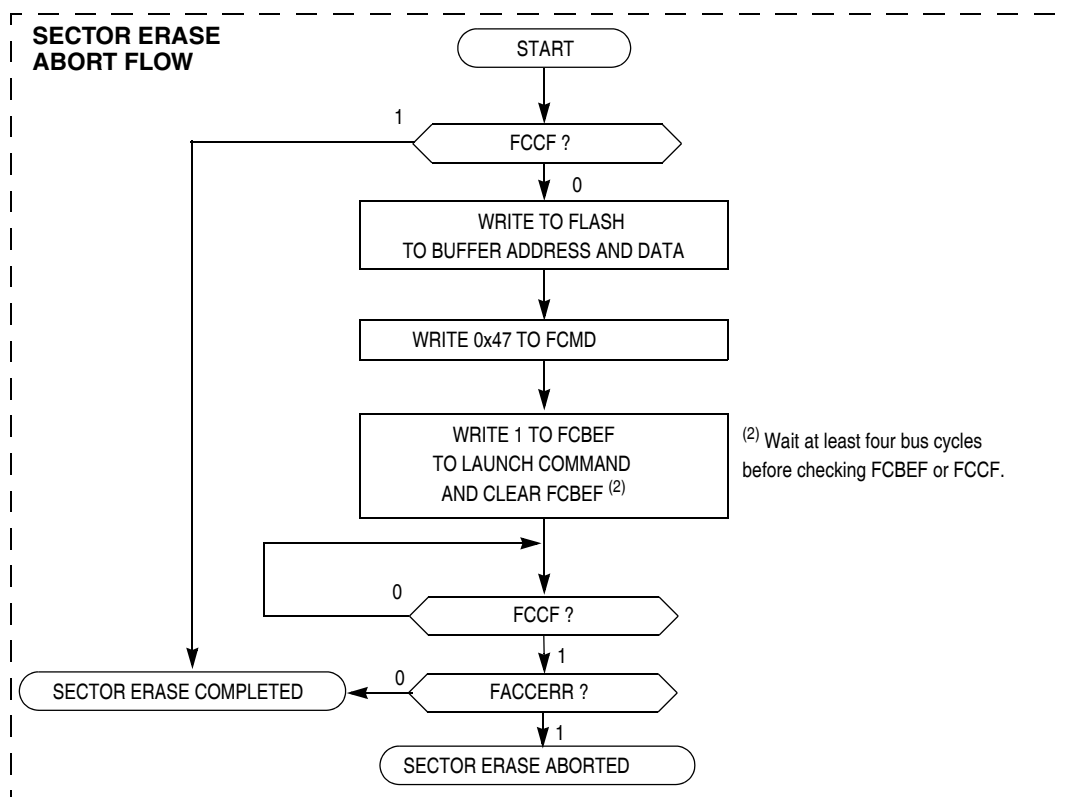


Figure 4-4. Sector Erase Abort Flowchart

The status flag corresponding to the interrupt source must be acknowledged (cleared) before returning from the ISR. Typically, the flag is cleared at the beginning of the ISR so that if another interrupt is generated by this same source, it will be registered so it can be serviced after completion of the current ISR.

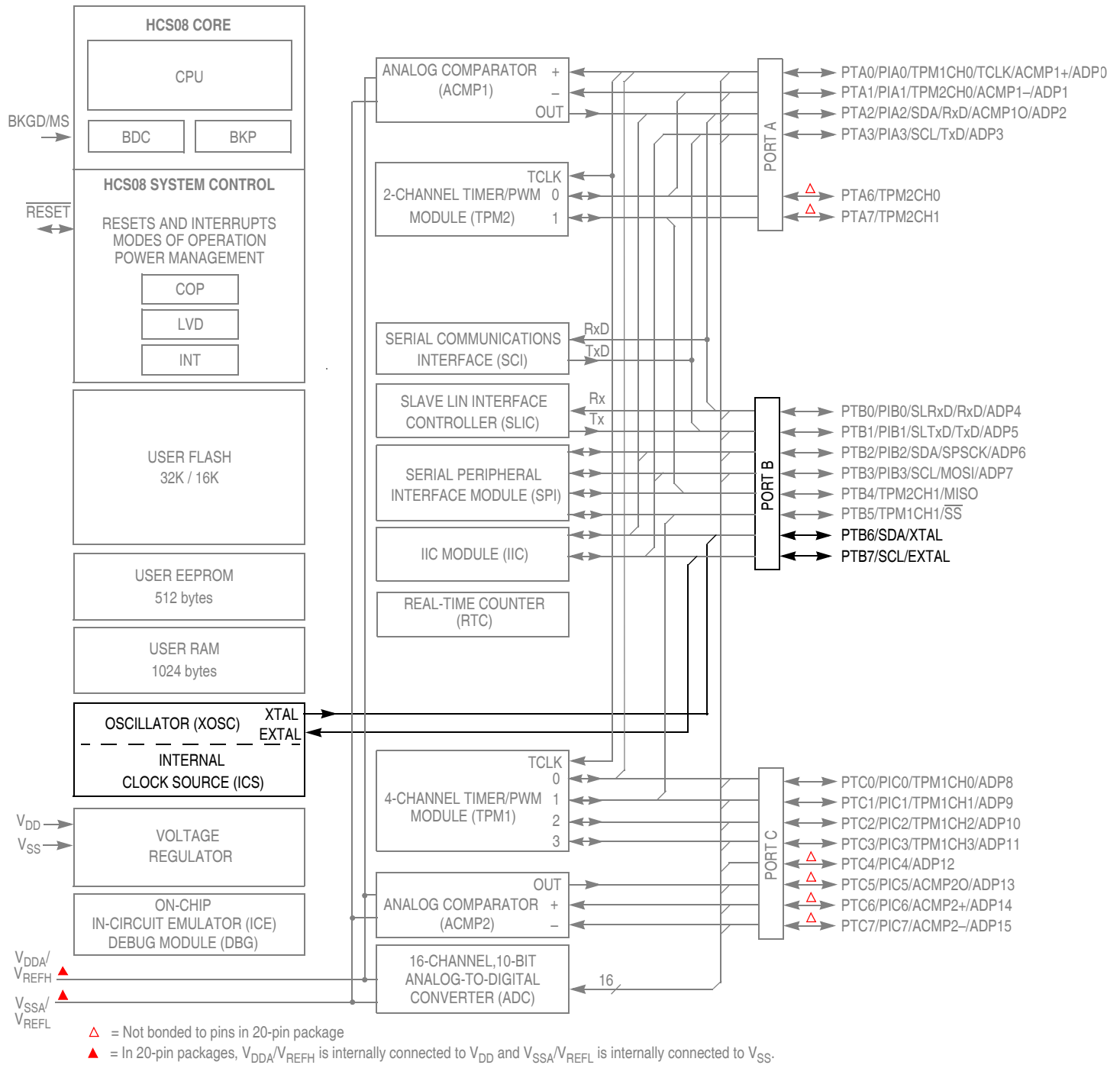
## 5.5.2 Interrupt Vectors, Sources, and Local Masks

