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	S08
Core Size	8-Bit
Speed	8MHz
Connectivity	SCI
Peripherals	LVD, POR, PWM, WDT
Number of I/O	23
Program Memory Size	32KB (32K x 8)
Program Memory Type	FLASH
EEPROM Size	-
RAM Size	2K x 8
Voltage - Supply (Vcc/Vdd)	1.8V ~ 3.6V
Data Converters	-
Oscillator Type	Internal
Operating Temperature	0°C ~ 70°C (TA)
Mounting Type	Surface Mount
Package / Case	28-SOIC (0.295", 7.50mm Width)
Supplier Device Package	28-SOIC
Purchase URL	https://www.e-xfl.com/product-detail/nxp-semiconductors/mc9s08rd32dwer

Chapter 6
Parallel Input/Output

Chapter 7
Central Processor Unit (S08CPUV2)

Chapter 3
Modes of Operation

3.1 Introduction

The operating modes of the MC9S08RC/RD/RE/RG are described in this section. Entry into each mode, exit from each mode, and functionality while in each of the modes are described.

3.2 Features

- Active background mode for code development
- Wait mode:
  - CPU shuts down to conserve power
  - System clocks running
  - Full voltage regulation maintained
- Stop modes:
  - System clocks stopped; voltage regulator in standby
  - Stop1 — Full power down of internal circuits for maximum power savings
  - Stop2 — Partial power down of internal circuits, RAM remains operational
  - Stop3 — All internal circuits powered for fast recovery

3.3 Run Mode

This is the normal operating mode for the MC9S08RC/RD/RE/RG. This mode is selected when the BKGD/MS pin is high at the rising edge of reset. In this mode, the CPU executes code from internal memory with execution beginning at the address fetched from memory at $FFFE:$FFFF after reset.

3.4 Active Background Mode

The active background mode functions are managed through the background debug controller (BDC) in the HCS08 core. The BDC, together with the on-chip debug module (DBG), provide the means for analyzing MCU operation during software development.

Active background mode is entered in any of five ways:

- When the BKGD/MS pin is low at the rising edge of reset
- When a BACKGROUND command is received through the BKGD pin
- When a BGND instruction is executed
- When encountering a BDC breakpoint
- When encountering a DBG breakpoint

After active background mode is entered, the CPU is held in a suspended state waiting for serial background commands rather than executing instructions from the user's application program.

Background commands are of two types:

- Non-intrusive commands, defined as commands that can be issued while the user program is running. Non-intrusive commands can be issued through the BKGD pin while the MCU is in run mode; non-intrusive commands can also be executed when the MCU is in the active background mode. Non-intrusive commands include:
  - Memory access commands
  - Memory-access-with-status commands
  - BDC register access commands
  - BACKGROUND command
- Active background commands, which can only be executed while the MCU is in active background mode, include commands to:
  - Read or write CPU registers
  - Trace one user program instruction at a time
  - Leave active background mode to return to the user's application program (GO)

The active background mode is used to program a bootloader or user application program into the FLASH program memory before the MCU is operated in run mode for the first time. When the MC9S08RC/RD/RE/RG is shipped from the Freescale Semiconductor factory, the FLASH program memory is usually erased so there is no program that could be executed in run mode until the FLASH memory is initially programmed. The active background mode can also be used to erase and reprogram the FLASH memory after it has been previously programmed.

For additional information about the active background mode, refer to the Development Support chapter.

3.5 Wait Mode

Wait mode is entered by executing a WAIT instruction. Upon execution of the WAIT instruction, the CPU enters a low-power state in which it is not clocked. The I bit in CCR is cleared when the CPU enters the wait mode, enabling interrupts. When an interrupt request occurs, the CPU exits the wait mode and resumes processing, beginning with the stacking operations leading to the interrupt service routine.

Only the BACKGROUND command and memory-access-with-status commands are available when the MCU is in wait mode. The memory-access-with-status commands do not allow memory access, but they report an error indicating that the MCU is in either stop or wait mode. The BACKGROUND command can be used to wake the MCU from wait mode and enter active background mode.

into stop2, the states of the I/O pins are latched. The states are held while in stop2 mode and after exiting stop2 mode until a 1 is written to PPDACK in SPMSC2.

Exit from stop2 is done by asserting any of the wakeup pins: $\overline{\text{RESET}}$, IRQ, or KBI1 that have been enabled, or through the real-time interrupt. IRQ and KBI1 pins are always active-low when used as wakeup pins in stop2 regardless of how they were configured before entering stop2. (KBI2 will not wake the MCU from stop2.)

Upon wakeup from stop2 mode, the MCU will start up as from a power-on reset (POR) except pin states remain latched. The CPU will take the reset vector. The system and all peripherals will be in their default reset states and must be initialized.

After waking up from stop2, the PPDF bit in SPMSC2 is set. This flag may be used to direct user code to go to a stop2 recovery routine. PPDF remains set and the I/O pin states remain latched until a 1 is written to PPDACK in SPMSC2.

For pins that were configured as general-purpose I/O, the user must copy the contents of the I/O port registers, which have been saved in RAM, back to the port registers before writing to the PPDACK bit. If the port registers are not restored from RAM before writing to PPDACK, then the register bits will be in their reset states when the I/O pin latches are opened and the I/O pins will switch to their reset states.

For pins that were configured as peripheral I/O, the user must reconfigure the peripheral module that interfaces to the pin before writing to the PPDACK bit. If the peripheral module is not enabled before writing to PPDACK, the pins will be controlled by their associated port control registers when the I/O latches are opened.

