

# Chapter 7

## Monitor ROM (MON)

### 7.1 Introduction

This section describes the monitor ROM (MON) and the monitor mode entry methods. The monitor ROM allows complete testing of the MCU through a single-wire interface with a host computer. This mode is also used for programming and erasing of FLASH memory in the MCU. Monitor mode entry can be achieved without use of the higher test voltage,  $V_{TST}$ , as long as vector addresses \$FFFE and \$FFFF are blank, thus reducing the hardware requirements for in-circuit programming.

### 7.2 Features

Features of the monitor ROM include the following:

- Normal user-mode pin functionality
- One pin dedicated to serial communication between monitor ROM and host computer
- Standard mark/space non-return-to-zero (NRZ) communication with host computer
- Execution of code in RAM or FLASH
- FLASH memory security feature<sup>(1)</sup>
- FLASH memory programming interface
- 959 bytes monitor ROM code size
- Monitor mode entry without high voltage,  $V_{TST}$ , if reset vector is blank (\$FFFE and \$FFFF contain \$FF)
- Standard monitor mode entry if high voltage,  $V_{TST}$ , is applied to  $\overline{IRQ}$
- Resident routines for FLASH programming and EEPROM emulation

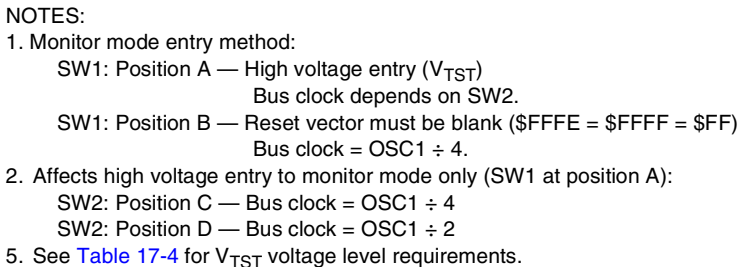
### 7.3 Functional Description

The monitor ROM receives and executes commands from a host computer. [Figure 7-1](#) shows a example circuit used to enter monitor mode and communicate with a host computer via a standard RS-232 interface.

Simple monitor commands can access any memory address. In monitor mode, the MCU can execute host-computer code in RAM while most MCU pins retain normal operating mode functions. All communication between the host computer and the MCU is through the PTB0 pin. A level-shifting and multiplexing interface is required between PTB0 and the host computer. PTB0 is used in a wired-OR configuration and requires a pull-up resistor.

---

1. No security feature is absolutely secure. However, Motorola's strategy is to make reading or copying the FLASH difficult for unauthorized users.



### Figure 7-1. Monitor Mode Circuit

### 7.3.1 Entering Monitor Mode

Table 7-1 shows the pin conditions for entering monitor mode. As specified in the table, monitor mode may be entered after a POR.

Communication at 9600 baud will be established provided one of the following sets of conditions is met:

1. If  $\overline{\text{IRQ}} = V_{\text{TST}}$ :
  - Clock on OSC1 is 4.9125MHz
  - PTB3 = low
2. If  $\overline{\text{IRQ}} = V_{\text{TST}}$ :
  - Clock on OSC1 is 9.8304MHz
  - PTB3 = high
3. If \$FFFE and \$FFFF are blank (contain \$FF):
  - Clock on OSC1 is 9.8304MHz
  - $\text{IRQ} = V_{\text{DD}}$

**Table 7-1. Monitor Mode Entry Requirements and Options**

$\overline{\text{IRQ}}$	\$FFFE and \$FFFF	PTB3	PTB2	PTB1	PTB0	OSC1 Clock <sup>(1)</sup>	Bus Frequency	Comments
$V_{\text{TST}}^{(2)}$	X	0	0	1	1	4.9152MHz	2.4576MHz	High voltage entry to monitor mode. 9600 baud communication on PTB0. COP disabled.
$V_{\text{TST}}^{(1)}$	X	1	0	1	1	9.8304MHz	2.4576MHz	
$V_{\text{DD}}$	BLANK (contain \$FF)	X	X	X	1	9.8304MHz	2.4576MHz	Blank reset vector (low-voltage) entry to monitor mode. 9600 baud communication on PTB0. COP disabled.
$V_{\text{DD}}$	NOT BLANK	X	X	X	X	X	$\text{OSC1} \div 4$	Enters User mode.

1. RC oscillator cannot be used for monitor mode; must use either external oscillator or XTAL oscillator circuit.

2. See Table 17-4 for  $V_{\text{TST}}$  voltage level requirements.

If  $V_{\text{TST}}$  is applied to  $\overline{\text{IRQ}}$  and PTB3 is low upon monitor mode entry (Table 7-1 condition set 1), the bus frequency is a divide-by-two of the clock input to OSC1. If PTB3 is high with  $V_{\text{TST}}$  applied to  $\overline{\text{IRQ}}$  upon monitor mode entry (Table 7-1 condition set 2), the bus frequency is a divide-by-four of the clock input to OSC1. Holding the PTB3 pin low when entering monitor mode causes a bypass of a divide-by-two stage at the oscillator *only if  $V_{\text{TST}}$  is applied to  $\overline{\text{IRQ}}$* . In this event, the OSCOUT frequency is equal to the 2OSCOUT frequency, and OSC1 input directly generates internal bus clocks. In this case, the OSC1 signal must have a 50% duty cycle at maximum bus frequency.