Table 5-2 provides a summary of all interrupt sources. Higher-priority sources are located toward the bottom of the table. The high-order byte of the address for the interrupt service routine is located at the first address in the vector address column, and the low-order byte of the address for the interrupt service routine is located at the next higher address.

When an interrupt condition occurs, an associated flag bit becomes set. If the associated local interrupt enable is 1, an interrupt request is sent to the CPU. Within the CPU, if the global interrupt mask (I bit in the CCR) is 0, the CPU will finish the current instruction; stack the PCL, PCH, X, A, and CCR CPU registers; set the I bit; and then fetch the interrupt vector for the highest priority pending interrupt. Processing then continues in the interrupt service routine.

Table 7-2. Instruction Set Summary (Sheet 3 of 9)

Source Form	Operation	Address Mode	Object Code	Cycles	Cyc-by-Cyc Details	Affect on CCR	
						V 1 1 H	I N Z C
BLE <i>rel</i>	Branch if Less Than or Equal To (if $Z \vee (N \oplus V) = 1$ ) (Signed)	REL	93 rr	3	ppp	- 1 1 -	- - - -
BLO <i>rel</i>	Branch if Lower (if $C = 1$ ) (Same as BCS)	REL	25 rr	3	ppp	- 1 1 -	- - - -
BLS <i>rel</i>	Branch if Lower or Same (if $C \vee Z = 1$ )	REL	23 rr	3	ppp	- 1 1 -	- - - -
BLT <i>rel</i>	Branch if Less Than (if $N \oplus V = 1$ ) (Signed)	REL	91 rr	3	ppp	- 1 1 -	- - - -
BMC <i>rel</i>	Branch if Interrupt Mask Clear (if $I = 0$ )	REL	2C rr	3	ppp	- 1 1 -	- - - -
BMI <i>rel</i>	Branch if Minus (if $N = 1$ )	REL	2B rr	3	ppp	- 1 1 -	- - - -
BMS <i>rel</i>	Branch if Interrupt Mask Set (if $I = 1$ )	REL	2D rr	3	ppp	- 1 1 -	- - - -
BNE <i>rel</i>	Branch if Not Equal (if $Z = 0$ )	REL	26 rr	3	ppp	- 1 1 -	- - - -
BPL <i>rel</i>	Branch if Plus (if $N = 0$ )	REL	2A rr	3	ppp	- 1 1 -	- - - -
BRA <i>rel</i>	Branch Always (if $I = 1$ )	REL	20 rr	3	ppp	- 1 1 -	- - - -
BRCLR <i>n,opr8a,rel</i>	Branch if Bit <i>n</i> in Memory Clear (if $Mn = 0$ )	DIR (b0) DIR (b1) DIR (b2) DIR (b3) DIR (b4) DIR (b5) DIR (b6) DIR (b7)	01 dd rr 03 dd rr 05 dd rr 07 dd rr 09 dd rr 0B dd rr 0D dd rr 0F dd rr	5 5 5 5 5 5 5 5	rpppp rpppp rpppp rpppp rpppp rpppp rpppp rpppp	- 1 1 -	- - - - †
BRN <i>rel</i>	Branch Never (if $I = 0$ )	REL	21 rr	3	ppp	- 1 1 -	- - - -
BRSET <i>n,opr8a,rel</i>	Branch if Bit <i>n</i> in Memory Set (if $Mn = 1$ )	DIR (b0) DIR (b1) DIR (b2) DIR (b3) DIR (b4) DIR (b5) DIR (b6) DIR (b7)	00 dd rr 02 dd rr 04 dd rr 06 dd rr 08 dd rr 0A dd rr 0C dd rr 0E dd rr	5 5 5 5 5 5 5 5	rpppp rpppp rpppp rpppp rpppp rpppp rpppp rpppp	- 1 1 -	- - - - †
BSET <i>n,opr8a</i>	Set Bit <i>n</i> in Memory ( $Mn \leftarrow 1$ )	DIR (b0) DIR (b1) DIR (b2) DIR (b3) DIR (b4) DIR (b5) DIR (b6) DIR (b7)	10 dd 12 dd 14 dd 16 dd 18 dd 1A dd 1C dd 1E dd	5 5 5 5 5 5 5 5	rfwpp rfwpp rfwpp rfwpp rfwpp rfwpp rfwpp rfwpp	- 1 1 -	- - - -
BSR <i>rel</i>	Branch to Subroutine PC $\leftarrow$ (PC) + \$0002 push (PCL); SP $\leftarrow$ (SP) - \$0001 push (PCH); SP $\leftarrow$ (SP) - \$0001 PC $\leftarrow$ (PC) + <i>rel</i>	REL	AD rr	5	ssppp	- 1 1 -	- - - -
CBEQ <i>opr8a,rel</i> CBEQA <i>#opr8i,rel</i> CBEQX <i>#opr8i,rel</i> CBEQ <i>opr8,X+,rel</i> CBEQ <i>,X+,rel</i> CBEQ <i>opr8,SP,rel</i>	Compare and... Branch if (A) = (M) Branch if (A) = (M) Branch if (X) = (M) Branch if (A) = (M) Branch if (A) = (M) Branch if (A) = (M)	DIR IMM IMM IX1+ IX+ SP1	31 dd rr 41 ii rr 51 ii rr 61 ff rr 71 rr 9E 61 ff rr	5 4 4 5 5 6	rpppp pppp pppp rpppp rfppp prpppp	- 1 1 -	- - - -



