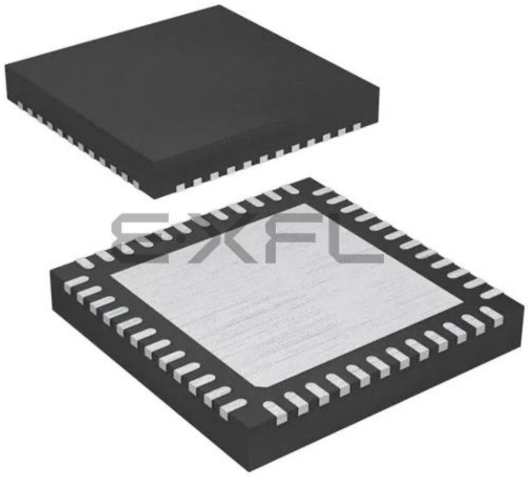
NXP USA Inc. - MC9S08GT60CFDER Datasheet



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	S08
Core Size	8-Bit
Speed	40MHz
Connectivity	I²C, SCI, SPI
Peripherals	LVD, POR, PWM, WDT
Number of I/O	39
Program Memory Size	60KB (60K x 8)
Program Memory Type	FLASH
EEPROM Size	-
RAM Size	4K x 8
Voltage - Supply (Vcc/Vdd)	1.8V ~ 3.6V
Data Converters	A/D 8x10b
Oscillator Type	Internal
Operating Temperature	-40°C ~ 85°C (TA)
Mounting Type	Surface Mount
Package / Case	48-VFQFN Exposed Pad
Supplier Device Package	48-QFN-EP (7x7)
Purchase URL	https://www.e-xfl.com/product-detail/nxp-semiconductors/mc9s08gt60cfder

Section Number Title Page

Chapter 8
Central Processor Unit (CPU)

Chapter 9
Keyboard Interrupt (KBI) Module

High-page registers, shown in Table 4-3, are accessed much less often than other I/O and control registers so they have been located outside the direct addressable memory space, starting at $1800.

Table 4-3. High-Page Register Summary

Address	Register Name	Bit 7	6	5	4	3	2	1	Bit 0
$1800	SRS	POR	PIN	COP	ILOP	0	ICG	LVD	0
$1801	SBDFR	0	0	0	0	0	0	0	BDFR
$1802	SOPT	COPE	COPT	STOPE	—	0	0	BKGDPE	—
$1803 – $1805	Reserved	— —	— —	— —	— —	— —	— —	— —	— —
$1806	SDIDH	REV3	REV2	REV1	REV0	ID11	ID10	ID9	ID8
$1807	SDIDL	ID7	ID6	ID5	ID4	ID3	ID2	ID1	ID0
$1808	SRTISC	RTIF	RTIACK	RTICLKS	RTIE	0	RTIS2	RTIS1	RTIS0
$1809	SPMSC1	LVDF	LVDACK	LVDIE	LVDRE	LVDSE	LVDE	0	0
$180A	SPMSC2	LVWF	LVWACK	LVDV	LVWV	PPDF	PPDACK	PDC	PPDC
$180B– $180F	Reserved	— —	— —	— —	— —	— —	— —	— —	— —
$1810	DBGCAH	Bit 15	14	13	12	11	10	9	Bit 8
$1811	DBGCAL	Bit 7	6	5	4	3	2	1	Bit 0
$1812	DBGCBH	Bit 15	14	13	12	11	10	9	Bit 8
$1813	DBGCBL	Bit 7	6	5	4	3	2	1	Bit 0
$1814	DBGFH	Bit 15	14	13	12	11	10	9	Bit 8
$1815	DBGFL	Bit 7	6	5	4	3	2	1	Bit 0
$1816	DBGC	DBGEN	ARM	TAG	BRKEN	RWA	RWAEN	RWB	RWBEN
$1817	DBGT	TRGSEL	BEGIN	0	0	TRG3	TRG2	TRG1	TRG0
$1818	DBGS	AF	BF	ARMF	0	CNT3	CNT2	CNT1	CNT0
$1819– $181F	Reserved	— —	— —	— —	— —	— —	— —	— —	— —
$1820	FCDIV	DIVLD	PRDIV8	DIV5	DIV4	DIV3	DIV2	DIV1	DIV0
$1821	FOPT	KEYEN	FNORED	0	0	0	0	SEC01	SEC00
$1822	Reserved	—	—	—	—	—	—	—	—
$1823	FCNFG	0	0	KEYACC	0	0	0	0	0
$1824	FPROT	FPOPEN	FPDIS	FPS2	FPS1	FPS0	0	0	0
$1825	FSTAT	FCBEF	FCCF	FPVIOL	FACCERR	0	FBLANK	0	0
$1826	FCMD	FCMD7	FCMD6	FCMD5	FCMD4	FCMD3	FCMD2	FCMD1	FCMD0
$1827– $182B	Reserved	— —	— —	— —	— —	— —	— —	— —	— —

Nonvolatile FLASH registers, shown in Table 4-4, are located in the FLASH memory. These registers include an 8-byte backdoor key which optionally can be used to gain access to secure memory resources. During reset events, the contents of NVPROT and NVOPT in the nonvolatile register area of the FLASH memory are transferred into corresponding FPROT and FOPT working registers in the high-page registers to control security and block protection options.

4.6.5 FLASH Status Register (FSTAT)

Bits 3, 1, and 0 always read 0 and writes have no meaning or effect. The remaining five bits are status bits that can be read at any time. Writes to these bits have special meanings that are discussed in the bit descriptions.

	Bit 7	6	5	4	3	2	1	Bit 0
Read:	FCBEF	FCCF	FPVIOL	FACCERR	0	FBLANK	0	0
Write:		▨			▨	▨	▨	▨
Reset:	1	1	0	0	0	0	0	0

▨ = Unimplemented or Reserved

Figure 4-8. FLASH Status Register (FSTAT)

FCBEF — FLASH Command Buffer Empty Flag

The FCBEF bit is used to launch commands. It also indicates that the command buffer is empty so that a new command sequence can be executed when performing burst programming. The FCBEF bit is cleared by writing a 1 to it or when a burst program command is transferred to the array for programming. Only burst program commands can be buffered.
1 = A new burst program command may be written to the command buffer.
0 = Command buffer is full (not ready for additional commands).

