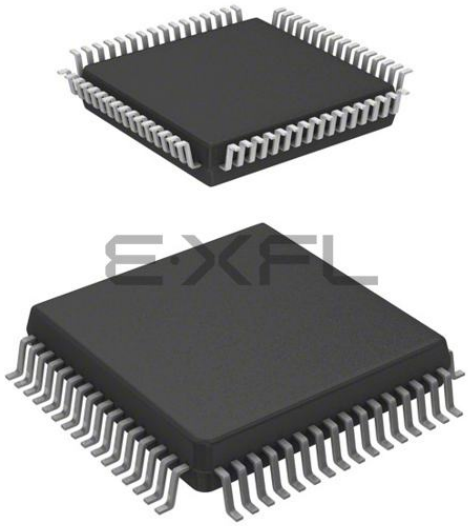


**Welcome to E-XFL.COM**

## 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 | HC08 |
| Core Size | 8-Bit |
| Speed | 8.4MHz |
| Connectivity | CANbus, SCI, SPI |
| Peripherals | LVD, POR, PWM |
| Number of I/O | 52 |
| Program Memory Size | 60KB (60K x 8) |
| Program Memory Type | FLASH |
| EEPROM Size | 1K x 8 |
| RAM Size | 2K x 8 |
| Voltage - Supply (Vcc/Vdd) | 4.5V ~ 5.5V |
| Data Converters | A/D 15x8b |
| Oscillator Type | Internal |
| Operating Temperature | -40°C ~ 105°C (TA) |
| Mounting Type | Surface Mount |
| Package / Case | 64-QFP |
| Supplier Device Package | 64-QFP (14x14) |
| Purchase URL | https://www.e-xfl.com/product-detail/nxp-semiconductors/mc908as60avfue |

## Chapter 14
## Monitor ROM (MON)



## Chapter 15
## Computer Operating Properly (COP)



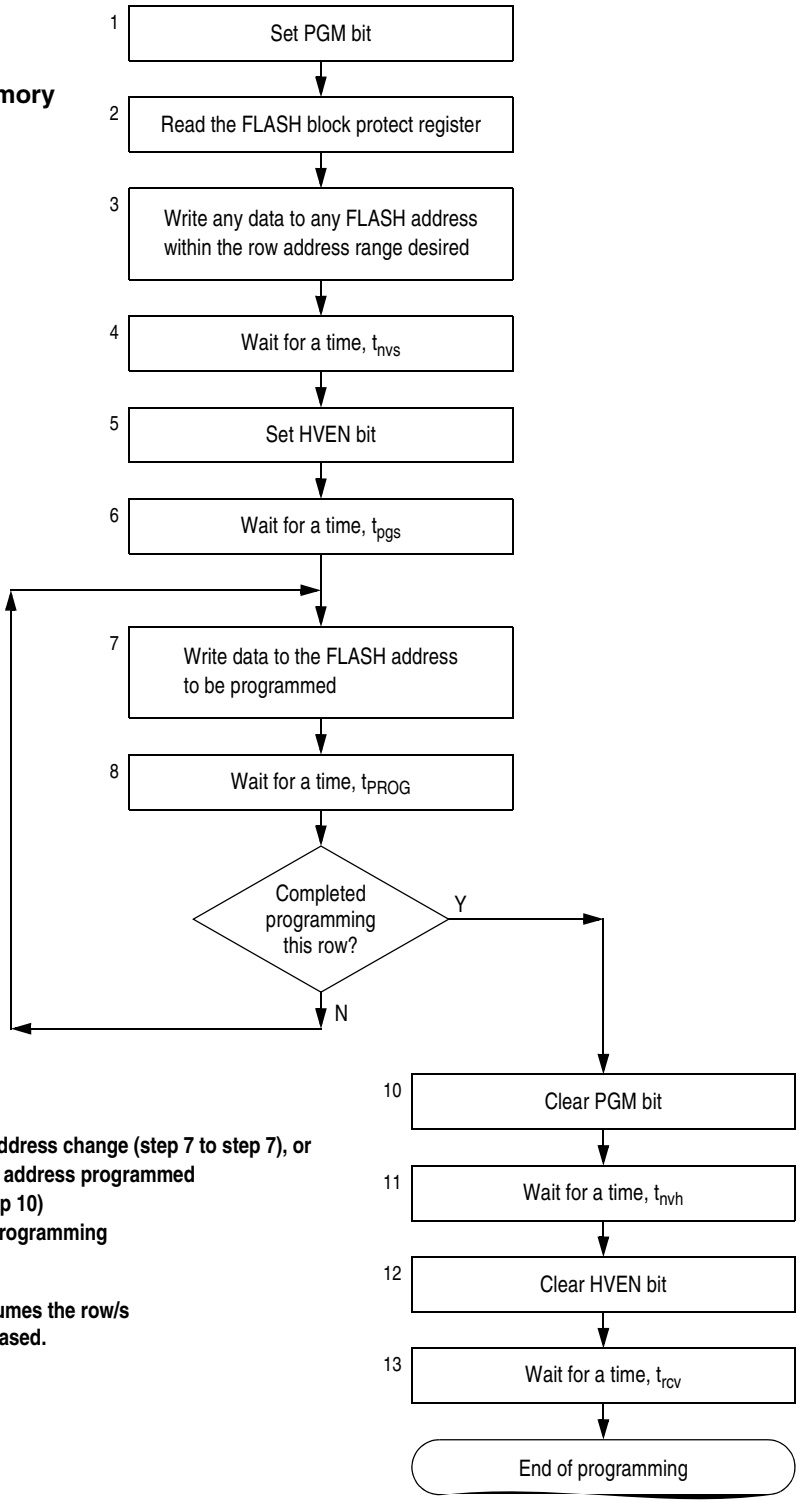
## Chapter 16
## Low-Voltage Inhibit (LVI)