3.6.3 Stop3 Mode

Upon entering stop3 mode, all of the clocks in the MCU, including the oscillator itself, are halted. The OSC is turned off, the ACMP is disabled, and the voltage regulator is put in standby. The states of all of the internal registers and logic, as well as the RAM content, are maintained. The I/O pin states are not latched at the pin as in stop2. Instead they are maintained by virtue of the states of the internal logic driving the pins being maintained.

Exit from stop3 is done by asserting $\overline{\text{RESET}}$, any asynchronous interrupt pin that has been enabled, or through the real-time interrupt. The asynchronous interrupt pins are the IRQ or KBI1 and KBI2 pins.

If stop3 is exited by means of the $\overline{\text{RESET}}$ pin, then the MCU will be reset and operation will resume after taking the reset vector. Exit by means of an asynchronous interrupt or the real-time interrupt will result in the MCU taking the appropriate interrupt vector.

A separate self-clocked source (≈1 kHz) for the real-time interrupt allows a wakeup from stop2 or stop3 mode with no external components. When RTIS2:RTIS1:RTIS0 = 0:0:0, the real-time interrupt function and this 1-kHz source are disabled. Power consumption is lower when the 1-kHz source is disabled, but in that case the real-time interrupt cannot wake the MCU from stop.

4.4.1 Features

Features of the FLASH memory include:

- FLASH Size
  - MC9S08RC/RD/RE/RG60 — 63374 bytes (124 pages of 512 bytes each)
  - MC9S08RC/RD/RE/RG32 — 32768 bytes (64 pages of 512 bytes each)
  - MC9S08RC/RD/RE16 — 16384 bytes (32 pages of 512 bytes each)
  - MC9S08RC/RD/RE8 — 8192 bytes (16 pages of 512 bytes each)
- Single power supply program and erase
- Command interface for fast program and erase operation
- Up to 100,000 program/erase cycles at typical voltage and temperature
- Flexible block protection
- Security feature for FLASH and RAM
- Auto power-down for low-frequency read accesses

4.4.2 Program and Erase Times

Before any program or erase command can be accepted, the FLASH clock divider register (FCDIV) must be written to set the internal clock for the FLASH module to a frequency (f_{FCLK}) between 150 kHz and 200 kHz (see Section 4.6.1, “FLASH Clock Divider Register (FCDIV)”). This register can be written only once, so normally this write is done during reset initialization. FCDIV cannot be written if the access error flag, FACCERR in FSTAT, is set. The user must ensure that FACCERR is not set before writing to the FCDIV register. One period of the resulting clock ($1/f_{FCLK}$) is used by the command processor to time program and erase pulses. An integer number of these timing pulses are used by the command processor to complete a program or erase command.

Table 4-4 shows program and erase times. The bus clock frequency and FCDIV determine the frequency of FCLK (f_{FCLK}). The time for one cycle of FCLK is $t_{FCLK} = 1/f_{FCLK}$. The times are shown as a number of cycles of FCLK and as an absolute time for the case where $t_{FCLK} = 5$ µs. Program and erase times shown include overhead for the command state machine and enabling and disabling of program and erase voltages.

Table 4-4. Program and Erase Times

Parameter	Cycles of FCLK	Time if FCLK = 200 kHz
Byte program	9	45 µs
Byte program (burst)	4	20 µs(1)
Page erase	4000	20 ms
Mass erase	20,000	100 ms

1. Excluding start/end overhead

4.4.3 Program and Erase Command Execution

The steps for executing any of the commands are listed below. The FCDIV register must be initialized and any error flags cleared before beginning command execution. The command execution steps are:

1. Write a data value to an address in the FLASH array. The address and data information from this write is latched into the FLASH interface. This write is a required first step in any command sequence. For erase and blank check commands, the value of the data is not important. For page erase commands, the address may be any address in the 512-byte page of FLASH to be erased. For mass erase and blank check commands, the address can be any address in the FLASH memory. Whole pages of 512 bytes are the smallest block of FLASH that may be erased. In the 60K version, there are two instances where the size of a block that is accessible to the user is less than 512 bytes: the first page following RAM, and the first page following the high page registers. These pages are overlapped by the RAM and high page registers respectively.

   NOTE

   Do not program any byte in the FLASH more than once after a successful erase operation. Reprogramming bits to a byte which is already programmed is not allowed without first erasing the page in which the byte resides or mass erasing the entire FLASH memory. Programming without first erasing may disturb data stored in the FLASH.

2. Write the command code for the desired command to FCMD. The five valid commands are blank check ($05), byte program ($20), burst program ($25), page erase ($40), and mass erase ($41). The command code is latched into the command buffer.
3. Write a 1 to the FCBEF bit in FSTAT to clear FCBEF and launch the command (including its address and data information).

A partial command sequence can be aborted manually by writing a 0 to FCBEF any time after the write to the memory array and before writing the 1 that clears FCBEF and launches the complete command. Aborting a command in this way sets the FACCERR access error flag, which must be cleared before starting a new command.