FCCF — FLASH Command Complete Flag

FCCF is set automatically when the command buffer is empty and no command is being processed. FCCF is cleared automatically when a new command is started (by writing 1 to FCBEF to register a command). Writing to FCCF has no meaning or effect.
1 = All commands complete
0 = Command in progress

FPVIOL — Protection Violation Flag

FPVIOL is set automatically when FCBEF is cleared to register a command that attempts to erase or program a location in a protected block (the erroneous command is ignored). FPVIOL is cleared by writing a 1 to FPVIOL.
1 = An attempt was made to erase or program a protected location.
0 = No protection violation.

FACCERR — Access Error Flag

FACCERR is set automatically when the proper command sequence is not followed exactly (the erroneous command is ignored), if a program or erase operation is attempted before the FCDIV register has been initialized, or if the MCU enters stop while a command was in progress. For a more detailed discussion of the exact actions that are considered access errors, see Section 4.4.5, "Access Errors." FACCERR is cleared by writing a 1 to FACCERR. Writing a 0 to FACCERR has no meaning or effect.
1 = An access error has occurred.
0 = No access error has occurred.

Chapter 5 Resets, Interrupts, and System Configuration

5.1 Introduction

This section discusses basic reset and interrupt mechanisms and the various sources of reset and interrupts in the MC9S08GB/GT. Some interrupt sources from peripheral modules are discussed in greater detail within other sections of this data manual. This section gathers basic information about all reset and interrupt sources in one place for easy reference. A few reset and interrupt sources, including the computer operating properly (COP) watchdog and real-time interrupt (RTI), are not part of on-chip peripheral systems with their own sections but are part of the system control logic.

5.2 Features

Reset and interrupt features include:

- Multiple sources of reset for flexible system configuration and reliable operation:
  - Power-on detection (POR)
  - Low voltage detection (LVD) with enable
  - External $\overline{\text{RESET}}$ pin with enable
  - COP watchdog with enable and two timeout choices
  - Illegal opcode
  - Serial command from a background debug host
- Reset status register (SRS) to indicate source of most recent reset
- Separate interrupt vectors for each module (reduces polling overhead) (see Table 5-1)

5.3 MCU Reset

Resetting the MCU provides a way to start processing from a known set of initial conditions. During reset, most control and status registers are forced to initial values and the program counter is loaded from the reset vector ($FFFE:$FFFF). On-chip peripheral modules are disabled and I/O pins are initially configured as general-purpose high-impedance inputs with pullup devices disabled. The I bit in the condition code register (CCR) is set to block maskable interrupts so the user program has a chance to initialize the stack pointer (SP) and system control settings. SP is forced to $00FF at reset.

The MC9S08GB/GT has seven sources for reset:

- Power-on reset (POR)
- Low-voltage detect (LVD)
- Computer operating properly (COP) timer
- Illegal opcode detect
- Background debug forced reset
- The reset pin ($\overline{\text{RESET}}$)
- Clock generator loss of lock and loss of clock reset

When the CPU receives a qualified interrupt request, it completes the current instruction before responding to the interrupt. The interrupt sequence follows the same cycle-by-cycle sequence as the SWI instruction and consists of:

- Saving the CPU registers on the stack
- Setting the I bit in the CCR to mask further interrupts
- Fetching the interrupt vector for the highest-priority interrupt that is currently pending
- Filling the instruction queue with the first three bytes of program information starting from the address fetched from the interrupt vector locations

While the CPU is responding to the interrupt, the I bit is automatically set to avoid the possibility of another interrupt interrupting the ISR itself (this is called nesting of interrupts). Normally, the I bit is restored to 0 when the CCR is restored from the value stacked on entry to the ISR. In rare cases, the I bit may be cleared inside an ISR (after clearing the status flag that generated the interrupt) so that other interrupts can be serviced without waiting for the first service routine to finish. This practice is not recommended for anyone other than the most experienced programmers because it can lead to subtle program errors that are difficult to debug.

The interrupt service routine ends with a return-from-interrupt (RTI) instruction which restores the CCR, A, X, and PC registers to their pre-interrupt values by reading the previously saved information off the stack.

NOTE

> For compatibility with the M68HC08, the H register is not automatically saved and restored. It is good programming practice to push H onto the stack at the start of the interrupt service routine (ISR) and restore it just before the RTI that is used to return from the ISR.

When two or more interrupts are pending when the I bit is cleared, the highest priority source is serviced first (see Table 5-1).

5.5.1 Interrupt Stack Frame

Figure 5-1 shows the contents and organization of a stack frame. Before the interrupt, the stack pointer (SP) points at the next available byte location on the stack. The current values of CPU registers are stored on the stack starting with the low-order byte of the program counter (PCL) and ending with the CCR. After stacking, the SP points at the next available location on the stack which is the address that is one less than the address where the CCR was saved. The PC value that is stacked is the address of the instruction in the main program that would have executed next if the interrupt had not occurred.

PTBDn — Port B Data Register Bit n (n = 0–7)

For port B pins that are inputs, reads return the logic level on the pin. For port B pins that are configured as outputs, reads return the last value written to this register.

Writes are latched into all bits of this register. For port B pins that are configured as outputs, the logic level is driven out the corresponding MCU pin.

Reset forces PTBD to all 0s, but these 0s are not driven out on the corresponding pins because reset also configures all port pins as high-impedance inputs with pullups disabled.

PTBPEn — Pullup Enable for Port B Bit n (n = 0–7)

For port B pins that are inputs, these read/write control bits determine whether internal pullup devices are enabled. For port B pins that are configured as outputs, these bits are ignored and the internal pullup devices are disabled.
1 = Internal pullup device enabled.
0 = Internal pullup device disabled.