**Figure 8-1. Block Diagram Highlighting ICS Block and Pins**



### 9.1.5 Block Diagram

The block diagram for the Analog Comparator module is shown [Figure 9-2](#).

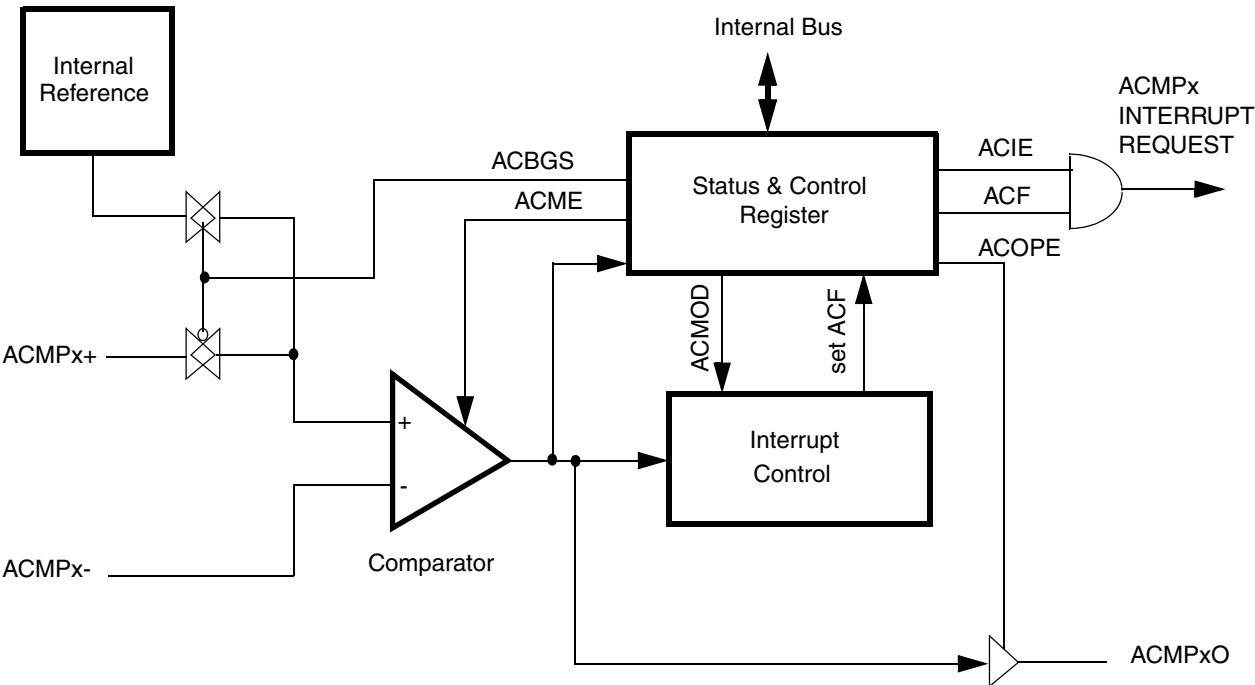


Figure 9-2. Analog Comparator 5V (ACMP5) Block Diagram



### 11.3.3 IIC Control Register (IICC1)

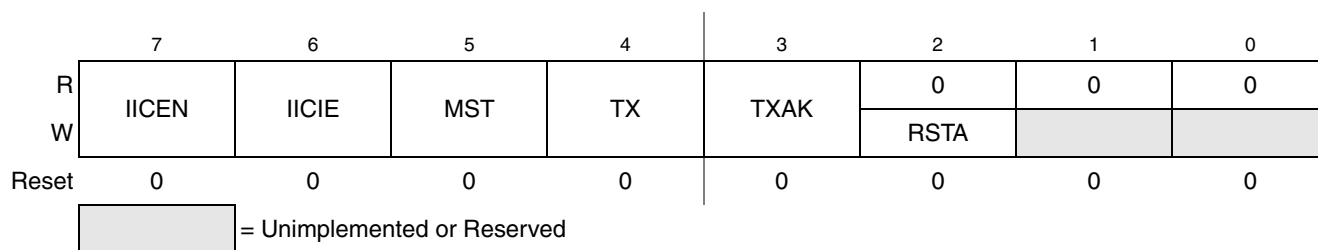


Figure 11-5. IIC Control Register (IICC1)

Table 11-6. IICC1 Field Descriptions

Field	Description
7 IICEN	<b>IIC Enable.</b> The IICEN bit determines whether the IIC module is enabled. 0 IIC is not enabled 1 IIC is enabled
6 IICIE	<b>IIC Interrupt Enable.</b> The IICIE bit determines whether an IIC interrupt is requested. 0 IIC interrupt request not enabled 1 IIC interrupt request enabled
5 MST	<b>Master Mode Select.</b> The MST bit changes from a 0 to a 1 when a start signal is generated on the bus and master mode is selected. When this bit changes from a 1 to a 0 a stop signal is generated and the mode of operation changes from master to slave. 0 Slave mode 1 Master mode
4 TX	<b>Transmit Mode Select.</b> The TX bit selects the direction of master and slave transfers. In master mode, this bit should be set according to the type of transfer required. Therefore, for address cycles, this bit is always high. When addressed as a slave, this bit should be set by software according to the SRW bit in the status register. 0 Receive 1 Transmit
3 TXAK	<b>Transmit Acknowledge Enable.</b> This bit specifies the value driven onto the SDA during data acknowledge cycles for master and slave receivers. 0 An acknowledge signal is sent out to the bus after receiving one data byte 1 No acknowledge signal response is sent
2 RSTA	<b>Repeat start.</b> Writing a 1 to this bit generates a repeated start condition provided it is the current master. This bit is always read as cleared. Attempting a repeat at the wrong time results in loss of arbitration.

