AN2549
Rev. 1, 6/2005

Booting the MSC8101 Device Through Serial EPROM

By Barbara Johnson

Serial EPROMs offer advantages over parallel EPROMs because they minimize hardware requirements, thus reducing system costs. Serial EPROMs with the Inter-Integrated Circuit (I^2C) bus use a simple 2-pin interface for communication. On the other hand, parallel EPROMs use separate data, address and control lines. The high pin count on parallel EPROMs results in relatively large and costly packages that can add to the total system cost. Thus, serial EPROMs offer a cost-effective solution to non-volatile memory requirements.

To take advantage of this cost-effective solution, the Freescale MSC8101 device can boot from a serial EPROM in which program and data are downloaded from the serial EPROM to the MSC8101 via the I^2C bus. This application note introduces the fundamentals of I^2C communications and describes how to boot the MSC8101 from the serial EPROM.

CONTENTS

1 I^2C Basics

Philips Semiconductors developed the I^2C bus as a simple, low-cost communication link between integrated circuits (IC). The I^2C bus consists of two active bi-directional lines, the serial data (SDA) and serial clock (SCL) lines. It is a multi-master bus, so multiple ICs can be connected to the bus, each with its own unique address. Each device can act as a receiver and/or transmitter and operates as either a bus master or bus slave. The bus master initiates a data transfer to/from one or more slaves,

generates the clock signal on SCL, and terminates the transfer. The device addressed by the master is the slave. The SCL clock pulses for each bit shifted out on SDA. Data on SDA must be stable during the high portion of the SCL clock period. SDA can change only when SCL is low. **Figure 1** shows the relative timing of these two signals.

A bus master generates a start condition to change the bus from an idle state to a busy state to begin a transaction. A high-to-low transition on SDA while SCL is high indicates a start condition, which acts as a signal to all connected devices for incoming data.

The bus master sends a byte that consists of a 7-bit address and a read/write (R/W) command bit on SDA. Each device compares the transmitted address with its own address. If these addresses match, the device is addressed as a slave. If they do not match, the device ignores the rest of the transaction. The R/W bit indicates whether the master wishes to receive data (R/W = 1) or transmit data (R/W = 0). The selected slave device follows the address and read/write byte with an acknowledgment on the SDA line. When SDA is low, a positive acknowledgment indicates to the master that the slave is responding. However, when a slave does not acknowledge the slave address transmitted by the master, SDA remains high. The master assumes that the addressed slave is not responding and generates a stop condition to terminate the transfer. A stop condition is defined by a low-to-high transition on SDA while SCL is high.

After the master receives the acknowledge signal, it can start transmitting or receiving data. The master drives the data on SDA for a write operation, and the slave drives data on SDA for a read operation. Each data byte is followed by an acknowledgment. The acknowledgment is always generated by the device receiving data, that is, the slave for a write and the master for a read. When the master finishes sending or receiving data, it generates a stop condition and the bus returns to an idle state. All transactions begin with a start condition and end with a stop condition. **Figure 1** shows an example I^2C transaction.