PTBSEn — Slew Rate Control Enable for Port B Bit n (n = 0–7)

For port B pins that are outputs, these read/write control bits determine whether the slew rate controlled outputs are enabled. For port B pins that are configured as inputs, these bits are ignored.
1 = Slew rate control enabled.
0 = Slew rate control disabled.

PTBDDn — Data Direction for Port B Bit n (n = 0–7)

These read/write bits control the direction of port B pins and what is read for PTBD reads.
1 = Output driver enabled for port B bit n and PTBD reads return the contents of PTBDn.
0 = Input (output driver disabled) and reads return the pin value.

6.6.3 Port C Registers (PTCD, PTCPE, PTCSE, and PTCDD)

Port C includes eight general-purpose I/O pins that share with the SCI2 and IIC modules. Port C pins used as general-purpose I/O pins are controlled by the port C data (PTCD), data direction (PTCDD), pullup enable (PTCPE), and slew rate control (PTCSE) registers.

If the SCI2 takes control of a port C pin, the corresponding PTCDD bit is ignored. PTCSE can be used to provide slew rate on the SCI2 transmit pin, TxD2. PTCPE can be used, provided the corresponding PTCDD bit is 0, to provide a pullup device on the SCI2 receive pin, RxD2.

If the IIC takes control of a port C pin, the corresponding PTCDD bit is ignored. PTCSE can be used to provide slew rate on the IIC serial data pin (SDA1), when in output mode and the IIC clock pin (SCL1). PTCPE can be used, provided the corresponding PTCDD bit is 0, to provide a pullup device on the IIC serial data pin, when in receive mode.

Reads of PTCD will return the logic value of the corresponding pin, provided PTCDD is 0.

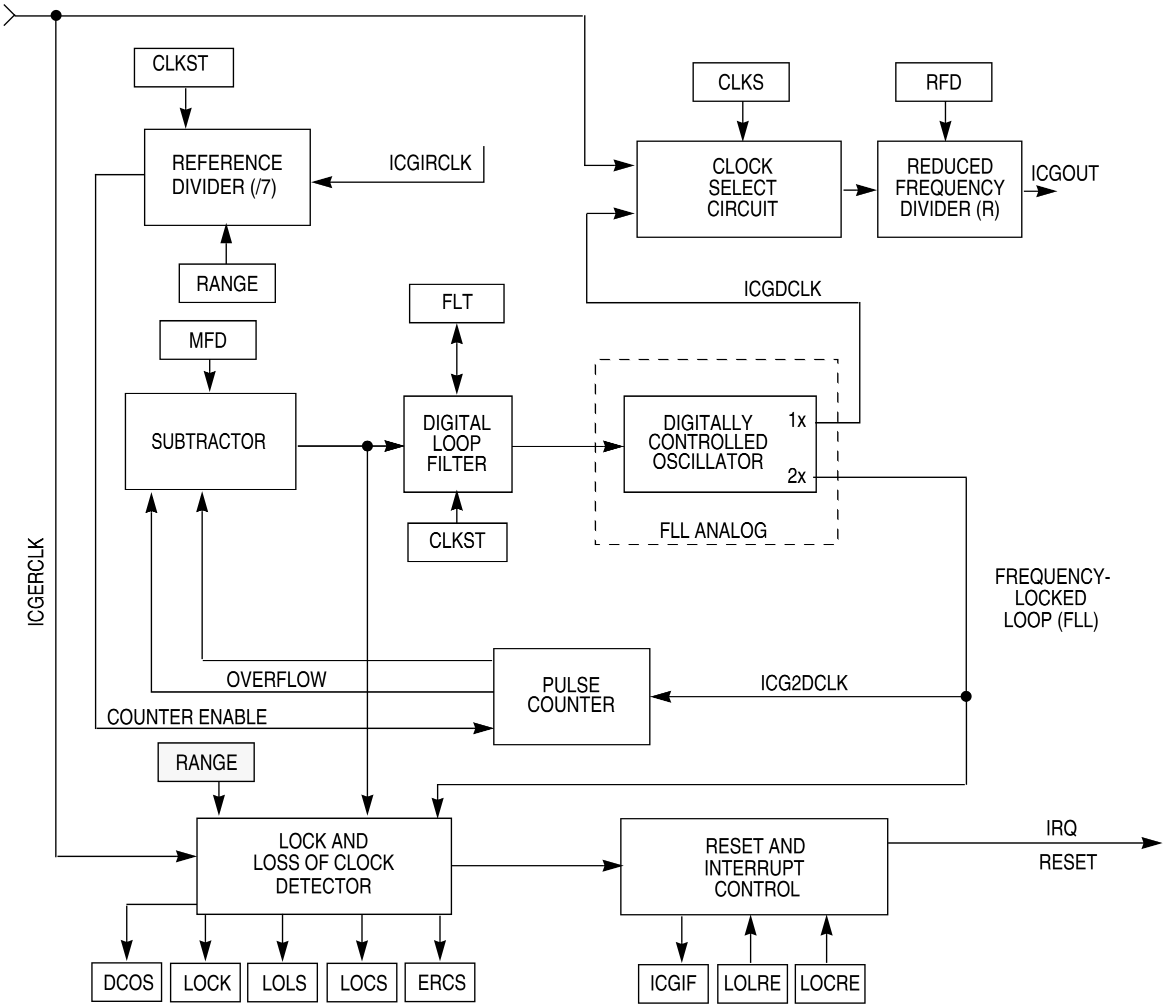


Figure 7-6. Detailed Frequency-Locked Loop Block Diagram

7.3.3 FLL Engaged, Internal Clock (FEI) Mode

FLL engaged internal (FEI) is entered when any of the following conditions occur:

- CLKS bits are written to 01
- The DCO clock stabilizes (DCOS = 1) while in SCM upon exiting the off state with CLKS = 01

In FLL engaged internal mode, the reference clock is derived from the internal reference clock ICGIRCLK, and the FLL loop will attempt to lock the ICGDCLK frequency to the desired value, as selected by the MFD bits.