### 11.3.4 IIC Status Register (IICS)

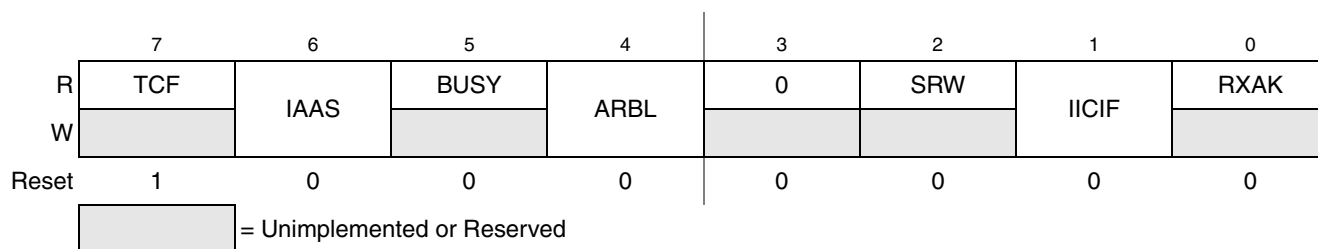


Figure 11-6. IIC Status Register (IICS)

## 12.1.2 Modes of Operation

Figure 12-2 shows the modes in which the SLIC will operate.

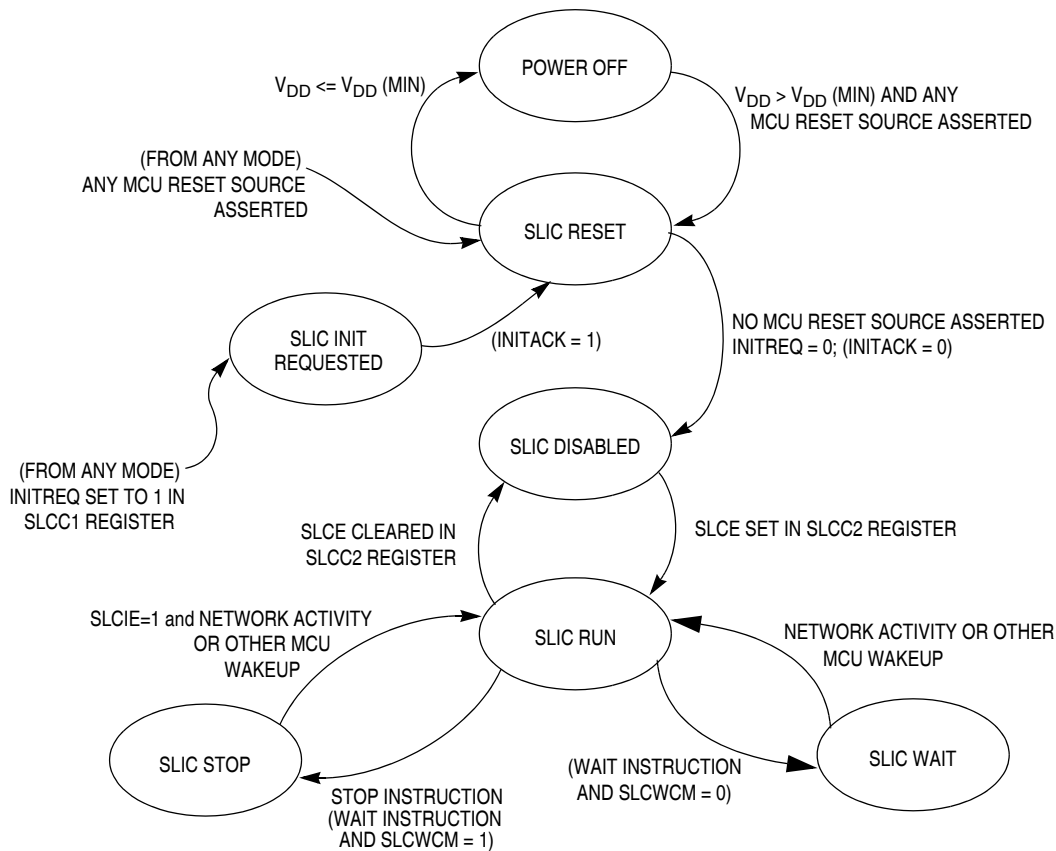


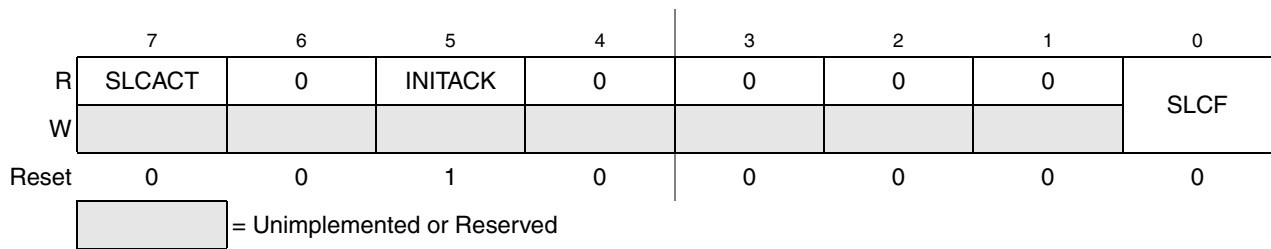
Figure 12-2. SLIC Operating Modes

### 12.1.2.1 Power Off

This mode is entered from the reset mode whenever the SLIC module supply voltage  $V_{DD}$  drops below its minimum specified value for the SLIC module to guarantee operation. The SLIC module will be placed in the reset mode by a system low-voltage reset (LVR) before being powered down. In this mode, the pin input and output specifications are not guaranteed.