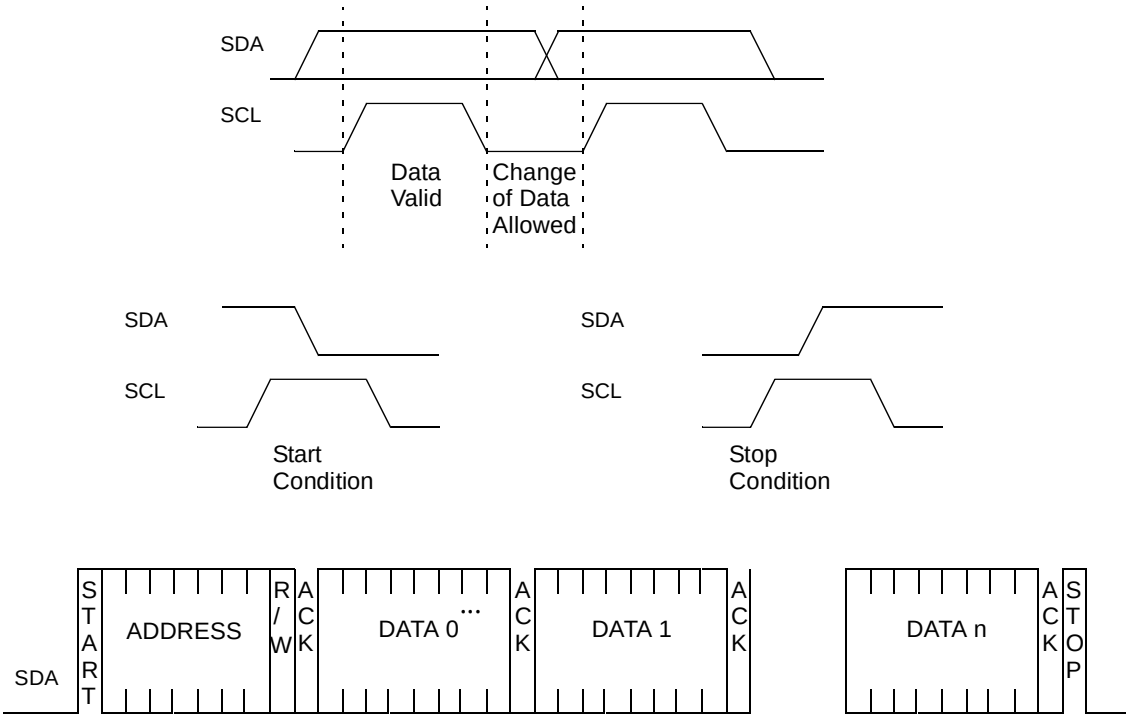

Figure 1. I^2C Timings

2 Boot Mode Configuration

The MSC8101 operating mode is configured by the external boot mode pins BTM[0–1], which are sampled on the rising edge of the power-on-reset $\overline{\text{PORESET}}$ signal. The bootloader program checks the logic state of these pins and jumps to the serial EPROM boot section in the ROM. **Table 1** shows the BTM[0–1] configuration for serial EPROM boot mode On the MSC8101ADS, the boot mode is configured by setting switch SW10[BTM0–BTM1] = OFF:ON.

Table 1. BTM[0–1] Configuration

BTM0	BTM1	Boot Mode
1	0	Boot through Serial EPROM

3 Serial EPROM Boot Procedure

After the MSC8101 bootloader program determines that the MSC8101 device is booting from the serial EPROM, it configures the I^2C serial interface registers and parameter RAM. It configures two GPIO pins, PB[18–19], as SCL and SDA, respectively, and enables the I^2C controller. The bootloader program divides the core frequency by 4000 to generate the SCL signal. For example, when the core frequency is 300 MHz, the bootloader program sets the SCL to 75 KHz, a frequency supported by most EPROMs. If the software watchdog disable bit HRCW[SWDIS] = 0 to indicate that the software watchdog is enabled, the bootloader program executes a sequence periodically to prevent the software watchdog counter from timing out and asserting the hardware reset during the bootloading process.

Next, the bootloader program determines where the boot program is stored in the EPROM. The EPROM address is calculated from the three internal space base (ISB) bits in the HRCW, which determine the base address of the internal memory space. The bootloader program reads the ISB bits and multiplies by two to calculate an address in the EPROM. This memory location contains a pointer to the location where the boot program is stored. For example, when the ISB = 011, the bootloader program reads the data in the EPROM at address $0x03 \times 2 = 0x06$. This address contains the 2-byte address of the boot code.

Each of the seven valid ISB settings has a corresponding 2-byte boot code address in the address table, as shown in **Table 2**. Memory locations 0x00 through 0x0F in the EPROM are reserved for the address table in which each master can access a different entry according to its ISB bits. The address table accommodates up to seven MSC8101 devices that can boot from the EPROM.

Table 2. Address Table

HRCW[ISB]	Internal Memory Space Base Address	Address of Pointer to Boot Program
000	0xF0000000	0x00
001	0xF0F00000	0x02
010	0xFF000000	0x04
011	0xFFF00000	0x06
100	Reserved	NA
101	0x00F00000	0x0A
110	0x0F000000	0x0C
111	0x0FF00000	0x0E

In this example, the code should load from 0x0010 for all valid ISB settings, and the address table should be programmed so that the corresponding address table entry for the ISB setting points to 0x0010. All entries in the table point to 0x0010, as shown in **Figure 2**.

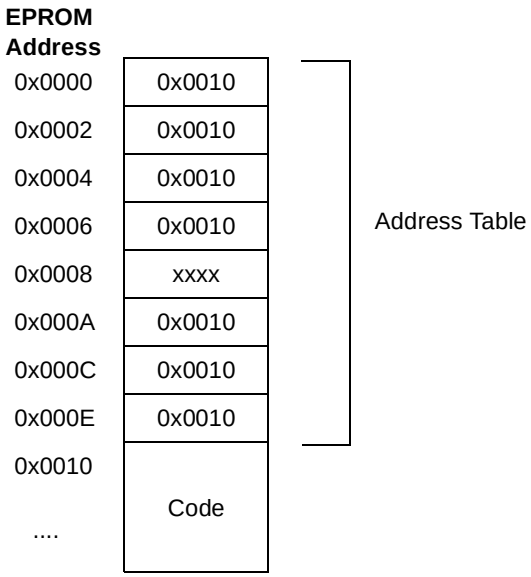

Figure 2. Address Table Entries

After the bootloader program determines where the boot code resides in the EPROM, it reads the first 64 bits extracts the block size, the checksum enable bit, the MSC8101 address where the code block is to be loaded, and the address of the next block in the EPROM.