7.3.3.1 FLL Engaged Internal Unlocked

FEI unlocked is a temporary state that is entered when FEI is entered and the count error (Δn) output from the subtractor is greater than the maximum n_{unlock} or less than the minimum n_{unlock}, as required by the lock detector to detect the unlock condition.

Table 7-5. ICGOUT Frequency Calculation Options

Clock Scheme	f_{ICGOUT}[1]	P	Note
SCM — self-clocked mode (FLL bypassed internal)	$f_{ICGDCLK}$ / R	NA	Typical f_{ICGOUT} = 8 MHz out of reset
FBE — FLL bypassed external	f_{ext} / R	NA	
FEI — FLL engaged internal	(f_{IRG} / 7)* 64*N / R	64	Typical f_{IRG} = 243 kHz

FEE — FLL engaged external	f_{ext} * P * N / R	Range = 0 ; P = 64
		Range = 1; P = 1	

[1] Ensure that $f_{ICGDCLK}$, which is equal to f_{ICGOUT} * R, does not exceed $f_{ICGDCLKmax}$.

Table 7-6. MFD and RFD Decode Table

MFD Value	Multiplication Factor (N)
000	4
001	6
010	8
011	10
100	12
101	14
110	16
111	18

RFD	Division Factor (R)
000	÷1
001	÷2
010	÷4
011	÷8
100	÷16
101	÷32
110	÷64
111	÷128

Register	Bit 7	6	5	4	3	2	1	Bit 0
ICGC1	0	RANGE	REFS	CLKS		OSCSTEN	0[1]	0
ICGC2	LOLRE	MFD			LOCRE	RFD		
ICGS1	CLKST		REFST	LOLS	LOCK	LOCS	ERCS	ICGIF
ICGS2	0	0	0	0	0	0	0	DCOS
ICGFLTU	0	0	0	0	FLT			
ICGFLTL	FLT							
ICGTRM	TRIM							

▨ = Unimplemented or Reserved

[1] This bit is reserved for Freescale Semiconductor internal use only. Any write operations to this register should write a 0 to this bit.

Figure 7-7. ICG Register Set

LOCRE — Loss of Clock Reset Enable

The LOCRE bit determines how the system handles a loss of clock condition.
1 = Generate a reset request on loss of clock.
0 = Generate an interrupt request on loss of clock.

RFD — Reduced Frequency Divider

The RFD bits control the value of the divider following the clock select circuitry. The value specified by the RFD bits establishes the division factor (R) applied to the selected output clock source. Writes to the RFD bits will not take effect if a previous write is not complete.

Table 7-8. RFD Reduced Frequency Divider Select

RFD	Division Factor (R)
000	÷1
001	÷2
010	÷4
011	÷8
100	÷16
101	÷32
110	÷64
111	÷128

7.5.3 ICG Status Register 1 (ICGS1)

	Bit 7	6	5	4	3	2	1	Bit 0
Read:	CLKST		REFST	LOLS	LOCK	LOCS	ERCS	ICGIF
Write:	▨		▨	▨	▨	▨	▨	1
Reset:	0	0	0	0	0	0	0	0

▨ = Unimplemented or Reserved

Figure 7-15. ICG Status Register 1 (ICGS1)

CLKST — Clock Mode Status

The CLKST bits indicate the current clock mode. The CLKST bits don’t update immediately after a write to the CLKS bits due to internal synchronization between clock domains.

Table 8-1. HCS08 Instruction Set Summary (Sheet 3 of 7)

Source Form	Operation	Description	Effect on CCR						Address Mode	Opcode	Operand	Bus Cycles[1]
			V	H	I	N	Z	C				
BMI *rel*	Branch if Minus	Branch if (N) = 1	–	–	–	–	–	–	REL	2B	rr	3
BMS *rel*	Branch if Interrupt Mask Set	Branch if (I) = 1	–	–	–	–	–	–	REL	2D	rr	3
BNE *rel*	Branch if Not Equal	Branch if (Z) = 0	–	–	–	–	–	–	REL	26	rr	3
BPL *rel*	Branch if Plus	Branch if (N) = 0	–	–	–	–	–	–	REL	2A	rr	3
BRA *rel*	Branch Always	No Test	–	–	–	–	–	–	REL	20	rr	3
BRCLR *n*,*opr8a*,*rel*	Branch if Bit *n* in Memory Clear	Branch if (Mn) = 0	–	–	–	–	–		DIR (b0) DIR (b1) DIR (b2) DIR (b3) DIR (b4) DIR (b5) DIR (b6) DIR (b7)	01 03 05 07 09 0B 0D 0F	dd rr dd rr dd rr dd rr dd rr dd rr dd rr dd rr	5 5 5 5 5 5 5 5
BRN *rel*	Branch Never	Uses 3 Bus Cycles	–	–	–	–	–	–	REL	21	rr	3
BRSET *n*,*opr8a*,*rel*	Branch if Bit *n* in Memory Set	Branch if (Mn) = 1	–	–	–	–	–		DIR (b0) DIR (b1) DIR (b2) DIR (b3) DIR (b4) DIR (b5) DIR (b6) DIR (b7)	00 02 04 06 08 0A 0C 0E	dd rr dd rr dd rr dd rr dd rr dd rr dd rr dd rr	5 5 5 5 5 5 5 5
BSET *n*,*opr8a*	Set Bit *n* in Memory	Mn ← 1	–	–	–	–	–	–	DIR (b0) DIR (b1) DIR (b2) DIR (b3) DIR (b4) DIR (b5) DIR (b6) DIR (b7)	10 12 14 16 18 1A 1C 1E	dd dd dd dd dd dd dd dd	5 5 5 5 5 5 5 5
BSR *rel*	Branch to Subroutine	PC ← (PC) + $0002 push (PCL); SP ← (SP) – $0001 push (PCH); SP ← (SP) – $0001 PC ← (PC) + *rel*	–	–	–	–	–	–	REL	AD	rr	5
CBEQ *opr8a*,*rel* CBEQA #*opr8i*,*rel* CBEQX #*opr8i*,*rel* CBEQ *oprx8*,X+,*rel* CBEQ ,X+,*rel* CBEQ *oprx8*,SP,*rel*	Compare and Branch if Equal	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 41 51 61 71 9E61	dd rr ii rr ii rr ff rr rr ff rr	5 4 4 5 5 6
CLC	Clear Carry Bit	C ← 0	–	–	–	–	–	0	INH	98		1
CLI	Clear Interrupt Mask Bit	I ← 0	–	–	0	–	–	–	INH	9A		1
CLR *opr8a* CLRA CLRX CLRH CLR *oprx8*,X CLR ,X CLR *oprx8*,SP	Clear	M ← $00 A ← $00 X ← $00 H ← $00 M ← $00 M ← $00 M ← $00	0	–	–	0	1	–	DIR INH INH INH IX1 IX SP1	3F 4F 5F 8C 6F 7F 9E6F	dd ff ff	5 1 1 1 5 4 6
CMP #*opr8i* CMP *opr8a* CMP *opr16a* CMP *oprx16*,X CMP *oprx8*,X CMP ,X CMP *oprx16*,SP CMP *oprx8*,SP	Compare Accumulator with Memory	(A) – (M) (CCR Updated But Operands Not Changed)		–	–				IMM DIR EXT IX2 IX1 IX SP2 SP1	A1 B1 C1 D1 E1 F1 9ED1 9EE1	ii dd hh ll ee ff ff ee ff ff	2 3 4 4 3 3 5 4