| Addr. | Register Name | | Bit 7 | 6 | 5 | 4 | 3 | 2 | 1 | Bit 0 |
|---|---|---|---|---|---|---|---|---|---|---|
| $0032 | Timer A Channel 4 Status and Control Register (TASC4) | Read: | CH4F | CH4IE | MS4B | MS4A | ELS4B | ELS4A | TOV4 | CH4MAX |
| | | Write: | 0 | | | | | | | |
| $0033 | Timer A Channel 4 Register High (TACH4H) | Read: | Bit 15 | 14 | 13 | 12 | 11 | 10 | 9 | Bit 8 |
| | | Write: | | | | | | | | |
| $0034 | Timer A Channel 4 Register Low (TACH4L) | Read: | Bit 7 | 6 | 5 | 4 | 3 | 2 | 1 | Bit 0 |
| | | Write: | | | | | | | | |
| $0035 | Timer A Channel 5 Status and Control Register (TASC5) | Read: | CH5F | CH5IE | 0 | MS5A | ELS5B | ELS5A | TOV5 | CH5MAX |
| | | Write: | 0 | | R | | | | | |
| $0036 | Timer A Channel 5 Register High (TACH5H) | Read: | Bit 15 | 14 | 13 | 12 | 11 | 10 | 9 | Bit 8 |
| | | Write: | | | | | | | | |
| $0037 | Timer A Channel 5 Register Low (TACH5L) | Read: | Bit 7 | 6 | 5 | 4 | 3 | 2 | 1 | Bit 0 |
| | | Write: | | | | | | | | |
| $0038 | Analog-to-Digital Status and Control Register (ADSCR) | Read: | COCO | AIEN | ADCO | ADCH4 | ADCH3 | ADCH2 | ADCH1 | ADCH0 |
| | | Write: | R | | | | | | | |
| $0039 | Analog-to-Digital Data Register (ADR) | Read: | AD7 | AD6 | AD5 | AD4 | AD3 | AD2 | AD1 | AD0 |
| | | Write: | | | | | | | | |
| $003A | Analog-to-Digital Input Clock Register (ADICLK) | Read: | ADIV2 | ADIV1 | ADIV0 | ADICLK | 0 | 0 | 0 | 0 |
| | | Write: | | | | | | | | |
| $003B | BDLC Analog and Roundtrip Delay Register (BARD) | Read: | ATE | RXPOL | 0 | 0 | BO3 | BO2 | BO1 | BO0 |
| | | Write: | | | R | R | | | | |
| $003C | BDLC Control Register 1 (BCR1) | Read: | IMSG | CLKS | R1 | R0 | 0 | 0 | IE | WCM |
| | | Write: | | | | | R | R | | |
| $003D | BDLC Control Register 2 (BCR2) | Read: | ALOOP | DLOOP | RX4XE | NBFS | TEOD | TSIFR | TMIFR1 | TMIFR0 |
| | | Write: | | | | | | | | |
| $003E | BDLC State Vector Register (BSVR) | Read: | 0 | 0 | I3 | I2 | I1 | I0 | 0 | 0 |
| | | Write: | R | R | R | R | R | R | R | R |
| $003F | BDLC Data Register (BDR) | Read: | BD7 | BD6 | BD5 | BD4 | BD3 | BD2 | BD1 | BD0 |
| | | Write: | | | | | | | | |
| $0040 | Timer B Status and Control Register (TBSCR) | Read: | TOF | TOIE | TSTOP | 0 | 0 | PS2 | PS1 | PS0 |
| | | Write: | 0 | | | TRST | R | | | |
| $0041 | Timer B Counter Register High (TBCNTH) | Read: | Bit 15 | 14 | 13 | 12 | 11 | 10 | 9 | Bit 8 |
| | | Write: | | | | | | | | |
| $0042 | Timer B Counter Register Low (TBCNTL) | Read: | Bit 7 | 6 | 5 | 4 | 3 | 2 | 1 | Bit 0 |
| | | Write: | | | | | | | | |

= Unimplemented  R = Reserved

**Figure 2-2. I/O Data, Status and Control Registers (Sheet 4 of 5)**

**Algorithm for programming a row (64 bytes) of FLASH memory**




**NOTE:**

**The time between each FLASH address change (step 7 to step 7), or the time between the last FLASH address programmed to clearing PGM bit (step 7 to step 10) must not exceed the maximum programming time, $t_{PROG}$ max.**

**This row program algorithm assumes the row/s to be programmed are initially erased.**

**Figure 4-4. FLASH Programming Algorithm Flowchart**

### 6.4.5 EEPROM-1 Programming and Erasing

The unprogrammed or erase state of an EEPROM bit is a logic 1. The factory default for all bytes within the EEPROM-1 array is $FF.

The programming operation changes an EEPROM bit from logic 1 to logic 0 (programming cannot change a bit from logic 0 to a logic 1). In a single programming operation, the minimum EEPROM programming size is one bit; the maximum is eight bits (one byte).

The erase operation changes an EEPROM bit from logic 0 to logic 1. In a single erase operation, the minimum EEPROM erase size is one byte; the maximum is the entire EEPROM-1 array.

The EEPROM can be programmed such that one or multiple bits are programmed (written to a logic 0) at a time. However, the user may never program the same bit location more than once before erasing the entire byte. In other words, the user is not allowed to program a logic 0 to a bit that is already programmed (bit state is already logic 0).

For some applications it might be advantageous to track more than 10K events with a single byte of EEPROM by programming one bit at a time. For that purpose, a special selective bit programming technique is available. An example of this technique is illustrated in Table 6-2.