### 12.1.2.2 Reset

This mode is entered from the power off mode whenever the SLIC module supply voltage  $V_{DD}$  rises above its minimum specified value ( $V_{DD(MIN)}$ ) and some MCU reset source is asserted. To prevent the SLIC from entering an unknown state, the internal MCU reset is asserted while powering up the SLIC module. SLIC reset mode is also entered from any other mode as soon as one of the MCU's possible reset sources (e.g., LVR, POR, COP,  $\overline{RST}$  pin, etc.) is asserted. SLIC reset mode may also be entered by the user software by asserting the INITREQ bit. INITACK indicates whether the SLIC module is in the reset mode as a result of writing INITREQ in SLCC1. While in the reset state the SLIC module clocks are stopped. Clearing the INITREQ allows the SLIC to proceed and enter SLIC run mode (if SLCE is set). The module



**Figure 12-8. SLIC Status Register (SLCS)**

**Table 12-6. SLCS Field Descriptions**

Field	Description
7 SLCACT <sup>1</sup>	<b>SLIC Active (Oscillator Trim Blocking Semaphore)</b> — SLCACT is used to indicate if it is safe to trim the oscillator based upon current SLIC activity in LIN mode. This bit indicates that the SLIC module might be currently receiving a message header, synchronization byte, ID byte, or sending or receiving data bytes. This bit is read-only. This bit has no meaning in BTM mode (BTM = 1). 0 SLIC module not active (safe to trim oscillator) SLCACT is cleared by the SLIC module only upon assertion of the RX Message Buffer Full Checksum OK (SLCSV = 0x10) or the TX Message Buffer Empty Checksum Transmitted (SLCSV = 0x08) interrupt sources. 1 SLIC module activity (not safe to trim oscillator) SLCACT is automatically set to 1 if a falling edge is seen on the SLCRX pin and has successfully been passed through the digital RX filter. This edge is the potential beginning of a LIN message frame.
5 INITACK	<b>Initialization Mode Acknowledge</b> — INITACK indicates whether the SLIC module is in the reset mode as a result of writing INITREQ in SLCC1. INITACK = 1 causes all SLIC register bits (except SLCWCM: write once) to be held in their reset state and become not writable until INITACK has been cleared. Clear INITACK by clearing INITREQ in SLCC1. After INITACK is cleared, the SLIC module proceeds to SLIC DISABLED mode (see <a href="#">Figure 12-2</a> ) in which the other SLIC register bits are writable and can be configured to the desired SLIC operating mode. INITACK is a read-only bit. 0 Normal operation 1 SLIC module is in reset state
0 SLCF	<b>SLIC Interrupt Flag</b> — The SLCF interrupt flag indicates if a SLIC module interrupt is pending. If set, the SLCVS is then used to determine what interrupt is pending. This flag is cleared by writing a 1 to the bit. If additional interrupt sources are pending, the bit will be automatically set to 1 again by the SLIC. 0 No SLIC interrupt pending 1 SLIC interrupt pending

<sup>1</sup> SLCACT may not be clear during all idle times of the bus. For example, if IMMSG was used to ignore the data interrupts of an extended message frame, SLCACT will remain set until another LIN message is received and either the RX Message Buffer Full Checksum OK (SLCSV = 0x10) or the TX Message Buffer Empty Checksum Transmitted (SLCSV = 0x08) interrupt sources are asserted and cleared. When clear, SLCACT always indicates times when the SLIC module is not active, but it is possible for the SLIC module to be not active with SLCACT set. SLCACT has no meaning in BTM mode.

### 12.3.5 SLIC State Vector Register (SLCSV)

SLIC state vector register (SLCSV) is provided to substantially decrease the CPU overhead associated with servicing interrupts while under operation of a LIN protocol. It provides an index offset that is directly related to the LIN module's current state, which can be used with a user supplied jump table to rapidly enter an interrupt service routine. This eliminates the need for the user to maintain a duplicate state machine in software.

unmasked, after 8 bytes are received or an error is detected. At this interrupt, the SLCSV will indicate an error condition (in case of byte framing error, idle bus) or that the receive buffer is full. If the data is successfully received, the user must then empty the buffer by reading SLCD7-SLCD0 and then subtract 8 from the software byte count. When this software counter reaches 8 or fewer, the remaining data bytes will fit in the buffer and only one interrupt should occur. At this time, the final interrupt may be handled normally, continuing to use the software counter to read the proper number of bytes from the appropriate SLCD registers.

#### NOTE

Do not write SLCDLC more than one time per LIN message frame. The SLIC tracks the number of sent or received bytes based on the value written to this register at the beginning of the data field and rewriting this register will corrupt the checksum calculation and cause unpredictable behavior in the SLIC module. The application software must track the number of sent or received bytes to know what the current byte count in the SLIC is. If programming in C, make sure to use the VOLATILE modifier on this variable (or make it a global variable) to ensure that it keeps its value between interrupts.

### 12.6.8.3 Possible Errors on Command Message Data

Possible errors on command message data are:

- Byte Framing Error
- Checksum-Error (LIN specified error)
- No-Bus-Activity (LIN specified error)
- Receiver Buffer Overrun Error

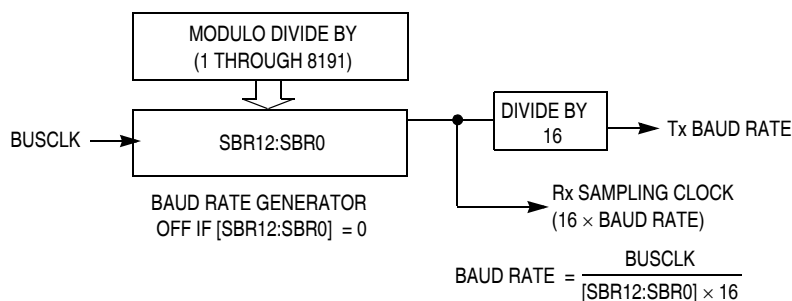
### 12.6.9 Handling Request LIN Message Frames

Figure 12-16 shows how to handle request message frames, where the SLIC module is sending data to the master node.