The code downloaded from the EPROM to the MSC8101 is organized into code blocks, each residing in a buffer. The bootloader program allocates a buffer for the first code block according to the address and size given in the first 64 bits and creates a buffer descriptor (BD) according to these parameters. The bootloader polls the BDs when a code block is received. It does not use interrupts. When the BD is ready, the bootloader starts to read the code block from the EPROM. If the checksum is enabled, the bootloader calculates a checksum on the block and compares the calculated checksum to the loaded checksum. If the checksum is correct, the bootloader continues to read the next block. If the checksum comparison fails, the bootloader tries to read the block again. If the second attempt also fails, then the bootloader aborts and stops the boot process.

If the checksum is disabled, the checksum comparison is skipped, and the bootloader continues loading the next code block after finishing the current block. A block size of zero indicates that the current block is an end block. When all code blocks are loaded, boot execution starts from the address specified in the end block. **Figure 3** shows the boot procedure.

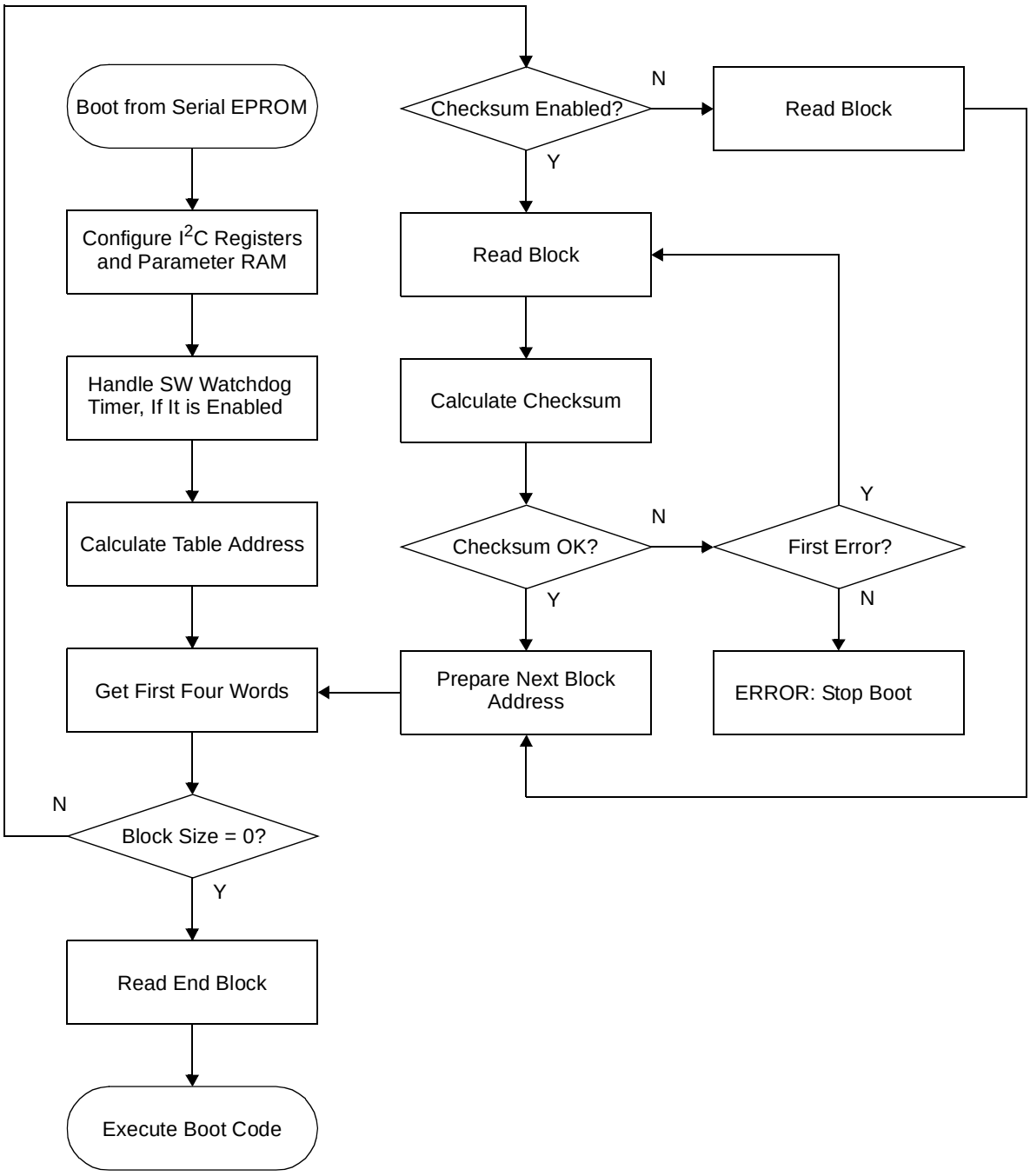

Figure 3. Serial EPROM Boot Load Procedure

4 Code Block Structure

The source program that is downloaded from the EPROM to the MSC8101 device can be organized into several code blocks (see **Figure 4**), each containing the following information:

- *Block size and checksum enable bit.* This 16-bit field defines the size of the block and determines whether checksum is to be performed on the block. The block size includes the checksum and $\overline{\text{checksum}}$ fields. If checksum is enabled, the most significant bit CSE is set. If checksum is disabled, CSE is cleared. The maximum block size is 64 KB.
- *Address of the next block.* This 16-bit field defines the EPROM address of the next block. If this field is 0x0000, the bootloader program assumes that the next block is in sequential order.
- *Address of source program in the current block.* Two 16-bit fields, the most significant part and the least significant part, determine where the source program in the block is to be loaded.
- *Source program.* This block must be in big-endian format, with the most significant part in the lower address.
- *Checksum for the current block.* This block contains the current checksum, which is a bit-wise XOR of the current word with the result of the XOR of the previous words in the block. To bypass the checksum comparison, clear the CSE.
- $\overline{\textit{Checksum}}$ *for the current block.* This field is the complement of the checksum block.

The end block indicates the end of code and assumes that at least one block of source code is loaded. The first two words are 0x0000 to indicate the end of the blocks.

- *Boot start address.* Two 16-bit fields, the most significant part and the least significant part, determine the location to which the MSC8101 device jumps at the end of the download. The next two words are 0x0000.
- *Checksum for the end block.* Contains the results of a bit-wise XOR of the current 16 bits with the result of the XOR of the previous 16 bits in the block.
- $\overline{\textit{Checksum}}$ *for the end block.* This field is the complement of the checksum field.

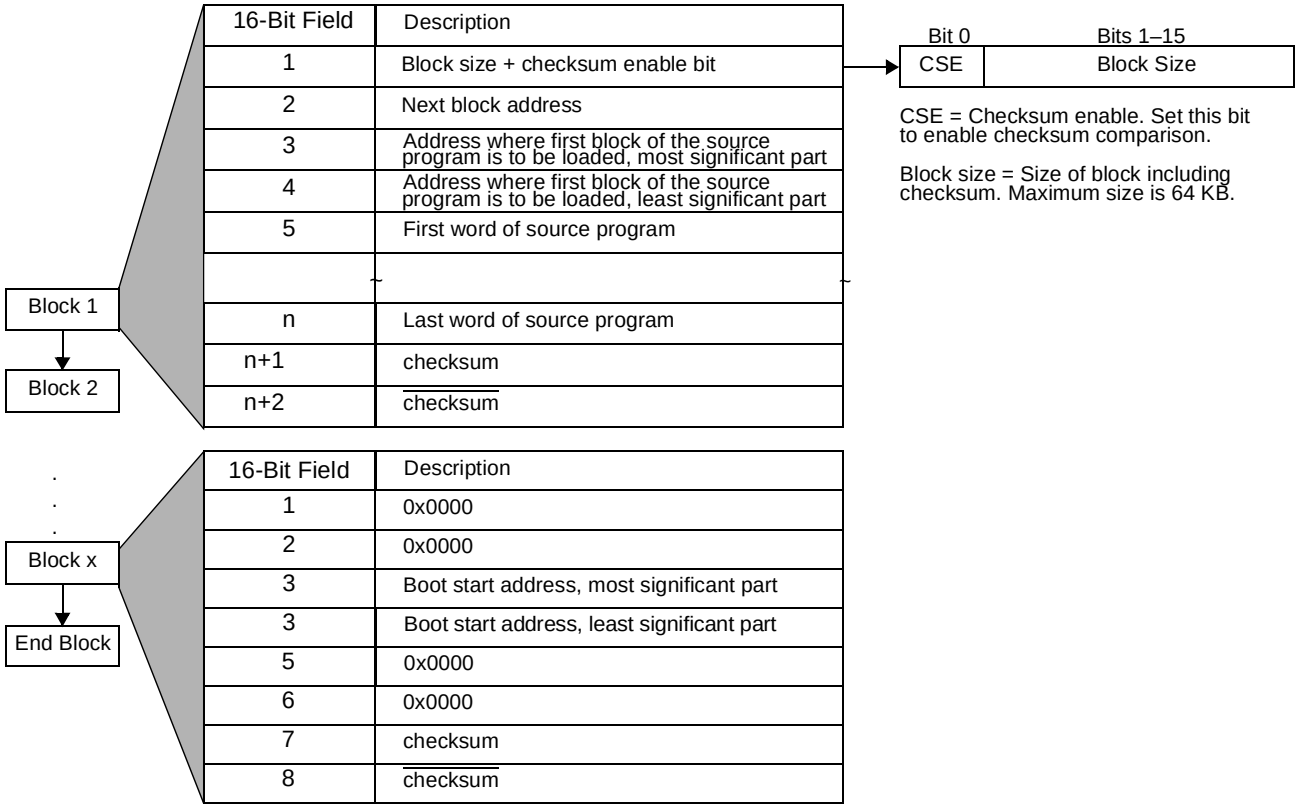

Figure 4. Block Structure

5 Example Source Program

Figure 5 shows an example source program in the serial EPROM. The opcodes shown to the left of the instructions can either be generated by a S-record utility or extracted from a program listing file. The code moves the data 0xAAAA1111, 0xBBBB1111, 0xCCCC1111, and 0xDDDD1111 to addresses 0x8000, 0x8004, 0x8008, and 0x800C, respectively. During the boot process, the source program words are downloaded from the serial EPROM to the MSC8101 device, starting at memory location 0x00001100.