Entering monitor mode with  $V_{\text{TST}}$  on  $\overline{\text{IRQ}}$ , the COP is disabled as long as  $V_{\text{TST}}$  is applied to either  $\overline{\text{IRQ}}$  or  $\overline{\text{RST}}$ . (See Chapter 5 System Integration Module (SIM) for more information on modes of operation.)

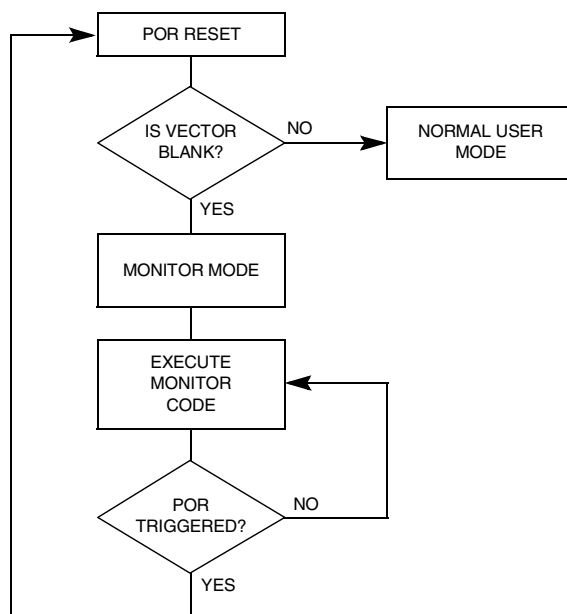
If entering monitor mode without high voltage on  $\overline{\text{IRQ}}$  and reset vector being blank (\$FFFE and \$FFFF) (Table 7-1 condition set 3, where applied voltage is  $V_{\text{DD}}$ ), then all port B pin requirements and conditions,

## Monitor ROM (MON)

including the PTB3 frequency divisor selection, are not in effect. This is to reduce circuit requirements when performing in-circuit programming.

Entering monitor mode with the reset vector being blank, the COP is always disabled regardless of the state of  $\overline{\text{IRQ}}$  or the  $\overline{\text{RST}}$ .

Figure 7-2. shows a simplified diagram of the monitor mode entry when the reset vector is blank and  $\overline{\text{IRQ}} = V_{\text{DD}}$ . An OSC1 frequency of 9.8304MHz is required for a baud rate of 9600.



**Figure 7-2. Low-Voltage Monitor Mode Entry Flowchart**

Enter monitor mode with the pin configuration shown above by pulling  $\overline{\text{RST}}$  low and then high. The rising edge of  $\overline{\text{RST}}$  latches monitor mode. Once monitor mode is latched, the values on the specified pins can change.

Once out of reset, the MCU waits for the host to send eight security bytes. (See [7.4 Security](#).) After the security bytes, the MCU sends a break signal (10 consecutive logic zeros) to the host, indicating that it is ready to receive a command. The break signal also provides a timing reference to allow the host to determine the necessary baud rate.

In monitor mode, the MCU uses different vectors for reset, SWI, and break interrupt. The alternate vectors are in the \$FE page instead of the \$FF page and allow code execution from the internal monitor firmware instead of user code.

Table 7-2 is a summary of the vector differences between user mode and monitor mode.

Table 7-2. Monitor Mode Vector Differences

Modes	Functions						
	COP	Reset Vector High	Reset Vector Low	Break Vector High	Break Vector Low	SWI Vector High	SWI Vector Low
User	Enabled	\$FFFE	\$FFFF	\$FFFC	\$FFFD	\$FFFC	\$FFFD
Monitor	Disabled <sup>(1)</sup>	\$FEFE	\$FEFF	\$FEFC	\$FEFD	\$FEFC	\$FEFD
Notes: 1. If the high voltage ( $V_{TST}$ ) is removed from the $\overline{IRQ}$ pin or the $\overline{RST}$ pin, the SIM asserts its COP enable output. The COP is a mask option enabled or disabled by the COPD bit in the configuration register.							

When the host computer has completed downloading code into the MCU RAM, the host then sends a RUN command, which executes an RTI, which sends control to the address on the stack pointer.

### 7.3.2 Baud Rate

The communication baud rate is dependant on oscillator frequency. The state of PTB3 also affects baud rate if entry to monitor mode is by  $\overline{IRQ} = V_{TST}$ . When PTB3 is high, the divide by ratio is 1024. If the PTB3 pin is at logic zero upon entry into monitor mode, the divide by ratio is 512.

Table 7-3. Monitor Baud Rate Selection

Monitor Mode Entry By:	OSC1 Clock Frequency	PTB3	Baud Rate
$\overline{IRQ} = V_{TST}$	4.9152 MHz	0	9600 bps
	9.8304 MHz	1	9600 bps
	4.9152 MHz	1	4800 bps
Blank reset vector, $\overline{IRQ} = V_{DD}$	9.8304 MHz	X	9600 bps
	4.9152 MHz	X	4800 bps

### 7.3.3 Data Format

Communication with the monitor ROM is in standard non-return-to-zero (NRZ) mark/space data format. (See [Figure 7-3](#) and [Figure 7-4](#).)