**Table 6-2. Example Selective Bit Programming Description**

| Description | Program Data in Binary | Result in Binary |
|---|---|---|
| Original state of byte (erased) | n/a | 1111:1111 |
| First event is recorded by programming bit position 0 | 1111:1110 | 1111:1110 |
| Second event is recorded by programming bit position 1 | 1111:1101 | 1111:1100 |
| Third event is recorded by programming bit position 2 | 1111:1011 | 1111:1000 |
| Fourth event is recorded by programming bit position 3 | 1111:0111 | 1111:0000 |
| Events five through eight are recorded in a similar fashion | | |

Note that none of the bit locations are actually programmed more than once although the byte was programmed eight times.

When this technique is utilized, a program/erase cycle is defined as multiple program sequences (up to eight) to a unique location followed by a single erase operation.

#### *6.4.5.1 Program/Erase Using AUTO Bit*

An additional feature available for EEPROM-1 program and erase operations is the AUTO mode. When enabled, AUTO mode will activate an internal timer that will automatically terminate the program/erase cycle and clear the EEPGM bit. Please see 6.4.5.2 EEPROM-1 Programming, 6.4.5.3 EEPROM-1 Erasing, and 6.5.1 EEPROM-1 Control Register for more information.

### 7.4.3 EEPROM-2 Program/Erase Protection

The EEPROM has a special feature that designates the 16 bytes of addresses from $06F0 to $06FF to be permanently secured. This program/erase protect option is enabled by programming the EEPRTCT bit in the EEPROM-2 Nonvolatile Register (EE2NVR) to a logic zero.

Once the EEPRTCT bit is programmed to 0 for the first time:

- Programming and erasing of secured locations $06F0 to $06FF is permanently disabled.
- Secured locations $06F0 to $06FF can be read as normal.
- Programming and erasing of EE2NVR is permanently disabled.
- Bulk and Block Erase operations are disabled for the unprotected locations $0600-$06EF, $0700-$07FF.
- Single byte program and erase operations are still available for locations $0600-$06EF and $0700-$07FF for all bytes that are not protected by the EEPROM-2 Block Protect EEBPx bits (see 7.4.4 EEPROM-2 Block Protection and 7.5.2 EEPROM-2 Array Configuration Register)

***NOTE***

*Once armed, the protect option is permanently enabled. As a consequence, all functions in the EE2NVR will remain in the state they were in immediately before the security was enabled.*

### 7.4.4 EEPROM-2 Block Protection

The 512 bytes of EEPROM-2 are divided into four 128-byte blocks. Each of these blocks can be protected from erase/program operations by setting the EEBPx bit in the EE2NVR. Table 7-1 shows the address ranges for the blocks.

**Table 7-1. EEPROM-2 Array Address Blocks**

| Block Number (EEBPx) | Address Range |
|---|---|
| EEBP0 | $0600–$067F |
| EEBP1 | $0680–$06FF |
| EEBP2 | $0700–$077F |
| EEBP3 | $0780–$07FF |

These bits are effective after a reset or a upon read of the EE2NVR register. The block protect configuration can be modified by erasing/programming the corresponding bits in the EE2NVR register and then reading the EE2NVR register. Please see 7.5.2 EEPROM-2 Array Configuration Register for more information.

***NOTE***

*Once EEDIVSECD in the EE2DIVHNVR is programmed to 0 and after a system reset, the EE2DIV security feature is permanently enabled because the EEDIVSECD bit in the EE2DIVH is always loaded with 0 thereafter. Once this security feature is armed, erase and program mode are disabled for EE2DIVHNVR and EE2DIVLNVR. Modifications to the EE2DIVH and EE2DIVL registers are also disabled. Therefore, be cautious on programming a value into the EE2DIVHNVR.*

### 8.3.5 Condition Code Register

The 8-bit condition code register contains the interrupt mask 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 register.

| | Bit 7 | 6 | 5 | 4 | 3 | 2 | 1 | Bit 0 |
|---|---|---|---|---|---|---|---|---|
| Read:<br>Write: | V | 1 | 1 | H | I | N | Z | C |
| Reset: | X | 1 | 1 | X | 1 | X | X | X |

X = Indeterminate

**Figure 8-6. Condition Code Register (CCR)**

**V — Overflow Flag**

The CPU sets the overflow flag when a two's complement overflow occurs. The signed branch instructions BGT, BGE, BLE, and BLT use the overflow flag.

1 = Overflow
0 = No overflow

**H — Half-Carry Flag**

The CPU sets the half-carry flag when a carry occurs between accumulator bits 3 and 4 during an add-without-carry (ADD) or add-with-carry (ADC) operation. The half-carry flag is required for binary-coded decimal (BCD) arithmetic operations. The DAA instruction uses the states of the H and C flags to determine the appropriate correction factor.

1 = Carry between bits 3 and 4
0 = No carry between bits 3 and 4

**I — Interrupt Mask**

When the interrupt mask is set, all maskable CPU interrupts are disabled. CPU interrupts are enabled when the interrupt mask is cleared. When a CPU interrupt occurs, the interrupt mask is set automatically after the CPU registers are saved on the stack, but before the interrupt vector is fetched.

1 = Interrupts disabled
0 = Interrupts enabled

***NOTE***

*To maintain M6805 Family compatibility, the upper byte of the index register (H) is not stacked automatically. If the interrupt service routine modifies H, then the user must stack and unstack H using the PSHH and PULH instructions.*

After the I bit is cleared, the highest-priority interrupt request is serviced first.

A return-from-interrupt (RTI) instruction pulls the CPU registers from the stack and restores the interrupt mask from the stack. After any reset, the interrupt mask is set and can be cleared only by the clear interrupt mask software instruction (CLI).

**N — Negative Flag**

The CPU sets the negative flag when an arithmetic operation, logic operation, or data manipulation produces a negative result, setting bit 7 of the result.