The source program is divided into three source blocks. The first program block is 0x18 or 24 bytes long, and checksum comparison is enabled. The block size includes the source program and the two checksum values, so the first word is 0x8018. The second word is 0x0030, which indicates the address of the next block in the EPROM. This block is loaded at address 0x00001100 in the MSC8101, and this address is specified in the third and fourth words of the source block. Twenty bytes of the source program are followed by the checksum values 0x7532 and 0x8CAD for a total block size of 24 bytes.

The second source program block is also 24 bytes long, and checksum comparison is enabled as indicated in the first word with a value of 0x8018. The next block in the EPROM is located at address 0x0050. This block is loaded at address 0x00001114 in the MSC8101. Twenty bytes of source program are followed by the checksum values 0x6233 and 0x9DCC for a total block size of 24 bytes.

The third source program block is 0x30 or 48 bytes long, and checksum comparison is also enabled. The next block in the EPROM is located at address 0x0088. This block is loaded at address 0x00001128 in the MSC8101. The 44 bytes of source program are followed by the checksum values 0x8E35 and 0x71CA for a total block size of 48 bytes.

The end block specifies that no more source program blocks are to be sent. The first two words are zero to indicate that this block is the last block. After the program is downloaded, the MSC8101 jumps to address 0x00001100 to start program execution The next two words also have a value of zero. Finally, the last two words are the checksum values.

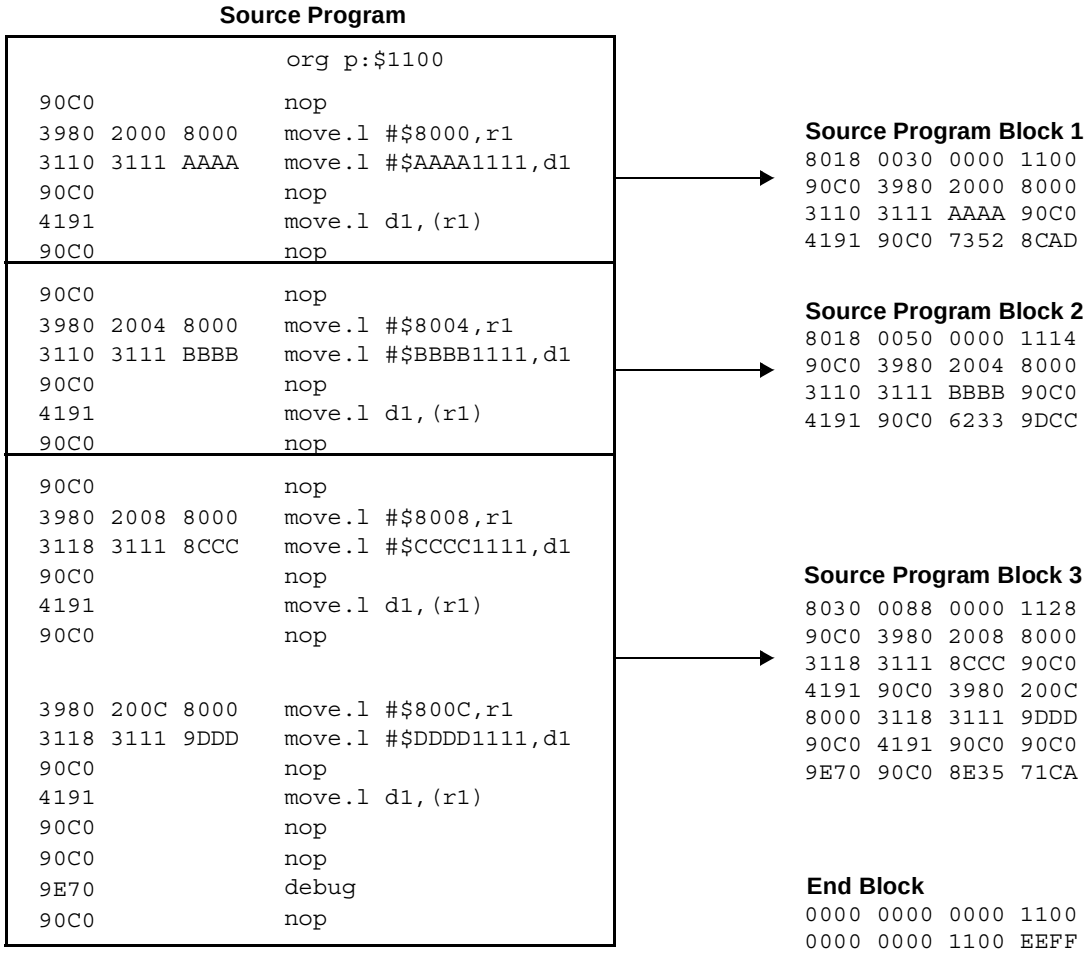

Figure 5. Source Program Blocks

5.1 Writing to the Serial EPROM

The source program blocks in **Section 5** define the data stored in the serial EPROM. However, writing to the serial EPROM requires additional addressing information that must be sent along with the data. In this example, writing to the AT24C128/256 serial EPROM requires sending the EPROM device address, followed by two 8-bit addresses and the data words.