A strictly monitored procedure must be adhered to, or the command will not be accepted. This minimizes the possibility of any unintended changes to the FLASH memory contents. The command complete flag (FCCF) indicates when a command is complete. The command sequence must be completed by clearing FCBEF to launch the command. Figure 4-3 is a flowchart for executing all of the commands except for burst programming. The FCDIV register must be initialized before using any FLASH commands. This must be done only once following a reset.

program time provided that the conditions above are met. In the case where the next sequential address is the beginning of a new row, the program time for that byte will be the standard time instead of the burst time. This is because the high voltage to the array must be disabled and then enabled again. If a new burst command has not been queued before the current command completes, then the charge pump will be disabled and high voltage removed from the array.

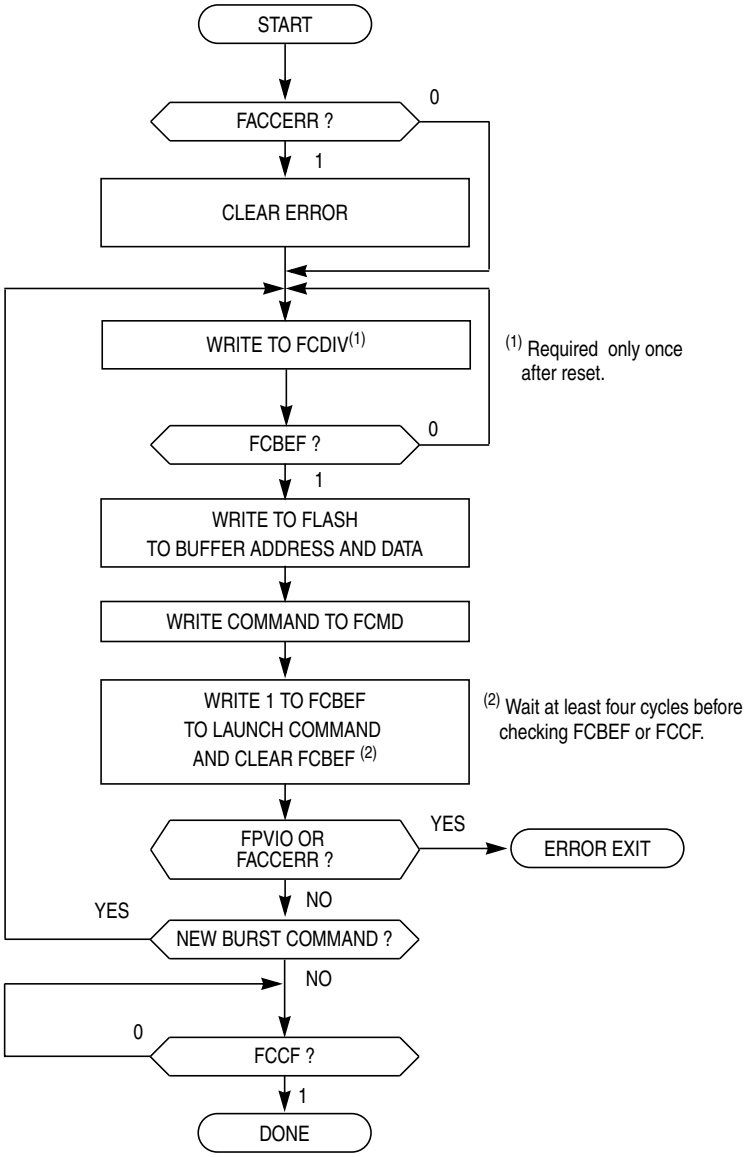


Figure 4-4. FLASH Burst Program Flowchart

4.4.5 Access Errors

An access error occurs whenever the command execution protocol is violated.

Any of the following specific actions will cause the access error flag (FACCERR) in FSTAT to be set. FACCERR must be cleared by writing a 1 to FACCERR in FSTAT before any command can be processed:

5.5.2.2 Edge and Level Sensitivity

The IRQMOD control bit reconfigures the detection logic so it detects edge events and pin levels. In this edge detection mode, the IRQF status flag becomes set when an edge is detected (when the IRQ pin changes from the deasserted to the asserted level), but the flag is continuously set (and cannot be cleared) as long as the IRQ pin remains at the asserted level.

5.5.3 Interrupt Vectors, Sources, and Local Masks

Table 5-1 provides a summary of all interrupt sources. Higher-priority sources are located towards 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.

5.8.4 System Options Register (SOPT)

This register may be read at any time. Bits 3 and 2 are unimplemented and always read 0. This is a write-once register so only the first write after reset is honored. Any subsequent attempt to write to SOPT (intentionally or unintentionally) is ignored to avoid accidental changes to these sensitive settings. SOPT must be written during the user's reset initialization program to set the desired controls even if the desired settings are the same as the reset settings.

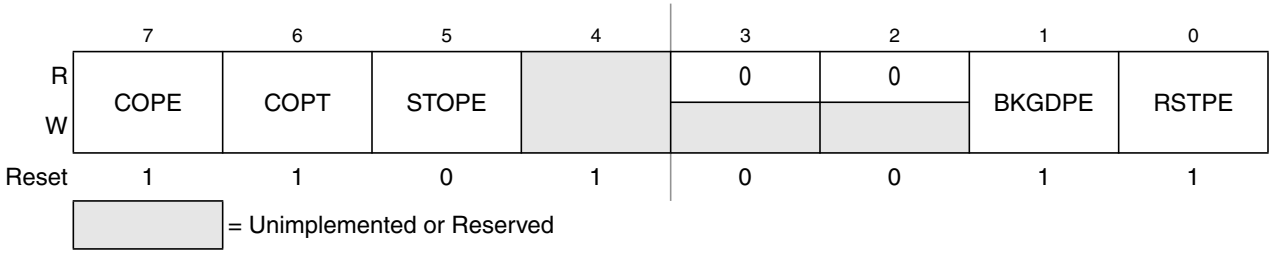


Figure 5-5. System Options Register (SOPT)