Request message frames refer to LIN messages frames where the master node is “requesting” the slave node to supply information. The implication is that the slave will then be transmitting data to the master for this message frame. This can be a standard LIN message frame of 1–8 data bytes, a reserved LIN system message (using 0x3D identifier), or an extended request message frame utilizing the reserved 0x3E identifier or perhaps the 0x3F LIN reserved extended identifier. The SLIC module is capable of handling request message frames containing up to 64 bytes of data, while still automatically calculating and/or verifying the checksum.

#### 12.6.9.1 Standard Request Message Frames

Dealing with request messages with the SLIC is very similar to dealing with command messages, with one important difference. Because the SLIC is now to be transmitting data in the LIN message frame, the user software must load the data to be transmitted into the message buffer prior to initiating the transmission.



**Figure 14-12. SCI Baud Rate Generation**

SCI communications require the transmitter and receiver (which typically derive baud rates from independent clock sources) to use the same baud rate. Allowed tolerance on this baud frequency depends on the details of how the receiver synchronizes to the leading edge of the start bit and how bit sampling is performed.

The MCU resynchronizes to bit boundaries on every high-to-low transition, but in the worst case, there are no such transitions in the full 10- or 11-bit time character frame so any mismatch in baud rate is accumulated for the whole character time. For a Freescale Semiconductor SCI system whose bus frequency is driven by a crystal, the allowed baud rate mismatch is about  $\pm 4.5$  percent for 8-bit data format and about  $\pm 4$  percent for 9-bit data format. Although baud rate modulo divider settings do not always produce baud rates that exactly match standard rates, it is normally possible to get within a few percent, which is acceptable for reliable communications.

### 14.3.2 Transmitter Functional Description

This section describes the overall block diagram for the SCI transmitter, as well as specialized functions for sending break and idle characters. The transmitter block diagram is shown in [Figure 14-2](#).

The transmitter output (TxD) idle state defaults to logic high (TXINV = 0 following reset). The transmitter output is inverted by setting TXINV = 1. The transmitter is enabled by setting the TE bit in SCIxC2. This queues a preamble character that is one full character frame of the idle state. The transmitter then remains idle until data is available in the transmit data buffer. Programs store data into the transmit data buffer by writing to the SCI data register (SCIxD).

The central element of the SCI transmitter is the transmit shift register that is either 10 or 11 bits long depending on the setting in the M control bit. For the remainder of this section, we will assume M = 0, selecting the normal 8-bit data mode. In 8-bit data mode, the shift register holds a start bit, eight data bits, and a stop bit. When the transmit shift register is available for a new SCI character, the value waiting in the transmit data register is transferred to the shift register (synchronized with the baud rate clock) and the transmit data register empty (TDRE) status flag is set to indicate another character may be written to the transmit data buffer at SCIxD.

If no new character is waiting in the transmit data buffer after a stop bit is shifted out the TxD pin, the transmitter sets the transmit complete flag and enters an idle mode, with TxD high, waiting for more characters to transmit.





### 15.1.3 Block Diagram

The block diagram for the RTC module is shown in [Figure 15-2](#).

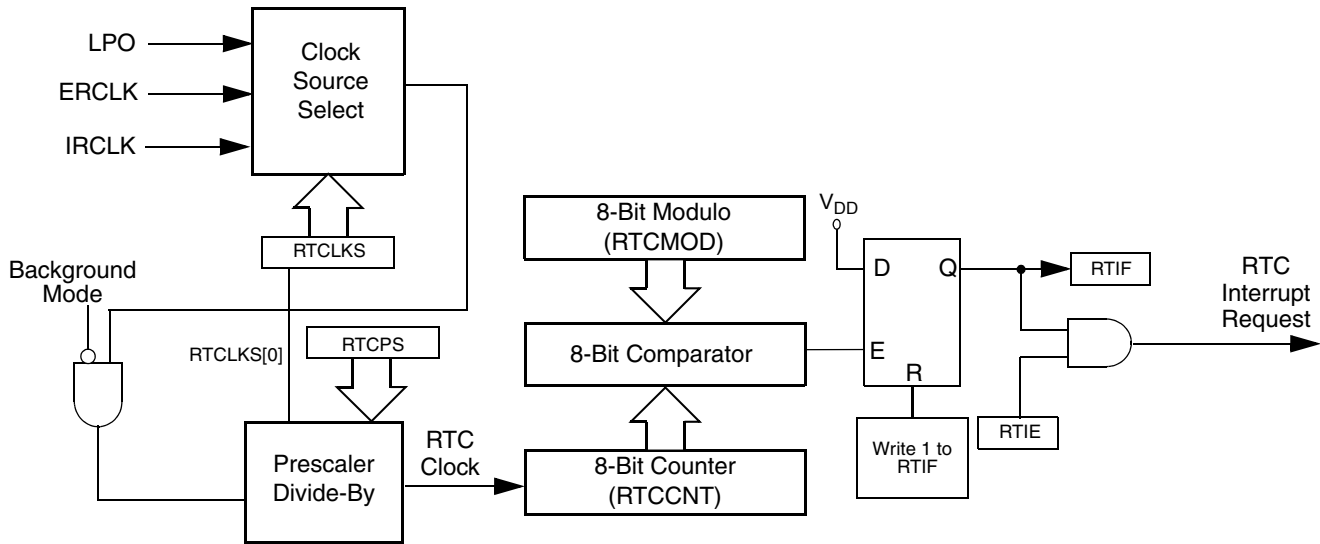


Figure 15-2. Real-Time Counter (RTC) Block Diagram

### 15.2 External Signal Description

The RTC does not include any off-chip signals.