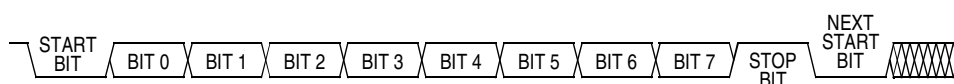


Figure 7-3. Monitor Data Format

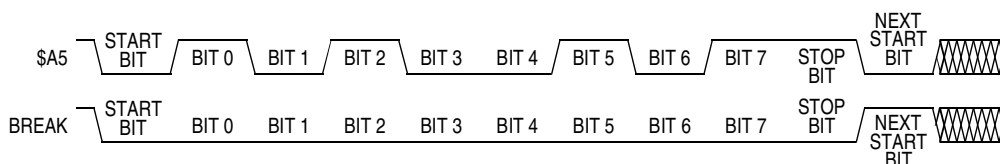


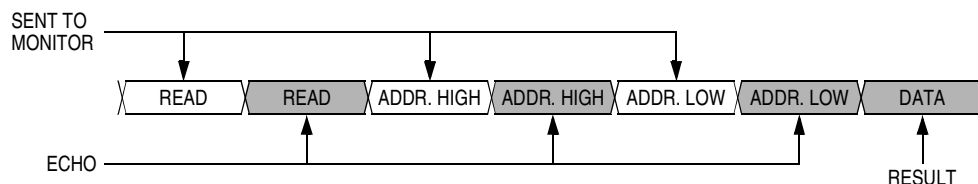
Figure 7-4. Sample Monitor Waveforms

## Monitor ROM (MON)

The data transmit and receive rate can be anywhere from 4800 baud to 28.8k-baud. Transmit and receive baud rates must be identical.

### 7.3.4 Echoing

As shown in [Figure 7-5](#), the monitor ROM immediately echoes each received byte back to the PTB0 pin for error checking.

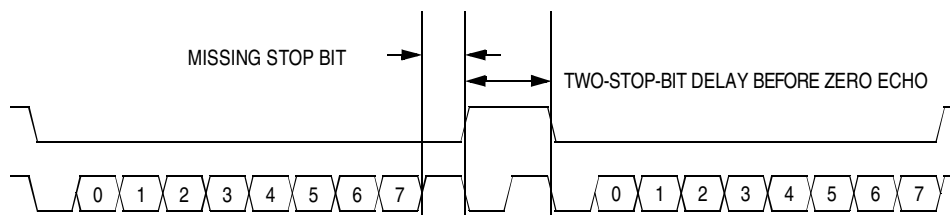


**Figure 7-5. Read Transaction**

Any result of a command appears after the echo of the last byte of the command.

### 7.3.5 Break Signal

A start bit followed by nine low bits is a break signal. (See [Figure 7-6](#).) When the monitor receives a break signal, it drives the PTB0 pin high for the duration of two bits before echoing the break signal.



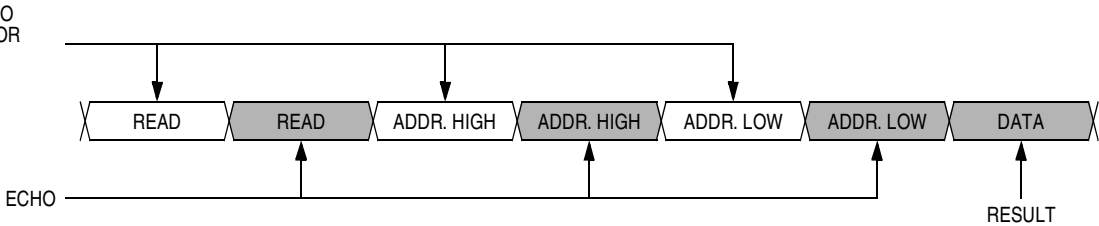
**Figure 7-6. Break Transaction**

### 7.3.6 Commands

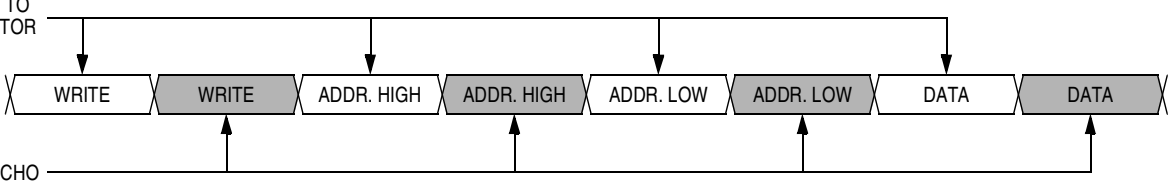
The monitor ROM uses the following commands:

- READ (read memory)
- WRITE (write memory)
- IREAD (indexed read)
- IWRITE (indexed write)
- READSP (read stack pointer)
- RUN (run user program)

**Table 7-4. READ (Read Memory) Command**

Description	Read byte from memory
Operand	Specifies 2-byte address in high byte:low byte order
Data Returned	Returns contents of specified address
Opcode	\$4A
<b>Command Sequence</b>  	

**Table 7-5. WRITE (Write Memory) Command**

Description	Write byte to memory
Operand	Specifies 2-byte address in high byte:low byte order; low byte followed by data byte
Data Returned	None
Opcode	\$49
<b>Command Sequence</b>  	