The write operation is shown in **Figure 7**. A start condition, which is indicated by a logic 1 on the SDA line, enables the EPROM for a read or write operation. The start condition is then followed by the 8-bit device address which consists of a mandatory 1, 0, 1, 0 for the most significant part and followed by the device address bits and the read/write bit for the least significant part. The address bits allow multiple EPROM devices to be connected on the same bus. These bits are compared to their corresponding hardwired input pins. In this example, assume that the EPROM address pins A_2, A_1 and A_0 are hardwired to logic 1 which means that the address bits must also be set to logic 1. A read operation is initiated if the least significant bit of the device address is set while a write operation is initiated if this bit is cleared. Since we are writing to the EPROM to program the device, the R/W bit is cleared. The EPROM returns an acknowledge of logic 0 when the device address is compared.

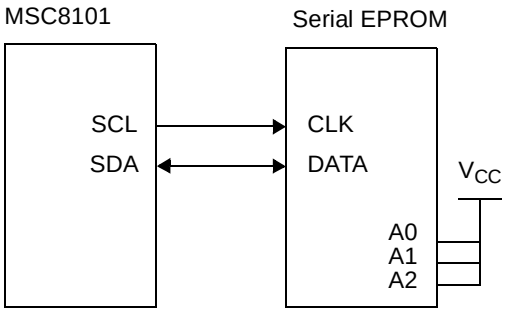

Figure 6. MSC8101 to EPROM Connection

Following the device address, a write operation requires two 8-bit addresses. The first 8-bit address is the most significant part of the EPROM address, and the second 8-bit address is the least significant part. These two bytes are concatenated to indicate the address in the EPROM where the data is stored. The EPROM outputs a logic 0 on the SDA line after the first and second byte addresses. The data byte follows the first and second byte addresses. The EPROM acknowledges receipt of the data byte with a logic 0 on the SDA line. The EPROM address is automatically incremented following receipt of each data byte. Up to 63 more data words can be written to the EPROM. If more than 64 data bytes are transmitted to the EPROM, the data byte address rolls over and previous data is overwritten. Therefore, to write more than 64 bytes, the device address and the first and second byte addresses must be reinitialized. The write sequence is terminated with a stop condition, which is a logic 1 on the SDA line after receiving an acknowledge that the last data byte was received.

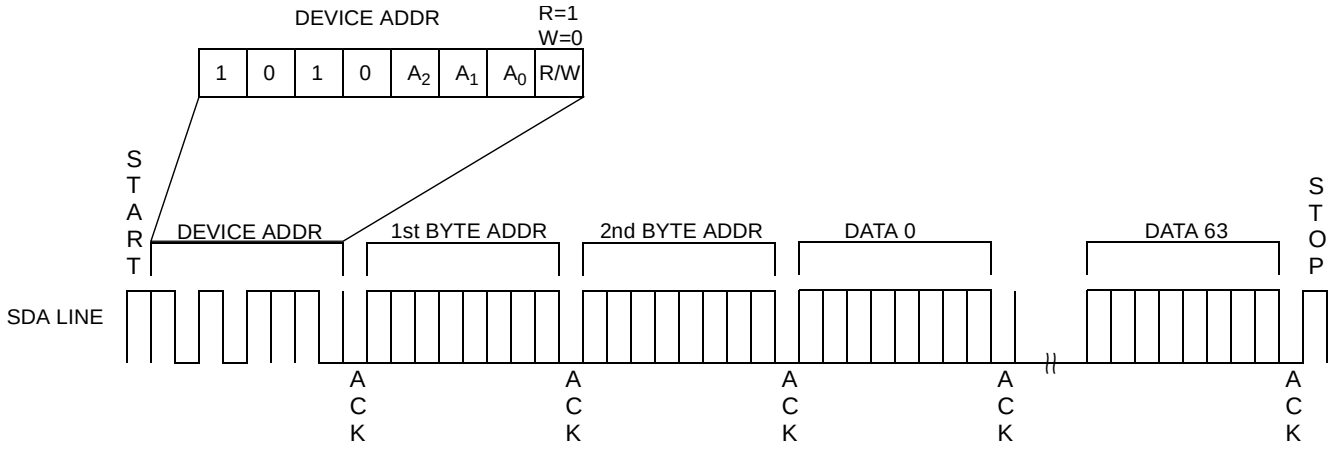

Figure 7. Write Operation

5.2 Example Write Sequence

Figure 8 shows the original source program blocks as defined in **Figure 5** and the additional addressing information required in a write sequence. The additional addressing information is shown in bold-face font. This example divides the source program blocks into five write sequences. In the first write sequence, the first byte written to the EPROM is the device address. Assuming that the address pins A_2, A_1 and A_0 are hardwired to logic 1, the device address is set to 0xAE for a write operation. The EPROM returns an acknowledge of logic 0 when the device address is received.