### 15.3 Register Definition

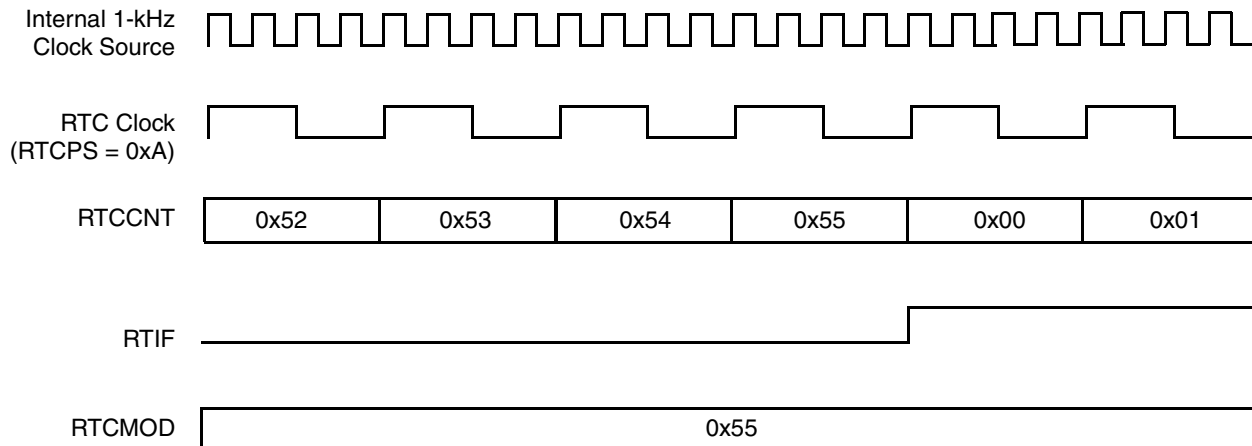
The RTC includes a status and control register, an 8-bit counter register, and an 8-bit modulo register.

Refer to the direct-page register summary in the memory section of this document for the absolute address assignments for all RTC registers. This section refers to registers and control bits only by their names and relative address offsets.

[Table 15-1](#) is a summary of RTC registers.

Table 15-1. RTC Register Summary

Name		7	6	5	4	3	2	1	0
RTCSC	R	RTIF	RTCLKS		RTIE	RTCPS			
	W								
RTCCNT	R	RTCCNT							
	W								
RTCMOD	R	RTCMOD							
	W								



**Figure 15-6. RTC Counter Overflow Example**

In the example of [Figure 15-6](#), the selected clock source is the 1-kHz internal oscillator clock source. The prescaler (RTCPs) is set to 0xA or divide-by-4. The modulo value in the RTCMOD register is set to 0x55. When the counter, RTCCNT, reaches the modulo value of 0x55, the counter overflows to 0x00 and continues counting. The real-time interrupt flag, RTIF, sets when the counter value changes from 0x55 to 0x00. A real-time interrupt is generated when RTIF is set, if RTIE is set.

## 15.5 Initialization/Application Information

This section provides example code to give some basic direction to a user on how to initialize and configure the RTC module. The example software is implemented in C language.

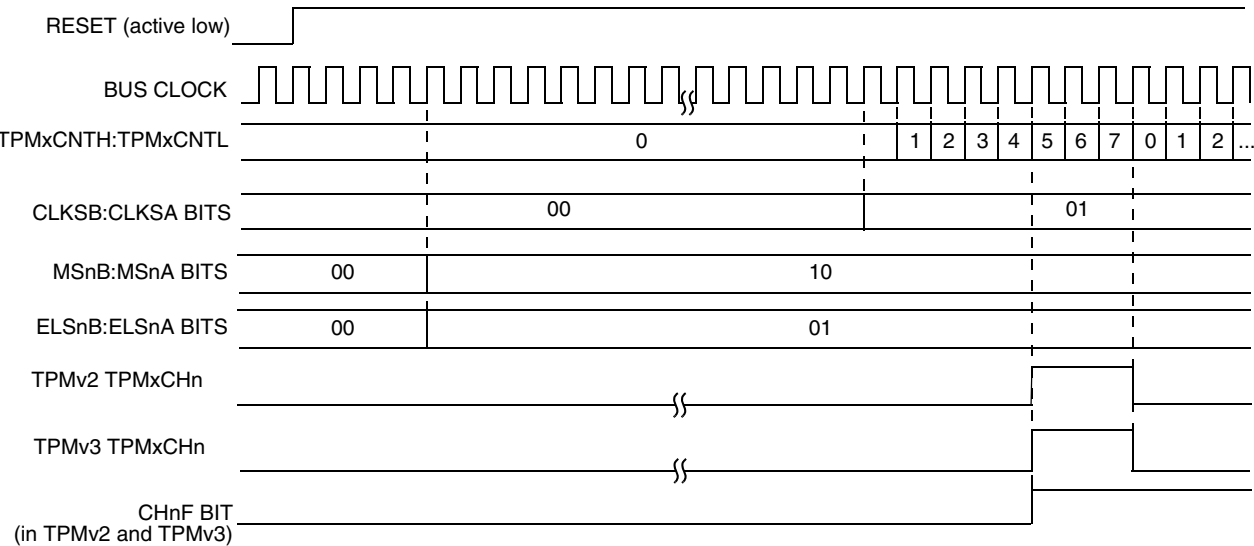
The example below shows how to implement time of day with the RTC using the 1-kHz clock source to achieve the lowest possible power consumption. Because the 1-kHz clock source is not as accurate as a crystal, software can be added for any adjustments. For accuracy without adjustments at the expense of additional power consumption, the external clock (ERCLK) or the internal clock (IRCLK) can be selected with appropriate prescaler and modulo values.

```
/* Initialize the elapsed time counters */
Seconds = 0;
Minutes = 0;
Hours = 0;
Days=0;

/* Configure RTC to interrupt every 1 second from 1-kHz clock source */
RTCMOD.byte = 0x00;
RTCSC.byte = 0x1F;

/*****
Function Name : RTC_ISR
Notes : Interrupt service routine for RTC module.
*****/
```

EPWM mode  
 TPMxMODH:TPMxMODL = 0x0007  
 TPMxMODH:TPMxMODL = 0x0005



**Figure 0-2. Generation of low-true EPWM signal by TPM v2 and v3 after the reset**

The following procedure can be used in TPM v3 (when the channel pin is also a port pin) to emulate the high-true EPWM generated by TPM v2 after the reset.