**Table 7-6. IREAD (Indexed Read) Command**

Description	Read next 2 bytes in memory from last address accessed
Operand	Specifies 2-byte address in high byte:low byte order
Data Returned	Returns contents of next two addresses
Opcode	\$1A
<p>Command Sequence</p> <pre> graph LR     subgraph Sequence         direction LR         I1[IREAD]         I2[IREAD]         D1[DATA]         D2[DATA]     end     SM[SENT TO MONITOR] --&gt; I1     E[ECHO] --&gt; I2     R[RESULT] --&gt; D2 </pre>	

**Table 7-7. IWRITE (Indexed Write) Command**

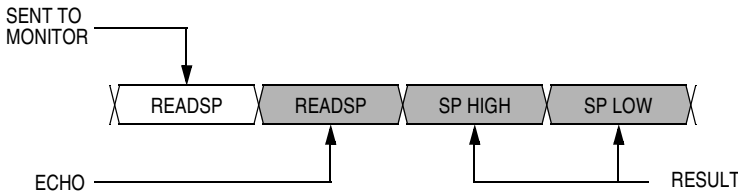
Description	Write to last address accessed + 1
Operand	Specifies single data byte
Data Returned	None
Opcode	\$19
<p>Command Sequence</p> <pre> graph LR     subgraph Sequence         direction LR         I1[IWRITE]         I2[IWRITE]         D1[DATA]         D2[DATA]     end     SM[SENT TO MONITOR] --&gt; I1     E[ECHO] --&gt; I2     D[DATA] --&gt; D1 </pre>	

**NOTE**

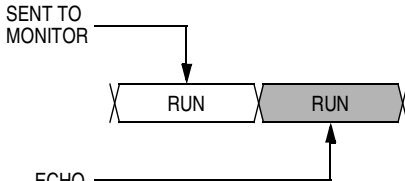
*A sequence of IREAD or IWRITE commands can sequentially access a block of memory over the full 64-Kbyte memory map.*



**Table 7-8. READSP (Read Stack Pointer) Command**

Description	Reads stack pointer
Operand	None
Data Returned	Returns stack pointer in high byte:low byte order
Opcode	\$0C
<b>Command Sequence</b>  	

**Table 7-9. RUN (Run User Program) Command**

Description	Executes RTI instruction
Operand	None
Data Returned	None
Opcode	\$28
<b>Command Sequence</b>  	

## 7.4 Security

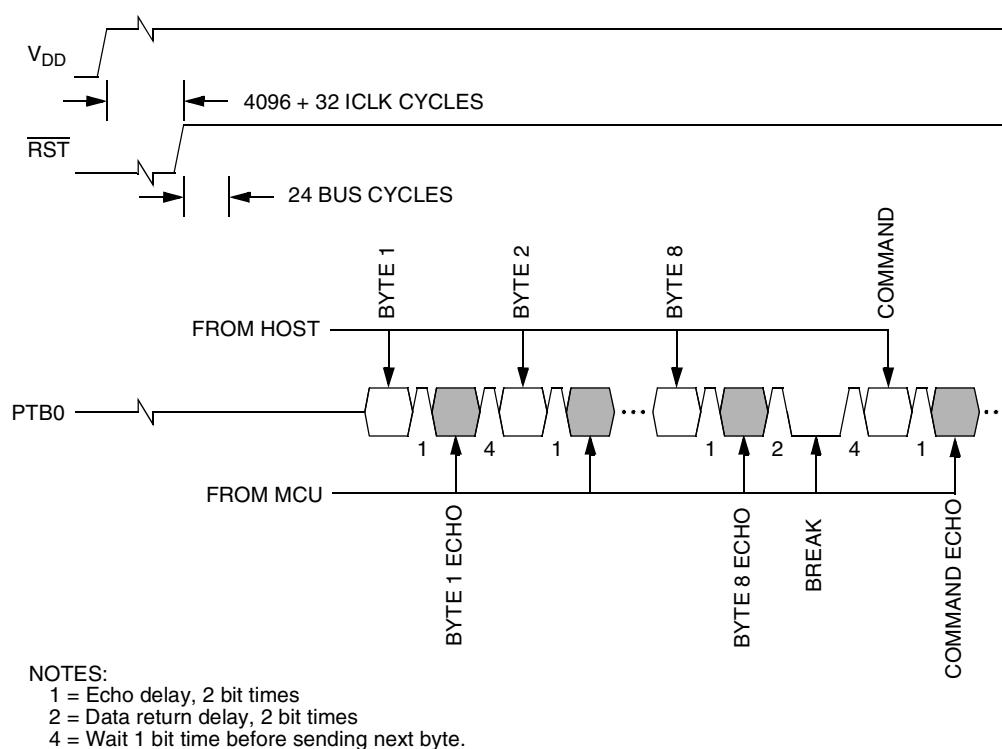
A security feature discourages unauthorized reading of FLASH locations while in monitor mode. The host can bypass the security feature at monitor mode entry by sending eight security bytes that match the bytes at locations \$FFF6–\$FFFD. Locations \$FFF6–\$FFFD contain user-defined data.