The next two bytes sent to the EPROM consist of the address where the data is to be stored. The EPROM returns an acknowledge on the SDA line after the first byte and also after the second byte. In this example, incoming data is stored starting at address 0x0000.

The next 16 bytes written to the EPROM are not part of the actual boot program but are entries to the address table. The address table consists of eight 2-byte entries that represent the starting address in the EPROM where the boot program is stored. Each entry corresponds to each of the eight possible combinations of HRCW[ISB] bits. In this example, all address table entries are 0x0010, so the boot program is stored in address 0x0010 in the EPROM. The actual boot code is stored starting at address 0x0010. A total of 32 bytes are written to the EPROM before the stop condition is sent. A value of 0xFF on the SDA indicates the end of transmission. In the first write sequence, the 32 bytes of code are stored at locations 0x0000 through 0x001F. The second write sequence writes to the next available address, which starts at 0x0020 in the EPROM. The next 32 bytes of code are written to this address. The process repeats for write sequences 3, 4, and 5.

You can program the serial EPROM on the MSC8101ADS by programming the MSC8101 I^2C controller to write the data shown in **Figure 5**. To verify that the serial EPROM has been programmed correctly, the I^2C controller can be configured to read the data from the serial EPROM. For programming details, refer to the chapter on the I^2C Controller in the *MSC8101 Reference Manual*.

EPROM Address		
	AE 0000	
0x0000	**0010 0010 0010 0010**	
0x0008	**0010 0010 0010 0010**	Write Sequence 1
0x0010	8018 0030 0000 1100	
0x0018	90C0 3980 2000 8000	
	FF	
	AE 0020	
0x0020	3110 3111 AAAA 90C0	
0x0028	4191 90C0 7352 8CAD	Write Sequence 2
0x0030	8018 0050 0000 1114	
0x0038	90C0 3980 2004 8000	
	FF	
	AE 0040	
0x0040	3110 3111 BBBB 90C0	
0x0048	4191 90C0 6233 9DCC	Write Sequence 3
0x0050	8030 0088 0000 1128	
0x0058	90C0 3980 2008 8000	
	FF	
	AE 0060	
0x0060	3118 3111 8CCC 90C0	
0x0068	4191 90C0 3980 200C	Write Sequence 4
0x0070	8000 3118 3111 9DDD	
0x0078	90C0 4191 90C0 90C0	
	FF	
	AE 0080	
0x0080	9E70 90C0 8E35 71CA	
0x0088	0000 0000 0000 1100	Write Sequence 5
0x0090	0000 0000 1100 EEFF	
	FFFF FFFF FFFF FFFF	
	FF	

Figure 8. Example Write Sequence

6 Bootloader Program

This section lists the REV A MSC8101 bootloader program for booting from a serial EPROM via I^2C.

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;RESERVED REGISTER:
; d5 : The address for read_bd function (Byte Shifted)
; d7 : The Rx Buffer Length for read_bd function
; d8 :
; d9 : IMMR - Set by previous boot_code
; d10: Block counter (count the number of loaded blocks)
; d12: Used for watchdog handler function.
; d14: Tx buffer address, for read_db function.
; r2 : Pointer to address of block data on EEPROM
; r5 : The Rx Buffer pointer for read_bd function
; r6 : IMMR + $0001_0000
; r7 : IMMR
;
;The follow registers were initiated by the previous code (boot_code_revA1)
; d13:Contain the ISB Bits
; d9 :IMMR
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; -------------------------------------------------------------
;; ----------------- SERIAL BOOT CODE STARTS HERE ----------------
;; -------------------------------------------------------------
from_serial
       move.l  d9,r7          ;d9 & r7 will have always IMMR value
       move.l  r7,d11
       move.l  #$10000,d12
       add     d11,d12,d11
       move.l  d11,r6         ;r6<-IMMR+$10000 for registers access