1 = Negative result
0 = Non-negative result

# Chapter 9
# System Integration Module (SIM)

## 9.1 Introduction

This chapter describes the system integration module (SIM), which supports up to 32 external and/or internal interrupts. Together with the central processor unit (CPU), the SIM controls all MCU activities. A block diagram of the SIM is shown in Figure 9-1. Figure 9-2 is a summary of the SIM input/output (I/O) registers. The SIM is a system state controller that coordinates CPU and exception timing. The SIM is responsible for:

- Bus clock generation and control for CPU and peripherals
  - Stop/wait/reset/break entry and recovery
  - Internal clock control
- Master reset control, including power-on reset (POR) and computer operating properly (COP) timeout
- Interrupt control:
  - Acknowledge timing
  - Arbitration control timing
  - Vector address generation
- CPU enable/disable timing

| Register Name | | Bit 7 | 6 | 5 | 4 | 3 | 2 | 1 | Bit 0 |
|---|---|---|---|---|---|---|---|---|---|
| PLL Control Register (PCTL) | Read: | PLLIE | PLLF | PLLON | BCS | 1 | 1 | 1 | 1 |
| | Write: | | | | | | | | |
| | Reset: | 0 | 0 | 1 | 0 | 1 | 1 | 1 | 1 |
| PLL Bandwidth Control Register (PBWC) | Read: | AUTO | LOCK | $\overline{\text{ACQ}}$ | XLD | 0 | 0 | 0 | 0 |
| | Write: | | | | | | | | |
| | Reset: | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| PLL Programming Register (PPG) | Read: | MUL7 | MUL6 | MUL5 | MUL4 | VRS7 | VRS6 | VRS5 | VRS4 |
| | Write: | | | | | | | | |
| | Reset: | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 0 |

= Unimplemented

**Figure 10-2. I/O Register Summary**

**Table 10-1. I/O Register Address Summary**

| **Register** | PCTL | PBWC | PPG |
|---|---|---|---|
| **Address** | $001C | $001D | $001E |

### 10.3.1 Crystal Oscillator Circuit

The crystal oscillator circuit consists of an inverting amplifier and an external crystal. The OSC1 pin is the input to the amplifier and the OSC2 pin is the output. The SIMOSCEN signal enables the crystal oscillator circuit.

The CGMXCLK signal is the output of the crystal oscillator circuit and runs at a rate equal to the crystal frequency. CGMXCLK is then buffered to produce CGMRCLK, the PLL reference clock.

CGMXCLK can be used by other modules which require precise timing for operation. The duty cycle of CGMXCLK is not guaranteed to be 50% and depends on external factors, including the crystal and related external components.

An externally generated clock also can feed the OSC1 pin of the crystal oscillator circuit. Connect the external clock to the OSC1 pin and let the OSC2 pin float.

### 10.3.2 Phase-Locked Loop Circuit (PLL)

The PLL is a frequency generator that can operate in either acquisition mode or tracking mode, depending on the accuracy of the output frequency. The PLL can change between acquisition and tracking modes either automatically or manually.

| Register Name | | Bit 7 | 6 | 5 | 4 | 3 | 2 | 1 | Bit 0 |
|---|---|---|---|---|---|---|---|---|---|
| SCI Control Register 1 (SCC1) | Read: | LOOPS | ENSCI | TXINV | M | WAKE | ILTY | PEN | PTY |
| | Write: | | | | | | | | |
| | Reset: | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| SCI Control Register 2 (SCC2) | Read: | SCTIE | TCIE | SCRIE | ILIE | TE | RE | RWU | SBK |
| | Write: | | | | | | | | |
| | Reset: | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| SCI Control Register 3 (SCC3) | Read: | R8 | T8 | R | R | ORIE | NEIE | FEIE | PEIE |
| | Write: | | | | | | | | |
| | Reset: | U | U | 0 | 0 | 0 | 0 | 0 | 0 |
| SCI Status Register 1 (SCS1) | Read: | SCTE | TC | SCRF | IDLE | OR | NF | FE | PE |
| | Write: | | | | | | | | |
| | Reset: | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| SCI Status Register 2 (SCS2) | Read: | 0 | 0 | 0 | 0 | 0 | 0 | BKF | RPF |
| | Write: | | | | | | | | |
| | Reset: | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| SCI Data Register (SCDR) | Read: | R7 | R6 | R5 | R4 | R3 | R2 | R1 | R0 |
| | Write: | T7 | T6 | T5 | T4 | T3 | T2 | T1 | T0 |
| | Reset: | Unaffected by Reset | | | | | | | |
| SCI Baud Rate Register (SCBR) | Read: | 0 | 0 | SCP1 | SCP0 | R | SCR2 | SCR1 | SCR0 |
| | Write: | | | | | | | | |
| | Reset: | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

= Unimplemented U = Unaffected R = Reserved

**Figure 18-2. SCI I/O Register Summary**

**Table 18-2. SCI I/O Register Address Summary**

| Register | SCC1 | SCC2 | SCC3 | SCS1 | SCS2 | SCDR | SCBR |
|---|---|---|---|---|---|---|---|
| Address | $0013 | $0014 | $0015 | $0016 | $0017 | $0018 | $0019 |

**CPHA — Clock Phase Bit**

This read/write bit controls the timing relationship between the serial clock and SPI data. (See Figure 19-4 and Figure 19-5.) To transmit data between SPI modules, the SPI modules must have identical CPHA bits. When CPHA = 0, the $\overline{SS}$ pin of the slave SPI module must be set to logic 1 between bytes. (See Figure 19-11). Reset sets the CPHA bit.

When CPHA = 0 for a slave, the falling edge of $\overline{SS}$ indicates the beginning of the transmission. This causes the SPI to leave its idle state and begin driving the MISO pin with the MSB of its data. Once the transmission begins, no new data is allowed into the shift register from the data register. Therefore, the slave data register must be loaded with the desired transmit data before the falling edge of $\overline{SS}$. Any data written after the falling edge is stored in the data register and transferred to the shift register at the current transmission.