### NOTE

*Do not leave locations \$FFF6–\$FFFD blank. For security reasons, program locations \$FFF6–\$FFFD even if they are not used for vectors.*

During monitor mode entry, the MCU waits after the power-on reset for the host to send the eight security bytes on pin PTB0. If the received bytes match those at locations \$FFF6–\$FFFD, the host bypasses the security feature and can read all FLASH locations and execute code from FLASH. Security remains bypassed until a power-on reset occurs. If the reset was not a power-on reset, security remains bypassed and security code entry is not required. (See [Figure 7-7](#).)

## Monitor ROM (MON)



**Figure 7-7. Monitor Mode Entry Timing**

Upon power-on reset, if the received bytes of the security code do not match the data at locations \$FFF6–\$FFFD, the host fails to bypass the security feature. The MCU remains in monitor mode, but reading a FLASH location returns an invalid value and trying to execute code from FLASH causes an illegal address reset. After receiving the eight security bytes from the host, the MCU transmits a break character, signifying that it is ready to receive a command.

### NOTE

*The MCU does not transmit a break character until after the host sends the eight security bytes.*

To determine whether the security code entered is correct, check to see if bit 6 of RAM address \$60 is set. If it is, then the correct security code has been entered and FLASH can be accessed.

If the security sequence fails, the device should be reset by a power-on reset and brought up in monitor mode to attempt another entry. After failing the security sequence, the FLASH module can also be mass erased by executing an erase routine that was downloaded into internal RAM. The mass erase operation clears the security code locations so that all eight security bytes become \$FF (blank).

## 7.5 ROM-Resident Routines

Eight routines stored in the monitor ROM area (thus ROM-resident) are provided for FLASH memory manipulation. Six of the eight routines are intended to simplify FLASH program, erase, and load operations. The other two routines are intended to simplify the use of the FLASH memory as EEPROM. [Table 7-10](#) shows a summary of the ROM-resident routines.

Table 7-10. Summary of ROM-Resident Routines

Routine Name	Routine Description	Call Address	Stack Used <sup>(1)</sup> (bytes)
<b>PRGRNGE</b>	Program a range of locations	\$FC06	15
<b>ERARNGE</b>	Erase a page or the entire array	\$FCBE	9
<b>LDRNGE</b>	Loads data from a range of locations	\$FF30	9
<b>MON_PRGRNGE</b>	Program a range of locations in monitor mode	\$FF28	17
<b>MON_ERARNGE</b>	Erase a page or the entire array in monitor mode	\$FF2C	11
<b>MON_LDRNGE</b>	Loads data from a range of locations in monitor mode	\$FF24	11
<b>EE_WRITE</b>	Emulated EEPROM write. Data size ranges from 2 to 15 bytes at a time.	\$FD3F	24
<b>EE_READ</b>	Emulated EEPROM read. Data size ranges from 2 to 15 bytes at a time.	\$FDD0	16

1. The listed stack size excludes the 2 bytes used by the calling instruction, JSR.

The routines are designed to be called as stand-alone subroutines in the user program or monitor mode. The parameters that are passed to a routine are in the form of a contiguous data block, stored in RAM. The index register (H:X) is loaded with the address of the first byte of the data block (acting as a pointer), and the subroutine is called (JSR). Using the start address as a pointer, multiple data blocks can be used, any area of RAM can be used. A data block has the control and data bytes in a defined order, as shown in Figure 7-8.

During the software execution, it does not consume any dedicated RAM location, the run-time heap will extend the system stack, all other RAM location will not be affected.

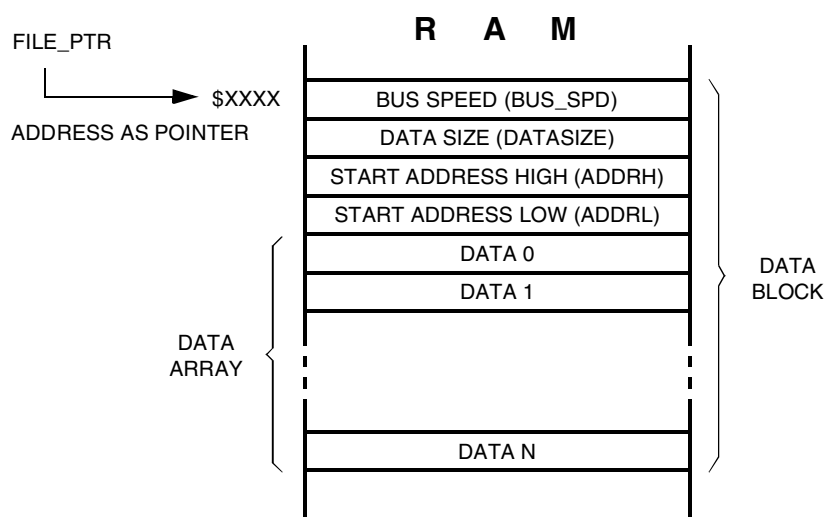


Figure 7-8. Data Block Format for ROM-Resident Routines

## Monitor ROM (MON)

The control and data bytes are described below.