Table 5-5. SOPT Field Descriptions

Field	Description
7 COPE	**COP Watchdog Enable** — This write-once bit defaults to 1 after reset. 0 COP watchdog timer disabled. 1 COP watchdog timer enabled (force reset on timeout).
6 COPT	**COP Watchdog Timeout** — This write-once bit defaults to 1 after reset. 0 Short timeout period selected (2^{18} cycles of BUSCLK). 1 Long timeout period selected (2^{20} cycles of BUSCLK).
5 STOPE	**Stop Mode Enable** — This write-once bit defaults to 0 after reset, which disables stop mode. If stop mode is disabled and a user program attempts to execute a STOP instruction, an illegal opcode reset is forced. 0 Stop mode disabled. 1 Stop mode enabled.
1 BKGDPE	**Background Debug Mode Pin Enable** — The BKGDPE bit enables the PTD0/BKGD/MS pin to function as BKGD/MS. When the bit is clear, the pin will function as PTD0, which is an output only general purpose I/O. This pin always defaults to BKGD/MS function after any reset. 0 BKGD pin disabled. 1 BKGD pin enabled.
0 RSTPE	**$\overline{\text{RESET}}$ Pin Enable** — The RSTPE bit enables the PTD1/$\overline{\text{RESET}}$ pin to function as $\overline{\text{RESET}}$. When the bit is clear, the pin will function as PTD1, which is an output only general purpose I/O. This pin always defaults to $\overline{\text{RESET}}$ function after any reset. 0 $\overline{\text{RESET}}$ pin disabled. 1 $\overline{\text{RESET}}$ pin enabled.

	7	6	5	4	3	2	1	0
R W	PTADD7	PTADD6	PTADD5	PTADD4	PTADD3	PTADD2	PTADD1	PTADD0
Reset	0	0	0	0	0	0	0	0

Figure 6-8. Data Direction for Port A (PTADD)

Table 6-3. PTADD Field Descriptions

Field	Description
7:0 PTADD[7:0]	**Data Direction for Port A Bits** — These read/write bits control the direction of port A pins and what is read for PTAD reads. 0 Input (output driver disabled) and reads return the pin value. 1 Output driver enabled for port A bit n and PTAD reads return the contents of PTADn.

6.6.2 Port B Registers (PTBD, PTBPE, and PTBDD)

Port B pins used as general-purpose I/O pins are controlled by the port B data (PTBD), data direction (PTBDD), and pullup enable (PTBPE) registers.

	7	6	5	4	3	2	1	0
R W	PTBD7	PTBD6	PTBD5	PTBD4	PTBD3	PTBD2	PTBD1	PTBD0
Reset	0	0	0	0	0	0	0	0

Figure 6-9. Port B Data Register (PTBD)

Table 6-4. PTBD Field Descriptions

Field	Description
7:0 PTBD[7:0]	**Port B Data Register Bits** — 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.

7.2.3 Stack Pointer (SP)

This 16-bit address pointer register points at the next available location on the automatic last-in-first-out (LIFO) stack. The stack may be located anywhere in the 64-Kbyte address space that has RAM and can be any size up to the amount of available RAM. The stack is used to automatically save the return address for subroutine calls, the return address and CPU registers during interrupts, and for local variables. The AIS (add immediate to stack pointer) instruction adds an 8-bit signed immediate value to SP. This is most often used to allocate or deallocate space for local variables on the stack.

SP is forced to 0x00FF at reset for compatibility with the earlier M68HC05 Family. HCS08 programs normally change the value in SP to the address of the last location (highest address) in on-chip RAM during reset initialization to free up direct page RAM (from the end of the on-chip registers to 0x00FF).

The RSP (reset stack pointer) instruction was included for compatibility with the M68HC05 Family and is seldom used in new HCS08 programs because it only affects the low-order half of the stack pointer.

7.2.4 Program Counter (PC)

The program counter is a 16-bit register that contains the address of the next instruction or operand to be fetched.

During normal program execution, the program counter automatically increments to the next sequential memory location every time an instruction or operand is fetched. Jump, branch, interrupt, and return operations load the program counter with an address other than that of the next sequential location. This is called a change-of-flow.

During reset, the program counter is loaded with the reset vector that is located at $FFFE and $FFFF. The vector stored there is the address of the first instruction that will be executed after exiting the reset state.

7.2.5 Condition Code Register (CCR)

The 8-bit condition code register contains the interrupt mask (I) and five flags that indicate the results of the instruction just executed. Bits 6 and 5 are set permanently to 1. The following paragraphs describe the functions of the condition code bits in general terms. For a more detailed explanation of how each instruction sets the CCR bits, refer to the *HCS08 Family Reference Manual, volume 1,* Freescale Semiconductor document order number HCS08RMv1/D.