When CPHA = 1 for a slave, the first edge of the SPSCK indicates the beginning of the transmission. The same applies when $\overline{SS}$ is high for a slave. The MISO pin is held in a high-impedance state, and the incoming SPSCK is ignored. In certain cases, it may also cause the MODF flag to be set. (See 19.6.2 Mode Fault Error). A 1 on the $\overline{SS}$ pin does not in any way affect the state of the SPI state machine.

**SPWOM — SPI Wired-OR Mode Bit**

This read/write bit disables the pullup devices on pins SPSCK, MOSI, and MISO so that those pins become open-drain outputs.

1 = Wired-OR SPSCK, MOSI, and MISO pins
0 = Normal push-pull SPSCK, MOSI, and MISO pins

**SPE — SPI Enable Bit**

This read/write bit enables the SPI module. Clearing SPE causes a partial reset of the SPI (see 19.9 Resetting the SPI). Reset clears the SPE bit.

1 = SPI module enabled
0 = SPI module disabled

**SPTIE — SPI Transmit Interrupt Enable Bit**

This read/write bit enables CPU interrupt requests generated by the SPTE bit. SPTE is set when a byte transfers from the transmit data register to the shift register. Reset clears the SPTIE bit.

1 = SPTE CPU interrupt requests enabled
0 = SPTE CPU interrupt requests disabled

### 19.13.2 SPI Status and Control Register

The SPI status and control register contains flags to signal the following conditions:

- Receive data register full
- Failure to clear SPRF bit before next byte is received (overflow error)
- Inconsistent logic level on $\overline{SS}$ pin (mode fault error)
- Transmit data register empty

The SPI status and control register also contains bits that perform these functions:

- Enable error interrupts
- Enable mode fault error detection
- Select master SPI baud rate

Above behavior cannot be achieved with a single transmit buffer. That buffer must be reloaded right after the previous message has been sent. This loading process lasts a definite amount of time and has to be completed within the inter-frame sequence (IFS) to be able to send an uninterrupted stream of messages. Even if this is feasible for limited CAN bus speeds, it requires that the CPU reacts with short latencies to the transmit interrupt.

A double buffer scheme would de-couple the re-loading of the transmit buffers from the actual message being sent and as such reduces the reactiveness requirements on the CPU. Problems may arise if the sending of a message would be finished just while the CPU re-loads the second buffer. In that case, no buffer would then be ready for transmission and the bus would be released.

At least three transmit buffers are required to meet the first of the above requirements under all circumstances. The MSCAN08 has three transmit buffers.

The second requirement calls for some sort of internal prioritisation which the MSCAN08 implements with the “local priority” concept described in 23.4.2 Receive Structures.

### 23.4.2 Receive Structures

The received messages are stored in a 2-stage input first in first out (FIFO). The two message buffers are mapped using a Ping Pong arrangement into a single memory area (see Figure 23-2). While the background receive buffer (RxBG) is exclusively associated to the MSCAN08, the foreground receive buffer (RxFG) is addressable by the CPU08. This scheme simplifies the handler software, because only one address area is applicable for the receive process.

Both buffers have a size of 13 bytes to store the CAN control bits, the identifier (standard or extended), and the data content (for details, see 23.12 Programmer’s Model of Message Storage).

The receiver full flag (RXF) in the MSCAN08 receiver flag register (CRFLG) (see 23.13.5 MSCAN08 Receiver Flag Register (CRFLG)), signals the status of the foreground receive buffer. When the buffer contains a correctly received message with matching identifier, this flag is set.

On reception, each message is checked to see if it passes the filter (for details see 23.5 Identifier Acceptance Filter) and in parallel is written into RxBG. The MSCAN08 copies the content of RxBG into RxFG[1], sets the RXF flag, and generates a receive interrupt to the CPU[2]. The user’s receive handler has to read the received message from RxFG and to reset the RXF flag to acknowledge the interrupt and to release the foreground buffer. A new message which can follow immediately after the IFS field of the CAN frame, is received into RxBG. The overwriting of the background buffer is independent of the identifier filter function.

When the MSCAN08 module is transmitting, the MSCAN08 receives its own messages into the background receive buffer, RxBG. It does NOT overwrite RxFG, generate a receive interrupt or acknowledge its own messages on the CAN bus. The exception to this rule is in loop-back mode (see 23.13.2 MSCAN08 Module Control Register 1), where the MSCAN08 treats its own messages exactly like all other incoming messages. The MSCAN08 receives its own transmitted messages in the event that it loses arbitration. If arbitration is lost, the MSCAN08 must be prepared to become receiver.

1. Only if the RXF flag is not set.
2. The receive interrupt will occur only if not masked. A polling scheme can be applied on RXF also.

- Time segment 2: This segment represents PHASE_SEG2 of the CAN standard. It can be programmed by setting the TSEG2 parameter to be 2 to 8 time quanta long.

$$\text{Bit rate} = \frac{f_{Tq}}{\text{No. of time quanta}}$$

The synchronization jump width (SJW) can be programmed in a range of 1 to 4 time quanta by setting the SJW parameter.

The above parameters can be set by programming the bus timing registers, CBTR0–CBTR1, see 23.13.3 MSCAN08 Bus Timing Register 0 and 23.13.4 MSCAN08 Bus Timing Register 1).

***NOTE***

*It is the user's responsibility to make sure that the bit timing settings are in compliance with the CAN standard,*

Table 23-8 gives an overview on the CAN conforming segment settings and the related parameter values.