- **Bus speed** — This one byte indicates the operating bus speed of the MCU. The value of this byte should be equal to 4 times the bus speed, and should not be set to less than 4 (i.e. minimum bus speed is 1 MHz).
- **Data size** — This one byte indicates the number of bytes in the data array that are to be manipulated. The maximum data array size is 128. Routines EE\_WRITE and EE\_READ are restricted to manipulate a data array between 2 to 15 bytes. Whereas routines ERARNGE and MON\_ERARNGE do not manipulate a data array, thus, this data size byte has no meaning.
- **Start address** — These two bytes, high byte followed by low byte, indicate the start address of the FLASH memory to be manipulated.
- **Data array** — This data array contains data that are to be manipulated. Data in this array are programmed to FLASH memory by the programming routines: PRGRNGE, MON\_PRGRNGE, EE\_WRITE. For the read routines: LDRNGE, MON\_LDRNGE, and EE\_READ, data is read from FLASH and stored in this array.

### 7.5.1 PRGRNGE

PRGRNGE is used to program a range of FLASH locations with data loaded into the data array.

**Table 7-11. PRGRNGE Routine**

<b>Routine Name</b>	PRGRNGE
<b>Routine Description</b>	Program a range of locations
<b>Calling Address</b>	\$FC06
<b>Stack Used</b>	15 bytes
<b>Data Block Format</b>	Bus speed (BUS_SPD) Data size (DATASIZE) Start address high (ADDRH) Start address (ADDRL) Data 1 (DATA1) : Data N (DATAN)

The start location of the FLASH to be programmed is specified by the address ADDRH:ADDRL and the number of bytes from this location is specified by DATASIZE. The maximum number of bytes that can be programmed in one routine call is 128 bytes (max. DATASIZE is 128).

ADDRH:ADDRL do not need to be at a page boundary, the routine handles any boundary misalignment during programming. A check to see that all bytes in the specified range are erased is not performed by this routine prior programming. Nor does this routine do a verification after programming, so there is no return confirmation that programming was successful. User must assure that the range specified is first erased.

The coding example below is to program 32 bytes of data starting at FLASH location \$EF00, with a bus speed of 4.9152 MHz. The coding assumes the data block is already loaded in RAM, with the address pointer, FILE\_PTR, pointing to the first byte of the data block.

```

                ORG     RAM
:
FILE_PTR:
BUS_SPD        DS.B    1; Indicates 4x bus frequency
DATASIZE       DS.B    1; Data size to be programmed
START_ADDR     DS.W    1; FLASH start address
DATAARRAY      DS.B    32; Reserved data array

PRGRNGE        EQU     $FC06
FLASH_START    EQU     $EF00

                ORG     FLASH
INITIALISATION:
    MOV     #20,      BUS_SPD
    MOV     #32,      DATASIZE
    LDHX    #FLASH_START
    STHX    START_ADDR
    RTS

MAIN:
    BSR     INITIALISATION
:
:
    LDHX    #FILE_PTR
    JSR     PRGRNGE

```

## 7.5.2 ERARNGE

ERARNGE is used to erase a range of locations in FLASH.

**Table 7-12. ERARNGE Routine**

<b>Routine Name</b>	ERARNGE
<b>Routine Description</b>	Erase a page or the entire array
<b>Calling Address</b>	\$FCBE
<b>Stack Used</b>	9 bytes
<b>Data Block Format</b>	Bus speed (BUS_SPD) Data size (DATASIZE) Starting address (ADDRH) Starting address (ADDRL)

There are two sizes of erase ranges: a page or the entire array. The ERARNGE will erase the page (64 consecutive bytes) in FLASH specified by the address ADDRH:ADDRL. This address can be any address within the page. Calling ERARNGE with ADDRH:ADDRL equal to \$FFFF will erase the entire FLASH array (mass erase). Therefore, care must be taken when calling this routine to prevent an accidental mass erase. To avoid undesirable routine return addresses after a mass erase, the ERARNGE routine should not be called from code executed from FLASH memory. Load the code into an area of RAM before calling the ERARNGE routine.

The ERARNGE routine do not use a data array. The DATASIZE byte is a dummy byte that is also not used.

## Monitor ROM (MON)

The coding example below is to perform a page erase, from \$EF00–\$EF3F. The Initialization subroutine is the same as the coding example for PRGRNGE (see [7.5.1 PRGRNGE](#)).

```
ERARNGE      EQU      $FCBE
MAIN:
    BSR      INITIALISATION
    :
    :
    LDHX     #FILE_PTR
    JSR      ERARNGE
    :
```

### 7.5.3 LDRNGE

LDRNGE is used to load the data array in RAM with data from a range of FLASH locations.

**Table 7-13. LDRNGE Routine**

<b>Routine Name</b>	LDRNGE
<b>Routine Description</b>	Loads data from a range of locations
<b>Calling Address</b>	\$FF30
<b>Stack Used</b>	9 bytes
<b>Data Block Format</b>	Bus speed (BUS_SPD) Data size (DATASIZE) Starting address (ADDRH) Starting address (ADDRL) Data 1 : Data N

The start location of FLASH from where data is retrieved is specified by the address ADDRH:ADDRL and the number of bytes from this location is specified by DATASIZE. The maximum number of bytes that can be retrieved in one routine call is 128 bytes. The data retrieved from FLASH is loaded into the data array in RAM. Previous data in the data array will be overwritten. User can use this routine to retrieve data from FLASH that was previously programmed.