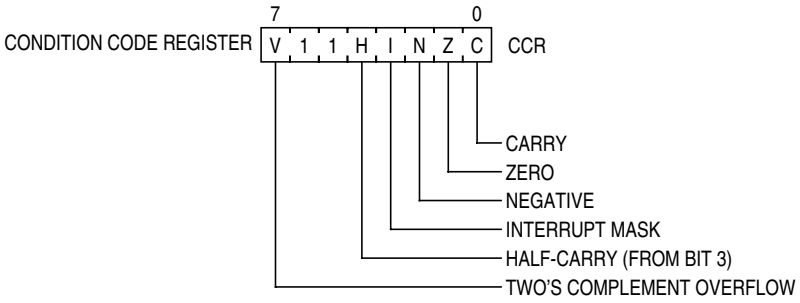


Figure 7-2. Condition Code Register

of an operand for a test and then use relative addressing mode to specify the branch destination address when the tested condition is true. For BRCLR, BRSET, CBEQ, and DBNZ, the addressing mode listed in the instruction set tables is the addressing mode needed to access the operand to be tested, and relative addressing mode is implied for the branch destination.

7.3.1 Inherent Addressing Mode (INH)

In this addressing mode, operands needed to complete the instruction (if any) are located within CPU registers so the CPU does not need to access memory to get any operands.

7.3.2 Relative Addressing Mode (REL)

Relative addressing mode is used to specify the destination location for branch instructions. A signed 8-bit offset value is located in the memory location immediately following the opcode. During execution, if the branch condition is true, the signed offset is sign-extended to a 16-bit value and is added to the current contents of the program counter, which causes program execution to continue at the branch destination address.

7.3.3 Immediate Addressing Mode (IMM)

In immediate addressing mode, the operand needed to complete the instruction is included in the object code immediately following the instruction opcode in memory. In the case of a 16-bit immediate operand, the high-order byte is located in the next memory location after the opcode, and the low-order byte is located in the next memory location after that.

7.3.4 Direct Addressing Mode (DIR)

In direct addressing mode, the instruction includes the low-order eight bits of an address in the direct page (0x0000–0x00FF). During execution a 16-bit address is formed by concatenating an implied 0x00 for the high-order half of the address and the direct address from the instruction to get the 16-bit address where the desired operand is located. This is faster and more memory efficient than specifying a complete 16-bit address for the operand.

7.3.5 Extended Addressing Mode (EXT)

In extended addressing mode, the full 16-bit address of the operand is located in the next two bytes of program memory after the opcode (high byte first).

7.3.6 Indexed Addressing Mode

Indexed addressing mode has seven variations including five that use the 16-bit H:X index register pair and two that use the stack pointer as the base reference.

0	=	Bit forced to 0
1	=	Bit forced to 1
	=	Bit set or cleared according to results of operation
U	=	Undefined after the operation

Machine coding notation

dd	=	Low-order 8 bits of a direct address 0x0000–0x00FF (high byte assumed to be 0x00)
ee	=	Upper 8 bits of 16-bit offset
ff	=	Lower 8 bits of 16-bit offset or 8-bit offset
ii	=	One byte of immediate data
jj	=	High-order byte of a 16-bit immediate data value
kk	=	Low-order byte of a 16-bit immediate data value
hh	=	High-order byte of 16-bit extended address
ll	=	Low-order byte of 16-bit extended address
rr	=	Relative offset

Source form

Everything in the source forms columns, *except expressions in italic characters,* is literal information that must appear in the assembly source file exactly as shown. The initial 3- to 5-letter mnemonic is always a literal expression. All commas, pound signs (#), parentheses, and plus signs (+) are literal characters.