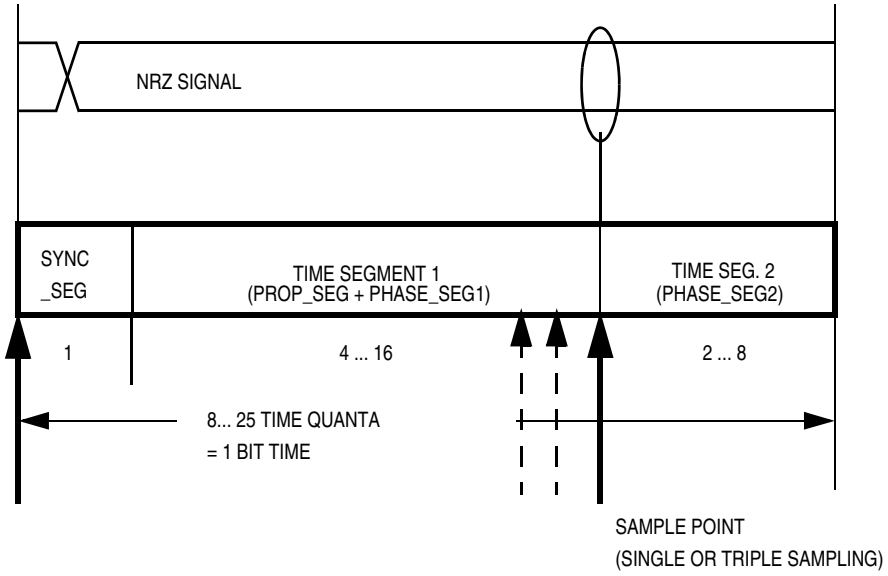



**Figure 23-8. Segments within the Bit Time**

**Table 23-3Time Segment Syntax**

| | |
|---|---|
| SYNC_SEG | System expects transitions to occur on the bus during this period. |
| Transmit point | A node in transmit mode will transfer a new value to the CAN bus at this point. |
| Sample point | A node in receive mode will sample the bus at this point. If the three samples per bit option is selected then this point marks the position of the third sample. |

**Table 23-4. CAN Standard Compliant Bit Time Segment Settings**

| Time Segment 1 | TSEG1 | Time Segment 2 | TSEG2 | Synchron. Jump Width | SJW |
|---|---|---|---|---|---|
| 5 .. 10 | 4 .. 9 | 2 | 1 | 1 .. 2 | 0 .. 1 |
| 4 .. 11 | 3 .. 10 | 3 | 2 | 1 .. 3 | 0 .. 2 |
| 5 .. 12 | 4 .. 11 | 4 | 3 | 1 .. 4 | 0 .. 3 |
| 6 .. 13 | 5 .. 12 | 5 | 4 | 1 .. 4 | 0 .. 3 |
| 7 .. 14 | 6 .. 13 | 6 | 5 | 1 .. 4 | 0 .. 3 |
| 8 .. 15 | 7 .. 14 | 7 | 6 | 1 .. 4 | 0 .. 3 |
| 9 .. 16 | 8 .. 15 | 8 | 7 | 1 .. 4 | 0 .. 3 |

## 23.11 Memory Map

The MSCAN08 occupies 128 bytes in the CPU08 memory space.

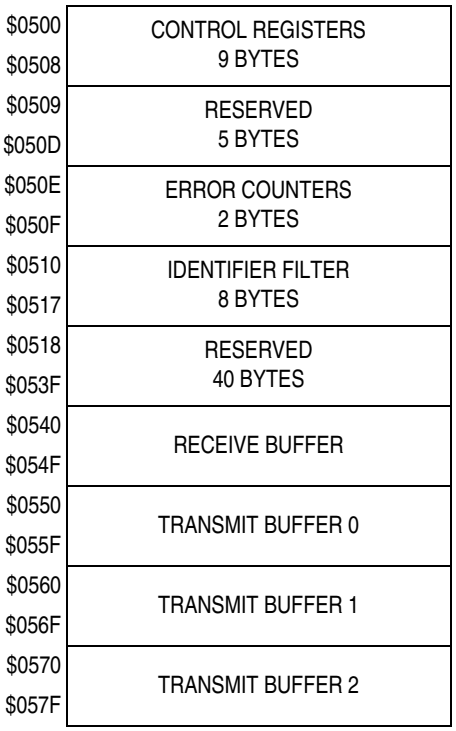



**Figure 23-9. MSCAN08 Memory Map**

## 27.4 BDLC MUX Interface

The MUX interface is responsible for bit encoding/decoding and digital noise filtering between the protocol handler and the physical interface.

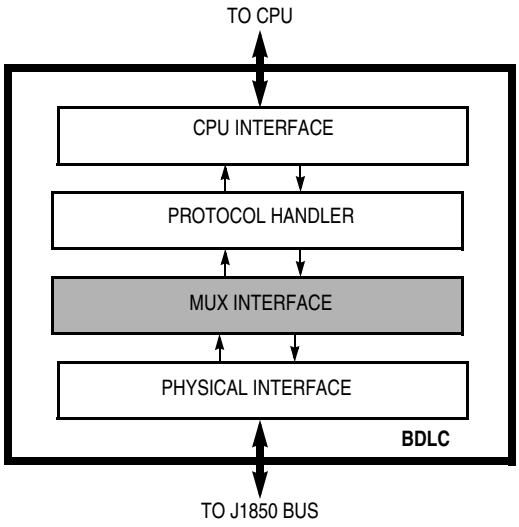



**Figure 27-4. BDLC Block Diagram**

### 27.4.1 Rx Digital Filter

The receiver section of the BDLC includes a digital low pass filter to remove narrow noise pulses from the incoming message. An outline of the digital filter is shown in Figure 27-5.