The coding example below is to retrieve 32 bytes of data starting from \$EF00 in FLASH. The Initialization subroutine is the same as the coding example for PRGRNGE (see [7.5.1 PRGRNGE](#)).

```
LDRNGE      EQU      $FF30
MAIN:
    BSR      INITIALIZATION
    :
    :
    LDHX     #FILE_PTR
    JSR      LDRNGE
    :
```

### 7.5.4 MON\_PRGRNGE

In monitor mode, MON\_PRGRNGE is used to program a range of FLASH locations with data loaded into the data array.

**Table 7-14. MON\_PRGRNGE Routine**

<b>Routine Name</b>	MON_PRGRNGE
<b>Routine Description</b>	Program a range of locations, in monitor mode
<b>Calling Address</b>	\$FC28
<b>Stack Used</b>	17 bytes
<b>Data Block Format</b>	Bus speed Data size Starting address (high byte) Starting address (low byte) Data 1 : Data N

The MON\_PRGRNGE routine is designed to be used in monitor mode. It performs the same function as the PRGRNGE routine (see [7.5.1 PRGRNGE](#)), except that MON\_PRGRNGE returns to the main program via an SWI instruction. After a MON\_PRGRNGE call, the SWI instruction will return the control back to the monitor code.

### 7.5.5 MON\_ERARNGE

In monitor mode, ERARNGE is used to erase a range of locations in FLASH.

**Table 7-15. MON\_ERARNGE Routine**

<b>Routine Name</b>	MON_ERARNGE
<b>Routine Description</b>	Erase a page or the entire array, in monitor mode
<b>Calling Address</b>	\$FF2C
<b>Stack Used</b>	11 bytes
<b>Data Block Format</b>	Bus speed Data size Starting address (high byte) Starting address (low byte)

The MON\_ERARNGE routine is designed to be used in monitor mode. It performs the same function as the ERARNGE routine (see [7.5.2 ERARNGE](#)), except that MON\_ERARNGE returns to the main program via an SWI instruction. After a MON\_ERARNGE call, the SWI instruction will return the control back to the monitor code.

### 7.5.6 MON\_LDRNGE

In monitor mode, LDRNGE is used to load the data array in RAM with data from a range of FLASH locations.

**Table 7-16. ICP\_LDRNGE Routine**

<b>Routine Name</b>	MON_LDRNGE
<b>Routine Description</b>	Loads data from a range of locations, in monitor mode
<b>Calling Address</b>	\$FF24
<b>Stack Used</b>	11 bytes
<b>Data Block Format</b>	Bus speed Data size Starting address (high byte) Starting address (low byte) Data 1 : Data N

The MON\_LDRNGE routine is designed to be used in monitor mode. It performs the same function as the LDRNGE routine (see [7.5.3 LDRNGE](#)), except that MON\_LDRNGE returns to the main program via an SWI instruction. After a MON\_LDRNGE call, the SWI instruction will return the control back to the monitor code.

### 7.5.7 EE\_WRITE

EE\_WRITE is used to write a set of data from the data array to FLASH.

**Table 7-17. EE\_WRITE Routine**

<b>Routine Name</b>	EE_WRITE
<b>Routine Description</b>	Emulated EEPROM write. Data size ranges from 2 to 15 bytes at a time.
<b>Calling Address</b>	\$FD3F
<b>Stack Used</b>	24 bytes
<b>Data Block Format</b>	Bus speed (BUS_SPD) Data size (DATASIZE) <sup>(1)</sup> Starting address (ADDRH) <sup>(2)</sup> Starting address (ADDRL) <sup>(1)</sup> Data 1 : Data N

1. The minimum data size is 2 bytes. The maximum data size is 15 bytes.

2. The start address must be a page boundary start address: \$xx00, \$xx40, \$xx80, or \$00C0.

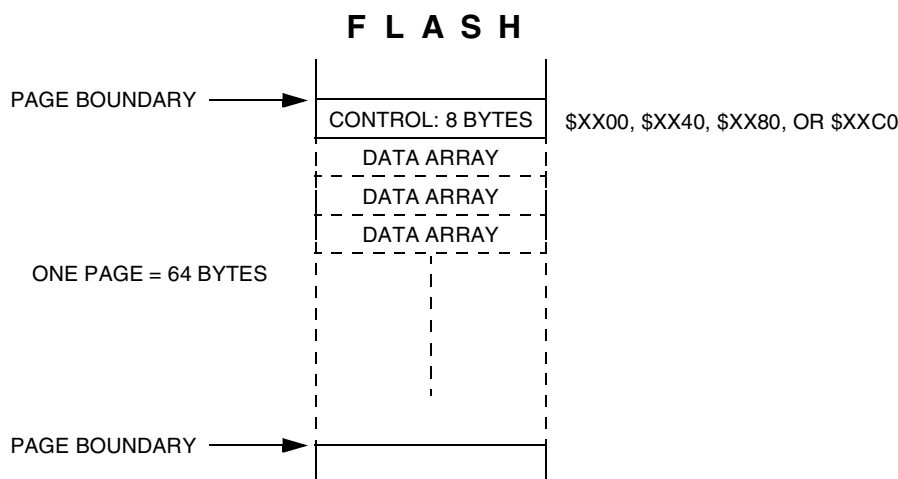
The start location of the FLASH to be programmed is specified by the address ADDRH:ADDRL and the number of bytes in the data array is specified by DATASIZE. The minimum number of bytes that can be



programmed in one routine call is 2 bytes, the maximum is 15 bytes. ADDR<sub>H</sub>:ADDR<sub>L</sub> must always be the start of boundary address (the page start address: \$XX00, \$XX40, \$XX80, or \$00C0) and DATASIZE must be the same size when accessing the same page.