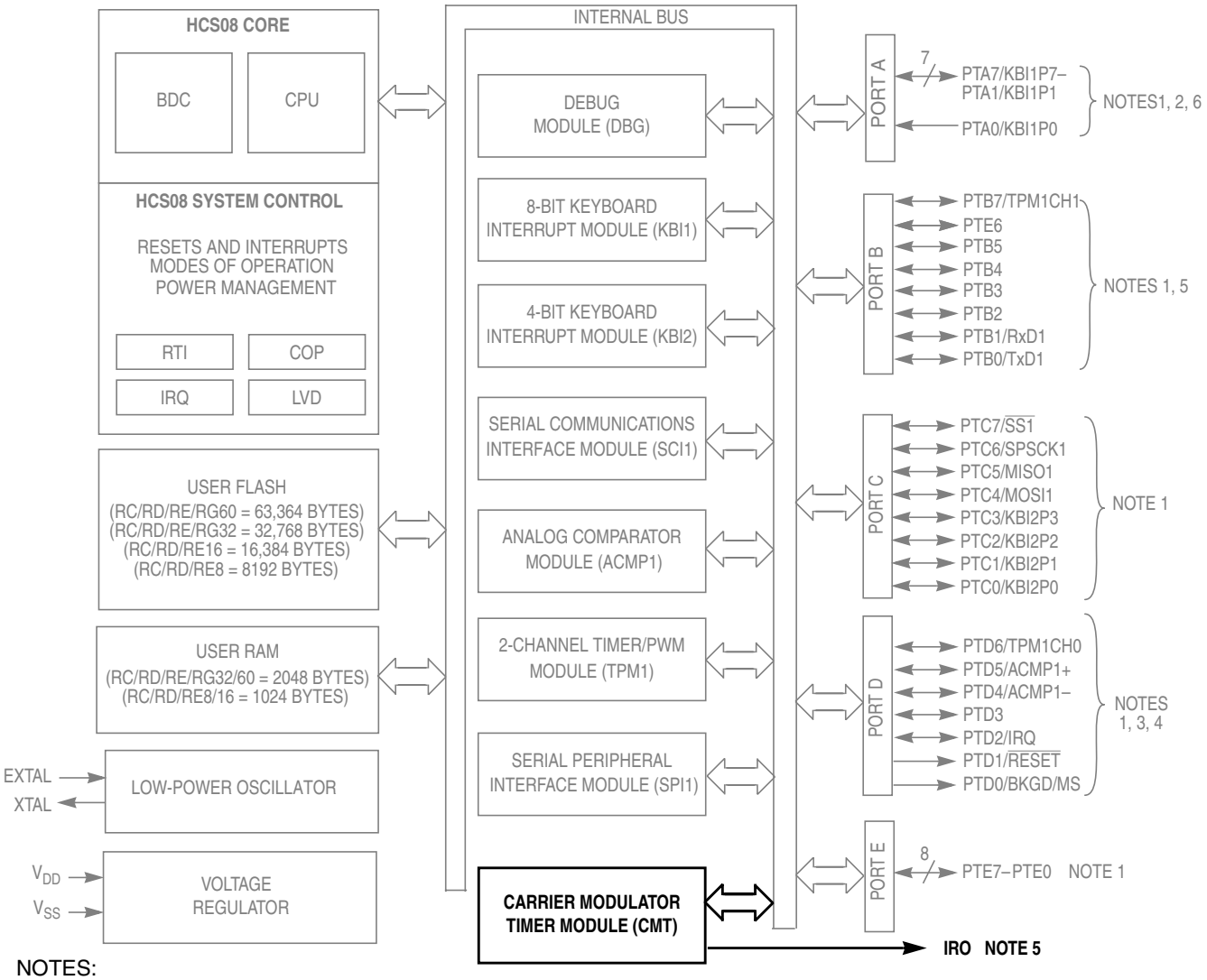
n	—	Any label or expression that evaluates to a single integer in the range 0–7
opr8i	—	Any label or expression that evaluates to an 8-bit immediate value
opr16i	—	Any label or expression that evaluates to a 16-bit immediate value
opr8a	—	Any label or expression that evaluates to an 8-bit value. The instruction treats this 8-bit value as the low order 8 bits of an address in the direct page of the 64-Kbyte address space (0x00xx).
opr16a	—	Any label or expression that evaluates to a 16-bit value. The instruction treats this value as an address in the 64-Kbyte address space.
oprx8	—	Any label or expression that evaluates to an unsigned 8-bit value, used for indexed addressing
oprx16	—	Any label or expression that evaluates to a 16-bit value. Because the HCS08 has a 16-bit address bus, this can be either a signed or an unsigned value.
rel	—	Any label or expression that refers to an address that is within –128 to +127 locations from the next address after the last byte of object code for the current instruction. The assembler will calculate the 8-bit signed offset and include it in the object code for this instruction.

Address modes

INH	=	Inherent (no operands)
IMM	=	8-bit or 16-bit immediate
DIR	=	8-bit direct
EXT	=	16-bit extended

Chapter 8
Carrier Modulator Timer (S08CMTV1)

8.1 Introduction



NOTES:
1. Port pins are software configurable with pullup device if input port
2. PTA0 does not have a clamp diode to VDD. PTA0 should not be driven above VDD. Also, PTA0 does not pullup to VDD when internal pullup is enabled.
3. IRQ pin contains software configurable pullup/pulldown device if IRQ enabled (IRQPE = 1)
4. The RESET pin contains integrated pullup device enabled if reset enabled (RSTPE = 1)
5. High current drive
6. Pins PTA[7:4] contain both pullup and pulldown devices. Pulldown enabled when KBI is enabled (KBIPEn = 1) and rising edge is selected (KBEDGn = 1).

Figure 8-1. MC9S08RC/RD/RE/RG Block Diagram

$$t_{exspace} = t_{space} + (t_{mark} + t_{space}) \times (\text{number of modulation periods}) \quad \textit{Eqn. 8-9}$$

For an example of extended space operation, see Figure 8-7.

NOTE

The EXSPC feature can be used to emulate a zero mark event.

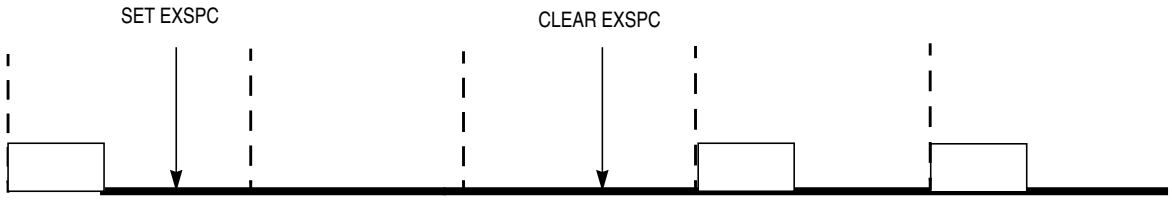


Figure 8-7. Extended Space Operation

8.5.3.2 EXSPC Operation in FSK Mode

In FSK mode, the modulator continues to count carrier out clocks, alternating between the primary and secondary registers at the end of each modulation period.

To calculate the length of an extended space in FSK mode, the user must know whether the EXSPC bit was set on a primary or secondary modulation period, as well as the total number of both primary and secondary modulation periods completed while the EXSPC bit is high. A status bit for the current modulation is not accessible to the CPU. If necessary, software should maintain tracking of the current modulation cycle (primary or secondary). The extended space period ends at the completion of the space period time of the modulation period during which the EXSPC bit is cleared.