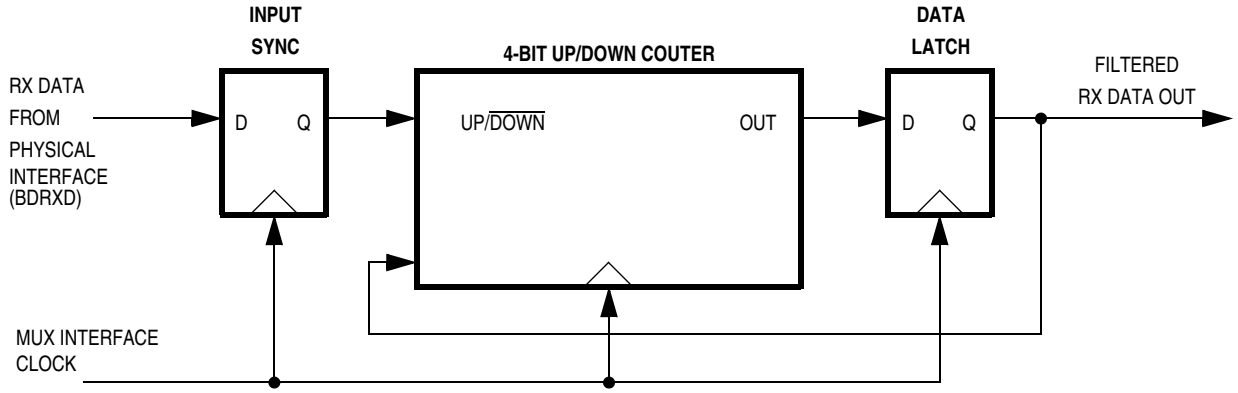



**Figure 27-5. BDLC Rx Digital Filter Block Diagram**

#### *27.4.1.1 Operation*

The clock for the digital filter is provided by the MUX interface clock (see $f_{BDLC}$ parameter in Table 27-3). At each positive edge of the clock signal, the current state of the receiver physical interface (BDRxD) signal is sampled. The BDRxD signal state is used to determine whether the counter should increment or decrement at the next negative edge of the clock signal.

If the BDLC detects a BREAK symbol while receiving a message, it treats the BREAK as a reception error and sets the invalid symbol flag in the BSVR, also ignoring the frame it was receiving. If while receiving a message in 4X mode, the BDLC detects a BREAK symbol, it treats the BREAK as a reception error, sets the invalid symbol flag, and exits 4X mode (for example, the RX4XE bit in BCR2 is cleared automatically). If bus control is required after the BREAK symbol is received and the IFS time has elapsed, the programmer must resend the transmission byte using highest priority.

***NOTE***

*The J1850 protocol BREAK symbol is not related to the HC08 break module. Chapter 13 Break Module (BRK)*

**IDLE — Idle Bus**

An idle condition exists on the bus during any passive period after expiration of the IFS period (for instance, ≥ 300 μs). Any node sensing an idle bus condition can begin transmission immediately.

### 27.4.3 J1850 VPW Symbols

Huntsinger's variable pulse width modulation (VPW) is an encoding technique in which each bit is defined by the time between successive transitions and by the level of the bus between transitions (for instance, active or passive). Active and passive bits are used alternately. This encoding technique is used to reduce the number of bus transitions for a given bit rate.

Each logic 1 or logic 0 contains a single transition and can be at either the active or passive level and one of two lengths, either 64 μs or 128 μs ($t_{NOM}$ at 10.4 kbps baud rate), depending upon the encoding of the previous bit. The start-of-frame (SOF), end-of-data (EOD), end-of-frame (EOF), and inter-frame separation (IFS) symbols always will be encoded at an assigned level and length. See Figure 27-6.

Each message will begin with an SOF symbol an active symbol and, therefore, each data byte (including the CRC byte) will begin with a passive bit, regardless of whether it is a logic 1 or a logic 0.

All VPW bit lengths stated in the following descriptions are typical values at a 10.4 kbps bit rate.

**Logic 0**

A logic 0 is defined as either:

- An active-to-passive transition followed by a passive period 64 μs in length, or
- A passive-to-active transition followed by an active period 128 μs in length

See Figure 27-6. J1850 VPW Symbols with Nominal Symbol Times **(a)**.

**Logic 1**

A logic 1 is defined as either:

- An active-to-passive transition followed by a passive period 128 μs in length, or
- A passive-to-active transition followed by an active period 64 μs in length

See Figure 27-6. J1850 VPW Symbols with Nominal Symbol Times (b).

**Normalization Bit (NB)**

The NB symbol has the same property as a logic 1 or a logic 0. It is only used in IFR message responses.

**Break Signal (BREAK)**

The BREAK signal is defined as a passive-to-active transition followed by an active period of at least 240 μs (See Figure 27-6. J1850 VPW Symbols with Nominal Symbol Times (c)).

Upon receiving a BDLC interrupt, the user can read the value within the BSVR, transferring it to the CPU's index register. The value can then be used to index into a jump table, with entries four bytes apart, to quickly enter the appropriate service routine. For example:

```
Service     LDX    BSVR          Fetch State Vector Number
            JMP    JMPTAB,X      Enter service routine,
*                                (must end in RTI)
*
JMPTAB      JMP    SERVE0        Service condition #0
            NOP
            JMP    SERVE1        Service condition #1
            NOP
            JMP    SERVE2        Service condition #2
            NOP
*
            JMP    SERVE8        Service condition #8
            END
```

***NOTE***

*The NOPs are used only to align the JMPs onto 4-byte boundaries so that the value in the BSVR can be used intact. Each of the service routines must end with an RTI instruction to guarantee correct continued operation of the device. Note also that the first entry can be omitted since it corresponds to no interrupt occurring.*

The service routines should clear all of the sources that are causing the pending interrupts. Note that the clearing of a high priority interrupt may still leave a lower priority interrupt pending, in which case bits I0, I1, and I2 of the BSVR will then reflect the source of the remaining interrupt request.

If fewer states are used or if a different software approach is taken, the jump table can be made smaller or omitted altogether.