In some applications, the user may want to repeatedly store and read a set of data from an area of non-volatile memory. This is easily possible when using an EEPROM array. As the write and erase operations can be executed on a byte basis. For FLASH memory, the minimum erase size is the page — 64 bytes per page for MC68HC908JL8. If the data array size is less than the page size, writing and erasing to the same page cannot fully utilize the page. Unused locations in the page will be wasted. The EE\_WRITE routine is designed to emulate the properties similar to the EEPROM. Allowing a more efficient use of the FLASH page for data storage.

When the user dedicates a page of FLASH for data storage, and the size of the data array defined, each call of the EE\_WRITE routine will automatically transfer the data in the data array (in RAM) to the next blank block of locations in the FLASH page. Once a page is filled up, the EE\_WRITE routine automatically erases the page, and starts to reuse the page again. In the 64-byte page, an 4-byte control block is used by the routine to monitor the utilization of the page. In effect, only 60 bytes are used for data storage. (see [Figure 7-9](#)). The page control operations are transparent to the user.



**Figure 7-9. EE\_WRITE FLASH Memory Usage**

When using this routine to store a 3-byte data array, the FLASH page can be programmed 20 times before an erase is required. In effect, the write/erase endurance is increased by 20 times. When a 15-byte data array is used, the write/erase endurance is increased by 5 times. Due to the FLASH page size limitation, the data array is limited from 2 bytes to 15 bytes.

The coding example below uses the \$EF00–\$EE3F page for data storage. The data array size is 15 bytes, and the bus speed is 4.9152 MHz. The coding assumes the data block is already loaded in RAM, with the address pointer, FILE\_PTR, pointing to the first byte of the data block.

## Monitor ROM (MON)

```

                ORG     RAM
:
FILE_PTR:
BUS_SPD        DS.B    1; Indicates 4x bus frequency
DATASIZE       DS.B    1; Data size to be programmed
START_ADDR     DS.W    1; FLASH starting address
DATAARRAY      DS.B    15; Reserved data array

EE_WRITE       EQU     $FD3F
FLASH_START    EQU     $EF00

                ORG     FLASH
INITIALISATION:
    MOV     #20,      BUS_SPD
    MOV     #15,      DATASIZE
    LDHX    #FLASH_START
    STHX    START_ADDR
    RTS

MAIN:
    BSR     INITIALISATION
:
:
    LHDX    #FILE_PTR
    JSR     EE_WRITE
```

### NOTE

*The EE\_WRITE routine is unable to check for incorrect data blocks, such as the FLASH page boundary address and data size. It is the responsibility of the user to ensure the starting address indicated in the data block is at the FLASH page boundary and the data size is 2 to 15. If the FLASH page is already programmed with a data array with a different size, the EE\_WRITE call will be ignored.*

### 7.5.8 EE\_READ

EE\_READ is used to load the data array in RAM with a set of data from FLASH.

**Table 7-18. EE\_READ Routine**

<b>Routine Name</b>	EE_READ
<b>Routine Description</b>	Emulated EEPROM read. Data size ranges from 2 to 15 bytes at a time.
<b>Calling Address</b>	\$FDD0
<b>Stack Used</b>	16 bytes
<b>Data Block Format</b>	Bus speed (BUS_SPD) Data size (DATASIZE) Starting address (ADDRH) <sup>(1)</sup> Starting address (ADDRL) <sup>(1)</sup> Data 1 : Data N

1. The start address must be a page boundary start address: \$xx00, \$xx40, \$xx80, or \$00C0.

The EE\_READ routine reads data stored by the EE\_WRITE routine. An EE\_READ call will retrieve the last data written to a FLASH page and loaded into the data array in RAM. Same as EE\_WRITE, the data size indicated by DATASIZE is 2 to 15, and the start address ADDRH:ADDRL must be the FLASH page boundary address.

The coding example below uses the data stored by the EE\_WRITE coding example (see [7.5.7 EE\\_WRITE](#)). It loads the 15-byte data set stored in the \$EF00–\$EE7F page to the data array in RAM. The initialization subroutine is the same as the coding example for EE\_WRITE (see [7.5.7 EE\\_WRITE](#)).

```
EE_READ      EQU      $FDD0
```

MAIN:

```
BSR      INITIALIZATION
:
:
LDHX     FILE_PTR
JSR      EE_READ
:
```

#### NOTE

*The EE\_READ routine is unable to check for incorrect data blocks, such as the FLASH page boundary address and data size. It is the responsibility of the user to ensure the starting address indicated in the data block is at the FLASH page boundary and the data size is 2 to 15. If the FLASH page is programmed with a data array with a different size, the EE\_READ call will be ignored.*