If the EXSPC bit was set during a primary modulation cycle, use the equation:

$$t_{exspace} = (t_{space})_p + (t_{mark} + t_{space})_s + (t_{mark} + t_{space})_p + \ldots \quad \textit{Eqn. 8-10}$$

Where the subscripts p and s refer to mark and space times for the primary and secondary modulation cycles.

If the EXSPC bit was set during a secondary modulation cycle, use the equation:

$$t_{exspace} = (t_{space})_s + (t_{mark} + t_{space})_p + (t_{mark} + t_{space})_s + \ldots \quad \textit{Eqn. 8-11}$$

8.5.4 Transmitter

The transmitter output block controls the state of the infrared out pin (IRO). The modulator output is gated on to the IRO pin when the modulator/carrier generator is enabled. When the modulator/carrier generator is disabled, the IRO pin is controlled by the state of the IRO latch.

A polarity bit in the CMTOC register enables the IRO pin to be high true or low true.

output compare, and edge-aligned PWM functions. The timer counter modulo registers, TPM1MODH:TPM1MODL, control the modulo value of the counter. (The values $0000 or $FFFF effectively make the counter free running.) Software can read the counter value at any time without affecting the counting sequence. Any write to either byte of the TPM1CNT counter resets the counter regardless of the data value written.

All TPM channels are programmable independently as input capture, output compare, or buffered edge-aligned PWM channels.

10.4 Pin Descriptions

Table 10-2 shows the MCU pins related to the TPM module. When TPM1CH0 is used as an external clock input, the associated TPM channel 0 can not use the pin. (Channel 0 can still be used in output compare mode as a software timer.) When any of the pins associated with the timer is configured as a timer input, a passive pullup can be enabled. After reset, the TPM modules are disabled and all pins default to general-purpose inputs with the passive pullups disabled.

10.4.1 External TPM Clock Sources

When control bits CLKSB:CLKSA in the timer status and control register are set to 1:1, the prescaler and consequently the 16-bit counter for TPM1 are driven by an external clock source connected to the TPM1CH0 pin. A synchronizer is needed between the external clock and the rest of the TPM. This synchronizer is clocked by the bus clock so the frequency of the external source must be less than one-half the frequency of the bus rate clock. The upper frequency limit for this external clock source is specified to be one-fourth the bus frequency to conservatively accommodate duty cycle and phase-locked loop (PLL) or frequency-locked loop (FLL) frequency jitter effects.

When the TPM is using the channel 0 pin for an external clock, the corresponding ELS0B:ELS0A control bits should be set to 0:0 so channel 0 is not trying to use the same pin.

10.4.2 TPM1CHn — TPM1 Channel n I/O Pins

Each TPM channel is associated with an I/O pin on the MCU. The function of this pin depends on the configuration of the channel. In some cases, no pin function is needed so the pin reverts to being controlled by general-purpose I/O controls. When a timer has control of a port pin, the port data and data direction registers do not affect the related pin(s). See the Pins and Connections chapter for additional information about shared pin functions.

10.5 Functional Description

All TPM functions are associated with a main 16-bit counter that allows flexible selection of the clock source and prescale divisor. A 16-bit modulo register also is associated with the main 16-bit counter in the TPM. Each TPM channel is optionally associated with an MCU pin and a maskable interrupt function.

The TPM has center-aligned PWM capabilities controlled by the CPWMS control bit in TPM1SC. When CPWMS is set to 1, timer counter TPM1CNT changes to an up-/down-counter and all channels in the associated TPM act as center-aligned PWM channels. When CPWMS = 0, each channel can

When CPHA = 1, the slave begins to drive its MISO output when $\overline{SS1}$ goes to active low, but the data is not defined until the first SPSCK edge. The first SPSCK edge shifts the first bit of data from the shifter onto the MOSI output of the master and the MISO output of the slave. The next SPSCK edge causes both the master and the slave to sample the data bit values on their MISO and MOSI inputs, respectively. At the third SPSCK edge, the SPI shifter shifts one bit position which shifts in the bit value that was just sampled, and shifts the second data bit value out the other end of the shifter to the MOSI and MISO outputs of the master and slave, respectively. When CHPA = 1, the slave's $\overline{SS}$ input is not required to go to its inactive high level between transfers.

Figure 13-6 shows the clock formats when CPHA = 0. At the top of the figure, the eight bit times are shown for reference with bit 1 starting as the slave is selected ($\overline{SS}$ IN goes low), and bit 8 ends at the last SPSCK edge. The MSB first and LSB first lines show the order of SPI data bits depending on the setting in LSBFE. Both variations of SPSCK polarity are shown, but only one of these waveforms applies for a specific transfer, depending on the value in CPOL. The SAMPLE IN waveform applies to the MOSI input of a slave or the MISO input of a master. The MOSI waveform applies to the MOSI output pin from a master and the MISO waveform applies to the MISO output from a slave. The $\overline{SS}$ OUT waveform applies to the slave select output from a master (provided MODFEN and SSOE = 1). The master $\overline{SS}$ output goes to active low at the start of the first bit time of the transfer and goes back high one-half SPSCK cycle after the end of the eighth bit time of the transfer. The $\overline{SS}$ IN waveform applies to the slave select input of a slave.