10.3 TPM Block Diagram

The TPM uses one input/output (I/O) pin per channel, TPMxCHn where x is the TPM number (for example, 1 or 2) and n is the channel number (for example, 0–4). The TPM shares its I/O pins with general-purpose I/O port pins (refer to the Pins and Connections chapter for more information). Figure 10-2 shows the structure of a TPM. Some MCUs include more than one TPM, with various numbers of channels.

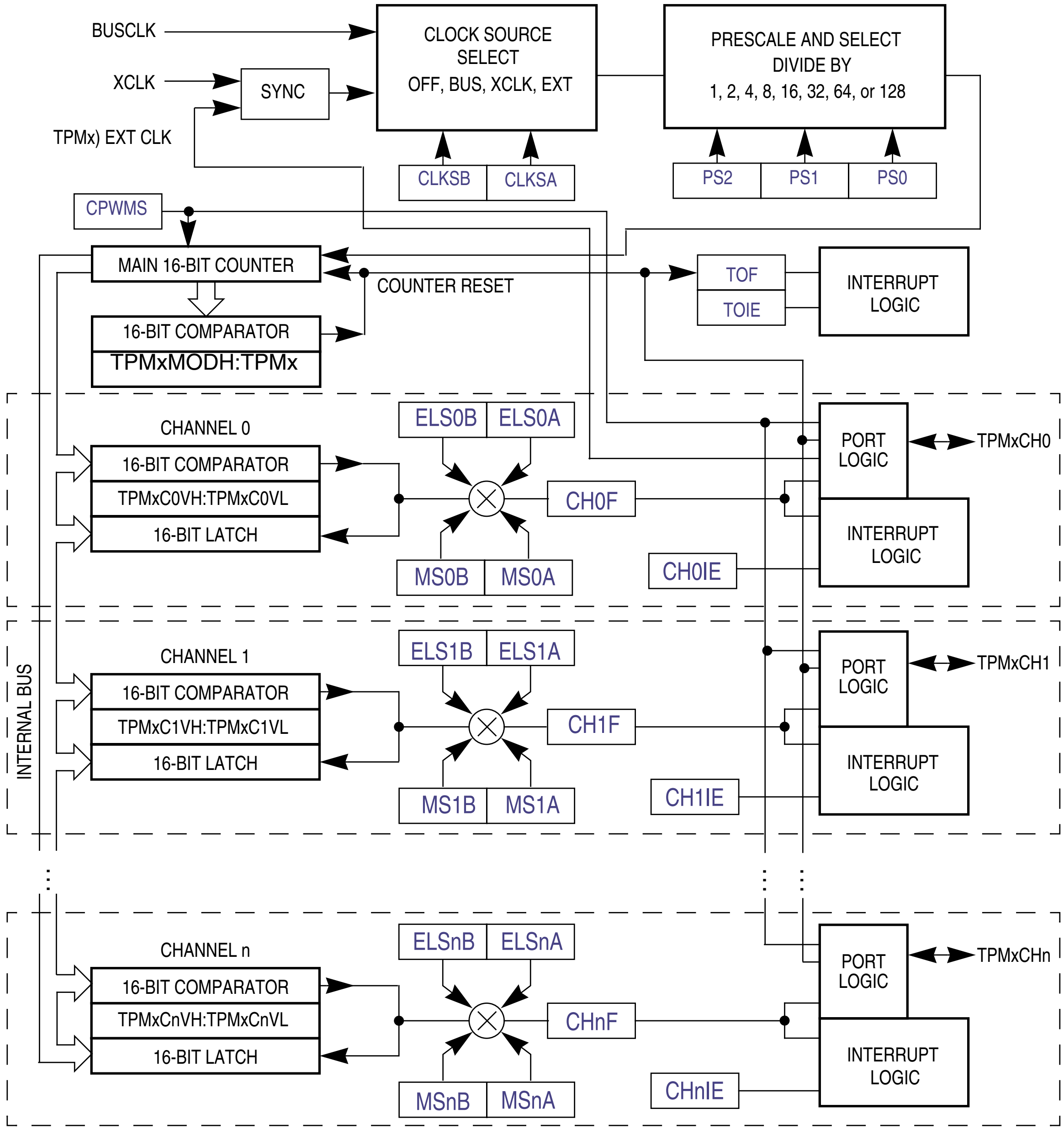


Figure 10-2. TPM Block Diagram

The central component of the TPM is the 16-bit counter that can operate as a free-running counter, a modulo counter, or an up-/down-counter when the TPM is configured for center-aligned PWM. The TPM

Because the HCS08 MCU is an 8-bit architecture, a coherency mechanism is built into the timer counter for read operations. Whenever either byte of the counter is read (TPMxCNTH or TPMxCNTL), both bytes are captured into a buffer so when the other byte is read, the value will represent the other byte of the count at the time the first byte was read. The counter continues to count normally, but no new value can be read from either byte until both bytes of the old count have been read.