- ...
- configure the channel pin as output port pin and set the output pin;
- configure the channel to generate the EPWM signal but keep ELSnB:ELSnA as 00;
- configure the other registers (TPMxMODH, TPMxMODL, TPMxCnVH, TPMxCnVL, ...);
- configure CLKSb:CLKSA bits (TPM v3 starts to generate the high-true EPWM signal, however TPM does not control the channel pin, so the EPWM signal is not available);
- wait until the TOF is set (or use the TOF interrupt);
- enable the channel output by configuring ELSnB:ELSnA bits (now EPWM signal is available);
- ...

## 17.1.2 Features

Features of the BDC module include:

- Single pin for mode selection and background communications
- BDC registers are not located in the memory map
- SYNC command to determine target communications rate
- Non-intrusive commands for memory access
- Active background mode commands for CPU register access
- GO and TRACE1 commands
- BACKGROUND command can wake CPU from stop or wait modes
- One hardware address breakpoint built into BDC
- Oscillator runs in stop mode, if BDC enabled
- COP watchdog disabled while in active background mode

Features of the ICE system include:

- Two trigger comparators: Two address + read/write (R/W) or one full address + data + R/W
- Flexible 8-word by 16-bit FIFO (first-in, first-out) buffer for capture information:
  - Change-of-flow addresses or
  - Event-only data
- Two types of breakpoints:
  - Tag breakpoints for instruction opcodes
  - Force breakpoints for any address access
- Nine trigger modes:
  - Basic: A-only, A OR B
  - Sequence: A then B
  - Full: A AND B data, A AND NOT B data
  - Event (store data): Event-only B, A then event-only B
  - Range: Inside range ( $A \leq \text{address} \leq B$ ), outside range ( $\text{address} < A$  or  $\text{address} > B$ )

## 17.2 Background Debug Controller (BDC)

All MCUs in the HCS08 Family contain a single-wire background debug interface that supports in-circuit programming of on-chip nonvolatile memory and sophisticated non-intrusive debug capabilities. Unlike debug interfaces on earlier 8-bit MCUs, this system does not interfere with normal application resources. It does not use any user memory or locations in the memory map and does not share any on-chip peripherals.

BDC commands are divided into two groups:

- Active background mode commands require that the target MCU is in active background mode (the user program is not running). Active background mode commands allow the CPU registers to be read or written, and allow the user to trace one user instruction at a time, or GO to the user program from active background mode.

A force-type breakpoint waits for the current instruction to finish and then acts upon the breakpoint request. The usual action in response to a breakpoint is to go to active background mode rather than continuing to the next instruction in the user application program.

The tag vs. force terminology is used in two contexts within the debug module. The first context refers to breakpoint requests from the debug module to the CPU. The second refers to match signals from the comparators to the debugger control logic. When a tag-type break request is sent to the CPU, a signal is entered into the instruction queue along with the opcode so that if/when this opcode ever executes, the CPU will effectively replace the tagged opcode with a BGND opcode so the CPU goes to active background mode rather than executing the tagged instruction. When the TRGSEL control bit in the DBGTR register is set to select tag-type operation, the output from comparator A or B is qualified by a block of logic in the debug module that tracks opcodes and only produces a trigger to the debugger if the opcode at the compare address is actually executed. There is separate opcode tracking logic for each comparator so more than one compare event can be tracked through the instruction queue at a time.

### 17.3.5 Trigger Modes

The trigger mode controls the overall behavior of a debug run. The 4-bit TRG field in the DBGTR register selects one of nine trigger modes. When TRGSEL = 1 in the DBGTR register, the output of the comparator must propagate through an opcode tracking circuit before triggering FIFO actions. The BEGIN bit in DBGTR chooses whether the FIFO begins storing data when the qualified trigger is detected (begin trace), or the FIFO stores data in a circular fashion from the time it is armed until the qualified trigger is detected (end trigger).

A debug run is started by writing a 1 to the ARM bit in the DBGCR register, which sets the ARMF flag and clears the AF and BF flags and the CNT bits in DBGSR. A begin-trace debug run ends when the FIFO gets full. An end-trace run ends when the selected trigger event occurs. Any debug run can be stopped manually by writing a 0 to ARM or DBGEN in DBGCR.

In all trigger modes except event-only modes, the FIFO stores change-of-flow addresses. In event-only trigger modes, the FIFO stores data in the low-order eight bits of the FIFO.

The BEGIN control bit is ignored in event-only trigger modes and all such debug runs are begin type traces. When TRGSEL = 1 to select opcode fetch triggers, it is not necessary to use R/W in comparisons because opcode tags would only apply to opcode fetches that are always read cycles. It would also be unusual to specify TRGSEL = 1 while using a full mode trigger because the opcode value is normally known at a particular address.

The following trigger mode descriptions only state the primary comparator conditions that lead to a trigger. Either comparator can usually be further qualified with R/W by setting RWAEN (RWBEN) and the corresponding RWA (RWB) value to be matched against R/W. The signal from the comparator with optional R/W qualification is used to request a CPU breakpoint if BRKEN = 1 and TAG determines whether the CPU request will be a tag request or a force request.