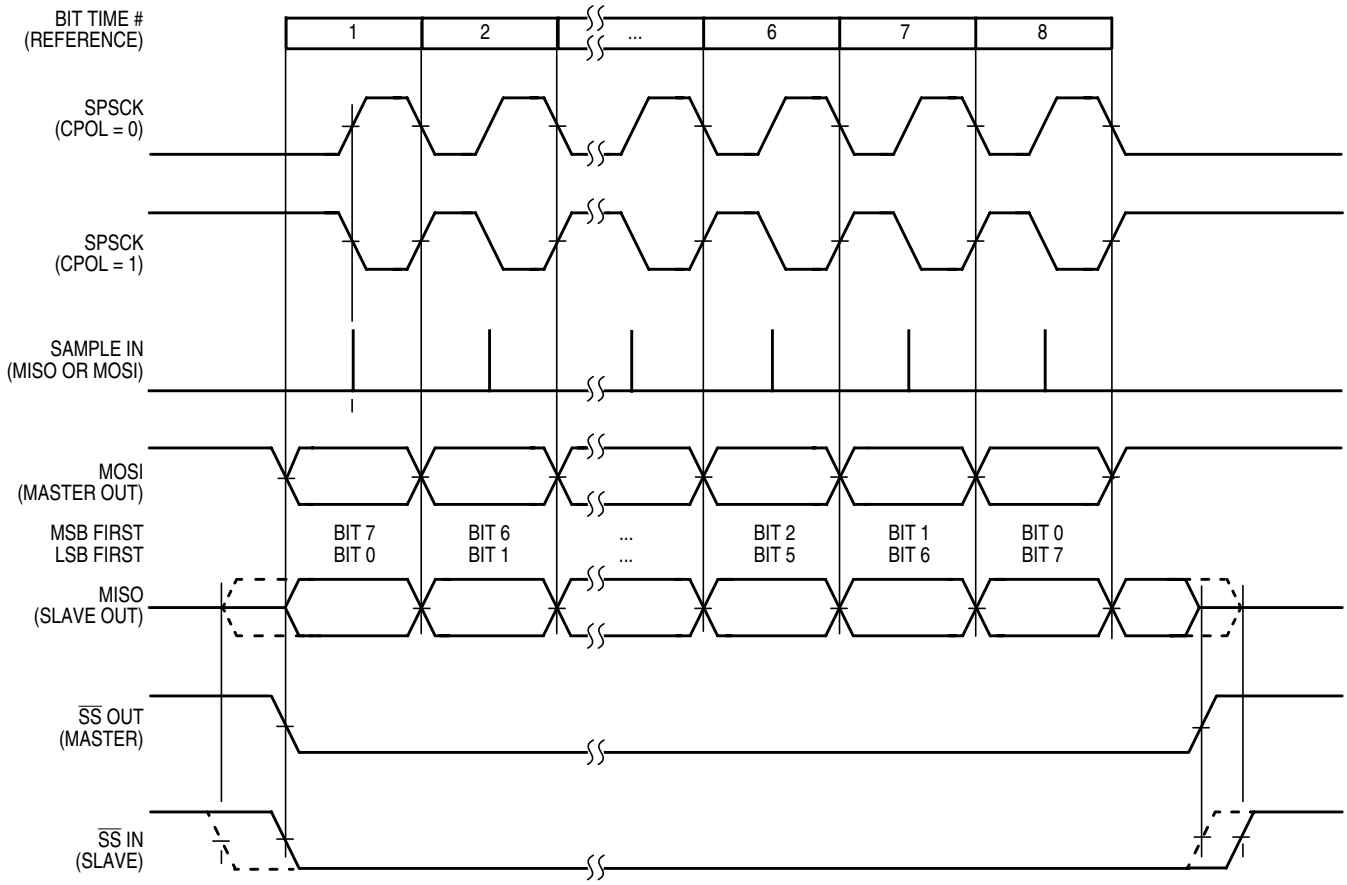


Figure 13-6. SPI Clock Formats (CPHA = 0)

— Inside range (A ≤ address ≤ B)
— Outside range (address < A or address > B)

15.2 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.
- Non-intrusive commands can be executed at any time even while the user's program is running. Non-intrusive commands allow a user to read or write MCU memory locations or access status and control registers within the background debug controller.

Typically, a relatively simple interface pod is used to translate commands from a host computer into commands for the custom serial interface to the single-wire background debug system. Depending on the development tool vendor, this interface pod may use a standard RS-232 serial port, a parallel printer port, or some other type of communications such as a universal serial bus (USB) to communicate between the host PC and the pod. The pod typically connects to the target system with ground, the BKGD pin, $\overline{\text{RESET}}$, and sometimes V_{DD}. An open-drain connection to reset allows the host to force a target system reset, which is useful to regain control of a lost target system or to control startup of a target system before the on-chip nonvolatile memory has been programmed. Sometimes V_{DD} can be used to allow the pod to use power from the target system to avoid the need for a separate power supply. However, if the pod is powered separately, it can be connected to a running target system without forcing a target system reset or otherwise disturbing the running application program.

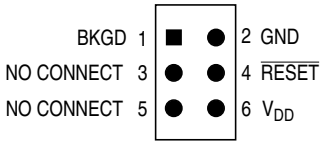


Figure 15-1. BDM Tool Connector

15.2.1 BKGD Pin Description

BKGD is the single-wire background debug interface pin. The primary function of this pin is for bidirectional serial communication of active background mode commands and data. During reset, this pin is used to select between starting in active background mode or starting the user's application program. This pin is also used to request a timed sync response pulse to allow a host development tool to determine the correct clock frequency for background debug serial communications.

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 15.3.5, “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.

15.3.3 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.

15.3.4 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.