## 27.6.5 BDLC Data Register

Address: $003F

| | Bit 7 | 6 | 5 | 4 | 3 | 2 | 1 | Bit 0 |
|---|---|---|---|---|---|---|---|---|
| Read:<br>Write: | D7 | D6 | D5 | D4 | D3 | D2 | D1 | D0 |
| Reset: | Unaffected by Reset | | | | | | | |

**Figure 27-20. BDLC Data Register (BDR)**

This register is used to pass the data to be transmitted to the J1850 bus from the CPU to the BDLC. It is also used to pass data received from the J1850 bus to the CPU. Each data byte (after the first one) should be written only after a Tx data register empty (TDRE) state is indicated in the BSVR.

Data read from this register will be the last data byte received from the J1850 bus. This received data should only be read after an Rx data register full (RDRF) interrupt has occurred. (See 27.6.4 BDLC State Vector Register)

The BDR is double buffered via a transmit shadow register and a receive shadow register. After the byte in the transmit shift register has been transmitted, the byte currently stored in the transmit shadow register is loaded into the transmit shift register. Once the transmit shift register has shifted the first bit out, the TDRE flag is set, and the shadow register is ready to accept the next data byte. The receive shadow register works similarly. Once a complete byte has been received, the receive shift register stores the newly received byte into the receive shadow register. The RDRF flag is set to indicate that a new byte of data has been received. The programmer has one BDLC byte reception time to read the shadow register and clear the RDRF flag before the shadow register is overwritten by the newly received byte.

To abort an in-progress transmission, the programmer should stop loading data into the BDR. This will cause a transmitter underrun error and the BDLC automatically will disable the transmitter on the next non-byte boundary. This means that the earliest a transmission can be halted is after at least one byte plus two extra logic 1s have been transmitted. The receiver will pick this up as an error and relay it in the state vector register as an invalid symbol error.

***NOTE***

The extra logic 1s are an enhancement to the J1850 protocol which forces a byte boundary condition fault. This is helpful in preventing noise from going onto the J1850 bus from a corrupted message.

## 27.7 Low-Power Modes

The following information concerns wait mode and stop mode.

### 27.7.1 Wait Mode

This power-conserving mode is entered automatically from run mode whenever the CPU executes a WAIT instruction and the WCM bit in BDLC control register 1 (BCR1) is previously clear. In BDLC wait mode, the BDLC cannot drive any data.

A subsequent successfully received message, including one that is in progress at the time that this mode is entered, will cause the BDLC to wake up and generate a CPU interrupt request if the interrupt enable (IE) bit in the BDLC control register 1 (BCR1) is previously set. (See 27.6.2 BDLC Control Register 1 for a better understanding of IE.) This results in less of a power saving, but the BDLC is guaranteed to receive correctly the message which woke it up, since the BDLC internal operating clocks are kept running.

***NOTE***

*Ensuring that all transmissions are complete or aborted before putting the BDLC into wait mode is important.*

### 27.7.2 Stop Mode

This power-conserving mode is entered automatically from run mode whenever the CPU executes a STOP instruction or if the CPU executes a WAIT instruction and the WCM bit in the BDLC control register 1 (BCR1) is previously set. This is the lowest power mode that the BDLC can enter.

A subsequent passive-to-active transition on the J1850 bus will cause the BDLC to wake up and generate a non-maskable CPU interrupt request. When a STOP instruction is used to put the BDLC in stop mode, the BDLC is not guaranteed to correctly receive the message which woke it up, since it may take some time for the BDLC internal operating clocks to restart and stabilize. If a WAIT instruction is used to put the BDLC in stop mode, the BDLC is guaranteed to correctly receive the byte which woke it up, if and only if an end-of-frame (EOF) has been detected prior to issuing the WAIT instruction by the CPU. Otherwise, the BDLC will not correctly receive the byte that woke it up.

**high byte** — The most significant eight bits of a word.

**illegal address** — An address not within the memory map

**illegal opcode** — A nonexistent opcode.

**I** — The interrupt mask bit in the condition code register of the CPU08. When I is set, all interrupts are disabled.

**index register (H:X)** — A 16-bit register in the CPU08. The upper byte of H:X is called H. The lower byte is called X. In the indexed addressing modes, the CPU uses the contents of H:X to determine the effective address of the operand. H:X can also serve as a temporary data storage location.

**input/output (I/O)** — Input/output interfaces between a computer system and the external world. A CPU reads an input to sense the level of an external signal and writes to an output to change the level on an external signal.

**instructions** — Operations that a CPU can perform. Instructions are expressed by programmers as assembly language mnemonics. A CPU interprets an opcode and its associated operand(s) and instruction.

**interrupt** — A temporary break in the sequential execution of a program to respond to signals from peripheral devices by executing a subroutine.

**interrupt request** — A signal from a peripheral to the CPU intended to cause the CPU to execute a subroutine.

**I/O** — See “input/output (I/0).”

**IRQ** — See "external interrupt module (IRQ)."

**jitter** — Short-term signal instability.

**latch** — A circuit that retains the voltage level (logic 1 or logic 0) written to it for as long as power is applied to the circuit.

**latency** — The time lag between instruction completion and data movement.

**least significant bit (LSB)** — The rightmost digit of a binary number.

**logic 1** — A voltage level approximately equal to the input power voltage ($V_{DD}$).

**logic 0** — A voltage level approximately equal to the ground voltage ($V_{SS}$).

**low byte** — The least significant eight bits of a word.

**low voltage inhibit module (LVI)** — A module in the M68HC08 Family that monitors power supply voltage.

**LVI** — See "low voltage inhibit module (LVI)."

**M68HC08** — A Freescale family of 8-bit MCUs.

**mark/space** — The logic 1/logic 0 convention used in formatting data in serial communication.

**mask** — 1. A logic circuit that forces a bit or group of bits to a desired state. 2. A photomask used in integrated circuit fabrication to transfer an image onto silicon.

**mask option** — A optional microcontroller feature that the customer chooses to enable or disable.