The main timer counter can be reset manually at any time by writing any value to either byte of the timer count TPMxCNTH or TPMxCNTL. Resetting the counter in this manner also resets the coherency mechanism in case only one byte of the counter was read before resetting the count.

10.5.2 Channel Mode Selection

Provided CPWMS = 0 (center-aligned PWM operation is not specified), the MSnB and MSnA control bits in the channel n status and control registers determine the basic mode of operation for the corresponding channel. Choices include input capture, output compare, and buffered edge-aligned PWM.

10.5.2.1 Input Capture Mode

With the input capture function, the TPM can capture the time at which an external event occurs. When an active edge occurs on the pin of an input capture channel, the TPM latches the contents of the TPM counter into the channel value registers (TPMxCnVH:TPMxCnVL). Rising edges, falling edges, or any edge may be chosen as the active edge that triggers an input capture.

When either byte of the 16-bit capture register is read, both bytes are latched into a buffer to support coherent 16-bit accesses regardless of order. The coherency sequence can be manually reset by writing to the channel status/control register (TPMxCnSC).

An input capture event sets a flag bit (CHnF) that can optionally generate a CPU interrupt request.

10.5.2.2 Output Compare Mode

With the output compare function, the TPM can generate timed pulses with programmable position, polarity, duration, and frequency. When the counter reaches the value in the channel value registers of an output compare channel, the TPM can set, clear, or toggle the channel pin.

In output compare mode, values are transferred to the corresponding timer channel value registers only after both 8-bit bytes of a 16-bit register have been written. This coherency sequence can be manually reset by writing to the channel status/control register (TPMxCnSC).

An output compare event sets a flag bit (CHnF) that can optionally generate a CPU interrupt request.

10.5.2.3 Edge-Aligned PWM Mode

This type of PWM output uses the normal up-counting mode of the timer counter (CPWMS = 0) and can be used when other channels in the same TPM are configured for input capture or output compare functions. The period of this PWM signal is determined by the setting in the modulus register (TPMxMODH:TPMxMODL). The duty cycle is determined by the setting in the timer channel value

PS2:PS1:PS0 — Prescale Divisor Select

This 3-bit field selects one of eight divisors for the TPM clock input as shown in Table 10-2. This prescaler is located after any clock source synchronization or clock source selection, so it affects whatever clock source is selected to drive the TPM system.

Table 10-2. Prescale Divisor Selection

PS2:PS1:PS0	TPM Clock Source Divided-By
0:0:0	1
0:0:1	2
0:1:0	4
0:1:1	8
1:0:0	16
1:0:1	32
1:1:0	64
1:1:1	128

10.7.2 Timer x Counter Registers (TPMxCNTH:TPMxCNTL)

The two read-only TPM counter registers contain the high and low bytes of the value in the TPM counter. Reading either byte (TPMxCNTH or TPMxCNTL) latches the contents of both bytes into a buffer where they remain latched until the other byte is read. This allows coherent 16-bit reads in either order. The coherency mechanism is automatically restarted by an MCU reset, a write of any value to TPMxCNTH or TPMxCNTL, or any write to the timer status/control register (TPMxSC).

Reset clears the TPM counter registers.

	Bit 7	6	5	4	3	2	1	Bit 0
Read:	Bit 15	14	13	12	11	10	9	Bit 8
Write:	Any write to TPMxCNTH clears the 16-bit counter.							
Reset:	0	0	0	0	0	0	0	0

Figure 10-6. Timer x Counter Register High (TPMxCNTH)

	Bit 7	6	5	4	3	2	1	Bit 0
Read:	Bit 7	6	5	4	3	2	1	Bit 0
Write:	Any write to TPMxCNTL clears the 16-bit counter.							
Reset:	0	0	0	0	0	0	0	0

Figure 10-7. Timer x Counter Register Low (TPMxCNTL)

CHnF — Channel n Flag

When channel n is configured for input capture, this flag bit is set when an active edge occurs on the channel n pin. When channel n is an output compare or edge-aligned PWM channel, CHnF is set when the value in the TPM counter registers matches the value in the TPM channel n value registers. This flag is seldom used with center-aligned PWMs because it is set every time the counter matches the channel value register, which correspond to both edges of the active duty cycle period.

A corresponding interrupt is requested when CHnF is set and interrupts are enabled (CHnIE = 1). Clear CHnF by reading TPMxCnSC while CHnF is set and then writing a 0 to CHnF. If another interrupt request occurs before the clearing sequence is complete, the sequence is reset so CHnF would remain set after the clear sequence was completed for the earlier CHnF. This is done so a CHnF interrupt request cannot be lost by clearing a previous CHnF.

Reset clears the CHnF bit. Writing a 1 to CHnF has no effect.
1 = Input capture or output compare event occurred on channel n.
0 = No input capture or output compare event occurred on channel n.

CHnIE — Channel n Interrupt Enable

This read/write bit enables interrupts from channel n. Reset clears the CHnIE bit.
1 = Channel n interrupt requests enabled.
0 = Channel n interrupt requests disabled (use software polling).

MSnB — Mode Select B for TPM Channel n

When CPWMS = 0, MSnB = 1 configures TPM channel n for edge-aligned PWM mode. For a summary of channel mode and setup controls, refer to Table 10-3.

MSnA — Mode Select A for TPM Channel n

When CPWMS = 0 and MSnB = 0, MSnA configures TPM channel n for input capture mode or output compare mode. Refer to Table 10-3 for a summary of channel mode and setup controls.

	Bit 7	6	5	4	3	2	1	Bit 0
Read: Write:	Bit 15	14	13	12	11	10	9	Bit 8
Reset:	0	0	0	0	0	0	0	0

Figure 10-11. Timer x Channel Value Register High (TPMxCnVH)

	Bit 7	6	5	4	3	2	1	Bit 0
Read: Write:	Bit 7	6	5	4	3	2	1	Bit 0
Reset:	0	0	0	0	0	0	0	0

Figure 10-12. Timer x Channel Value Register Low (TPMxCnVL)

In input capture mode, reading either byte (TPMxCnVH or TPMxCnVL) latches the contents of both bytes into a buffer where they remain latched until the other byte is read. This latching mechanism also resets (becomes unlatched) when the TPMxCnSC register is written.

In output compare or PWM modes, writing to either byte (TPMxCnVH or TPMxCnVL) latches the value into a buffer. When both bytes have been written, they are transferred as a coherent 16-bit value into the timer channel value registers. This latching mechanism may be manually reset by writing to the TPMxCnSC register.

This latching mechanism allows coherent 16-bit writes in either order, which is friendly to various compiler implementations.

Note, because the clocks are halted, the SCI module will resume operation upon exit from stop (only in stop3 mode). Software should ensure stop mode is not entered while there is a character being transmitted out of or received into the SCI module.

11.9.1 Loop Mode

When LOOPS = 1, the RSRC bit in the same register chooses between loop mode (RSRC = 0) or single-wire mode (RSRC = 1). Loop mode is sometimes used to check software, independent of connections in the external system, to help isolate system problems. In this mode, the transmitter output is internally connected to the receiver input and the RxD1 pin is not used by the SCI, so it reverts to a general-purpose port I/O pin.

11.9.2 Single-Wire Operation

When LOOPS = 1, the RSRC bit in the same register chooses between loop mode (RSRC = 0) or single-wire mode (RSRC = 1). Single-wire mode is used to implement a half-duplex serial connection. The receiver is internally connected to the transmitter output and to the TxD1 pin. The RxD1 pin is not used and reverts to a general-purpose port I/O pin.

In single-wire mode, the TXDIR bit in SCIxC3 controls the direction of serial data on the TxD1 pin. When TXDIR = 0, the TxD1 pin is an input to the SCI receiver and the transmitter is temporarily disconnected from the TxD1 pin so an external device can send serial data to the receiver. When TXDIR = 1, the TxD1 pin is an output driven by the transmitter. In single-wire mode, the internal loop back connection from the transmitter to the receiver causes the receiver to receive characters that are sent out by the transmitter.

11.10 SCI Registers and Control Bits

The SCI has eight 8-bit registers to control baud rate, select SCI options, report SCI status, and for transmit/receive data.

Refer to the direct-page register summary in the Memory section of this data sheet for the absolute address assignments for all SCI registers. This section refers to registers and control bits only by their names. A Freescale-provided equate or header file is used to translate these names into the appropriate absolute addresses.

Some MCU systems have more than one SCI, so register names include placeholder characters to identify which SCI is being referenced. For example, SCIxC1 refers to the SCIx control register 1 and SCI2C1 is the control register 1 for SCI2.

11.10.1 SCI x Baud Rate Registers (SCIxBDH, SCIxBDL)

This pair of registers controls the prescale divisor for SCI baud rate generation. To update the 13-bit baud rate setting [SBR12:SBR0], first write to SCIxBDH to buffer the high half of the new value and then write to SCIxBDL. The working value in SCIxBDH does not change until SCIxBDL is written.

SCIxBDL is reset to a non-zero value, so after reset the baud rate generator remains disabled until the first time the receiver or transmitter is enabled (RE or TE bits in SCIxC2 are written to 1).

12.4.5 SPI Data Register (SPI1D)

	Bit 7	6	5	4	3	2	1	Bit 0
Read: Write:	Bit 7	6	5	4	3	2	1	Bit 0
Reset:	0	0	0	0	0	0	0	0

Figure 12-11. SPI Data Register (SPI1D)

Reads of this register return the data read from the receive data buffer. Writes to this register write data to the transmit data buffer. When the SPI is configured as a master, writing data to the transmit data buffer initiates an SPI transfer.

Data should not be written to the transmit data buffer unless the SPI transmit buffer empty flag (SPTEF) is set, indicating there is room in the transmit buffer to queue a new transmit byte.

Data may be read from SPI1D any time after SPRF is set and before another transfer is finished. Failure to read the data out of the receive data buffer before a new transfer ends causes a receive overrun condition and the data from the new transfer is lost.

13.2.1.1 START Signal

When the bus is free; i.e., no master device is engaging the bus (both SCL and SDA lines are at logical high), a master may initiate communication by sending a START signal. As shown in Figure 13-3, a START signal is defined as a high-to-low transition of SDA while SCL is high. This signal denotes the beginning of a new data transfer (each data transfer may contain several bytes of data) and brings all slaves out of their idle states.

13.2.1.2 Slave Address Transmission

The first byte of data transferred immediately after the START signal is the slave address transmitted by the master. This is a seven-bit calling address followed by a R/W bit. The R/W bit tells the slave the desired direction of data transfer.

1 = Read transfer, the slave transmits data to the master.
0 = Write transfer, the master transmits data to the slave.

Only the slave with a calling address that matches the one transmitted by the master will respond by sending back an acknowledge bit. This is done by pulling the SDA low at the 9th clock (see Figure 13-3).

No two slaves in the system may have the same address. If the IIC module is the master, it must not transmit an address that is equal to its own slave address. The IIC cannot be master and slave at the same time. However, if arbitration is lost during an address cycle, the IIC will revert to slave mode and operate correctly even if it is being addressed by another master.

13.2.1.3 Data Transfer

Before successful slave addressing is achieved, the data transfer can proceed byte-by-byte in a direction specified by the R/W bit sent by the calling master.

All transfers that come after an address cycle are referred to as data transfers, even if they carry sub-address information for the slave device

Each data byte is 8 bits long. Data may be changed only while SCL is low and must be held stable while SCL is high as shown in Figure 13-3. There is one clock pulse on SCL for each data bit, the MSB being transferred first. Each data byte is followed by a 9th (acknowledge) bit, which is signalled from the receiving device. An acknowledge is signalled by pulling the SDA low at the ninth clock. In summary, one complete data transfer needs nine clock pulses.

If the slave receiver does not acknowledge the master in the 9th bit time, the SDA line must be left high by the slave. The master interprets the failed acknowledge as an unsuccessful data transfer.

If the master receiver does not acknowledge the slave transmitter after a data byte transmission, the slave interprets this as an end of data transfer and releases the SDA line.

In either case, the data transfer is aborted and the master does one of two things:

- Relinquishes the bus by generating a STOP signal.
- Commences a new calling by generating a repeated START signal.

Figure 15-2 shows an external host transmitting a logic 1 or 0 to the BKGD pin of a target HCS08 MCU. The host is asynchronous to the target so there is a 0-to-1 cycle delay from the host-generated falling edge to where the target perceives the beginning of the bit time. Ten target BDC clock cycles later, the target senses the bit level on the BKGD pin. Typically, the host actively drives the pseudo-open-drain BKGD pin during host-to-target transmissions to speed up rising edges. Because the target does not drive the BKGD pin during the host-to-target transmission period, there is no need to treat the line as an open-drain signal during this period.

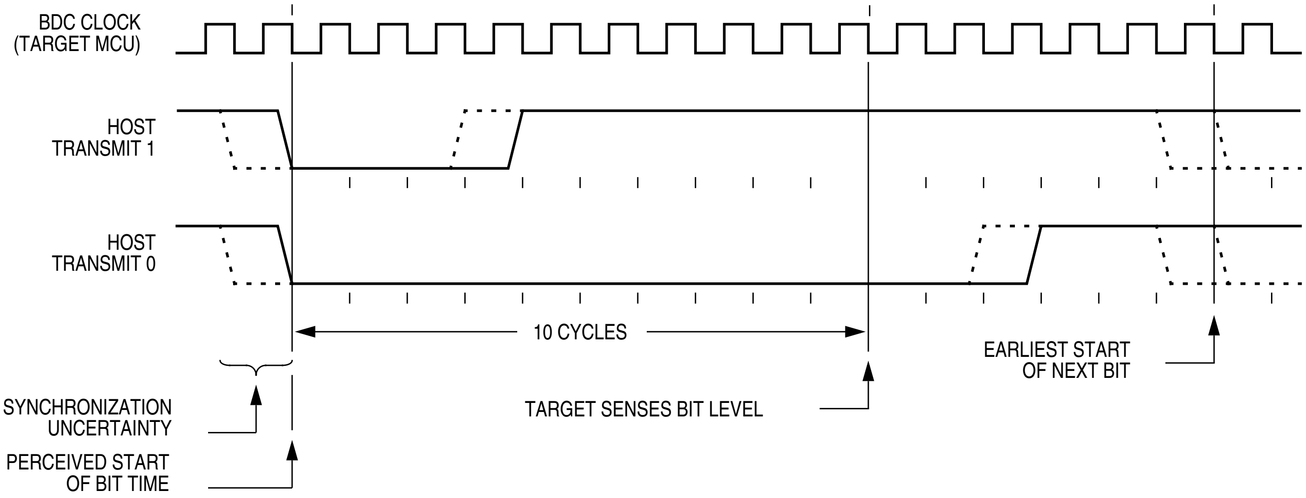


Figure 15-2. BDC Host-to-Target Serial Bit Timing

the host must perform ((8 – CNT) – 1) dummy reads of the FIFO to advance it to the first significant entry in the FIFO.

In most trigger modes, the information stored in the FIFO consists of 16-bit change-of-flow addresses. In these cases, read DBGFH then DBGFL to get one coherent word of information out of the FIFO. Reading DBGFL (the low-order byte of the FIFO data port) causes the FIFO to shift so the next word of information is available at the FIFO data port. In the event-only trigger modes (see Section 15.4.5, “Trigger Modes”), 8-bit data information is stored into the FIFO. In these cases, the high-order half of the FIFO (DBGFH) is not used and data is read out of the FIFO by simply reading DBGFL. Each time DBGFL is read, the FIFO is shifted so the next data value is available through the FIFO data port at DBGFL.

In trigger modes where the FIFO is storing change-of-flow addresses, there is a delay between CPU addresses and the input side of the FIFO. Because of this delay, if the trigger event itself is a change-of-flow address or a change-of-flow address appears during the next two bus cycles after a trigger event starts the FIFO, it will not be saved into the FIFO. In the case of an end-trace, if the trigger event is a change-of-flow, it will be saved as the last change-of-flow entry for that debug run.

The FIFO can also be used to generate a profile of executed instruction addresses when the debugger is not armed. When ARM = 0, reading DBGFL causes the address of the most-recently fetched opcode to be saved in the FIFO. To use the profiling feature, a host debugger would read addresses out of the FIFO by reading DBGFH then DBGFL at regular periodic intervals. The first eight values would be discarded because they correspond to the eight DBGFL reads needed to initially fill the FIFO. Additional periodic reads of DBGFH and DBGFL return delayed information about executed instructions so the host debugger can develop a profile of executed instruction addresses.

15.4.3 Change-of-Flow Information

To minimize the amount of information stored in the FIFO, only information related to instructions that cause a change to the normal sequential execution of instructions is stored. With knowledge of the source and object code program stored in the target system, an external debugger system can reconstruct the path of execution through many instructions from the change-of-flow information stored in the FIFO.

For conditional branch instructions where the branch is taken (branch condition was true), the source address is stored (the address of the conditional branch opcode). Because BRA and BRN instructions are not conditional, these events do not cause change-of-flow information to be stored in the FIFO.

Indirect JMP and JSR instructions use the current contents of the H:X index register pair to determine the destination address, so the debug system stores the run-time destination address for any indirect JMP or JSR. For interrupts, RTI, or RTS, the destination address is stored in the FIFO as change-of-flow information.

15.4.4 Tag vs. Force Breakpoints and Triggers

Tagging is a term that refers to identifying an instruction opcode as it is fetched into the instruction queue, but not taking any other action until and unless that instruction is actually executed by the CPU. This distinction is important because any change-of-flow from a jump, branch, subroutine call, or interrupt causes some instructions that have been fetched into the instruction queue to be thrown away without being executed.