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; Configure I2 Register &
;; Parameter RAM
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
       move.l  d9,r3                  ;r3 <- IMMR
       bmset   #I2C_BASE,r3.l         ;r3 receive I2C page base addr
       ;sw watchdog
       move.l  #$0,d11                ;clr d11, sw watchdog not enabled
       move.l  (r6+$4),d12            ;get sypcr value
       bmtsts  #$0004,d12.l
       jf      init_
       nop
       move.l  d11,r15                ; clear watchdog handle counter
       move.l  #$1,d11                ; indicate sw watchdog enabled
       move.l  #$0100,r15             ; initialize sw watchdog handle counter
;Initiate I2C_BASE
       init_   move.w #I2C_BASE_VAL,d4
       nop                            ;insert due to restriction 6.4.4.a.2
       move.w  d4,(r3)                ;Save d4 into I2C_BASE
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; Initialize Parameter RAM
       move.l  #RTBASEVAL,r1          ;RBASE=3e50,TBASE=3e40
       nop                            ;insert due to restriction 6.4.4.a.2
       move.l  r1,(r7+RBASE)          ;$00
       move.l  #$1212ff00,r1          ;RFCR=12(use local BUS),TFCR=12,MRLBR=ff00
       nop                            ;insert due to restriction 6.4.4.a.2
       move.l  r1,(r7+RFCR)           ;$04
```

```
        move.l  #$00000000,r1           ;RSTATE
        nop                             ;insert due to restriction 6.4.4.a.2
        move.l  r1,(r7+RSTATE)          ;$08
        move.l  r1,(r7+RPTR)            ;$0c
        move.l  r1,(r7+RBPTR)           ;$10
        move.l  r1,(r7+RTEMP)           ;$14
        move.l  r1,(r7+TSTATE)          ;$18
        move.l  r1,(r7+TPTR)            ;$1c
        move.l  r1,(r7+TBPTR)           ;$20
        move.l  r1,(r7+TTEMP)           ;$24
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
        move.l  #$10000,d1
        nop                             ;insert due to restriction 6.4.4.a.2
        add     d1,d9,d4                ;d4 receive IMMR +1_0000
        nop                             ;insert due to restriction 6.4.4.a.2
        move.l  d4,r6                   ;R6 <-IMMR +1_0000
        ;Initiate I2C command - Page = 01010,code=01011
        move.l  #$29610000,r1
        nop
        move.l  r1,(r6+CPCR)            ;Write the cmd to CPCR
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;Save registers in the backup area of DP ram
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
        move.l  (r6+SCCR),r1
        nop
        move.l  r1,(r7+SCCRB)
        move.l  (r6+PODRB),r1
        nop
        move.l  r1,(r7+PODRBB)
        move.l(r6+PSORB),r1
        nop
        move.l  r1,(r7+PSORBB)
        move.l  (r6+PPARB),r1
        nop
        move.l  r1,(r7+PPARBB)
        move.l  (r6+PDIRB),r1
        nop
        move.l  r1,(r7+PDIRBB)
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;Initiate SCCR to 01 -> divide by 16
        move.l  #1,r1
        move.l  r1,(r6+SCCR)
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;Initiate IO Port
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;Configure port B SCL=PB18,SDA=PB19
        move.l  #$00003000,r1
        nop                                 ;insert due to restriction 6.4.4.a.2
        move.l  r1,(r6+PODRB)               ; PODRB[18,19]=1
        move.l  r1,(r6+PSORB)               ; PSORB[18,19]=1
        move.l  r1,(r6+PPARB)
        move.l  #$00000000,r1
        nop                                 ;insert due to restriction 6.4.4.a.2
        move.l  r1,(r6+PDIRB)
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;Initiate SCCR to 01 -> divide by 16
        move.l  #1,r1
        move.l  r1,(r6+SCCR)
```

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; Read first Block Address
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;D5 <- D5 x 2 + byte shift left
       asll    #$9,d5                          ;d5 <- Address(1 Byte shifted)
       move.l  #$2,d7                          ;d7 <- Rx BD length
       move.l  #TXB,d4                         ;prepare Tx buffer address in d14
       move.l  r5,d14                          ;r5 holds SRAM base address
       nop
       add     d4,d14,d14                      ;add sram base
       move.l  #RXB,r5                         ;r5 <-Rx Buffer Offset from SRAM
       move.l  r7,d4                           ;d4 <-SRAM base
       bsr     read_bd

;In the read_bd routine, the start of code address (2 bytes)
;was read to the RX Buffer at address RXB
       move.w  (RXB),d5                        ;d5 Receive the code start address
       bmclr   #$FFFF,d5.h                     ;clear unread upper bits
       move.l d5,d8                            ;Back d5 it into d8

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;Prepare for read First Block Header
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

       ;D5 <- D5 x 2 + byte shift left
       asll    #$8,d5                  ;d5 <- Address(1 Byte shifted)
       move.l  #$8,d7                  ;d7 <- Header lenght(4 words)

       ;;Calculate the address for Head at DPRAM
       move.l  #RXB,d2                 ;d2<- Block Header (Ofset from IMMR)
       nop                             ;inserted due to restriction 6.4.4.a.2

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; Get first 4 words
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
Get4FWord
       move.l  d2,r5                   ; r5<-Head Pointer (RXB)
       move.l  #$8,d7                  ; d7 <- Header lenght(4 words)
       bsr     read_bd                 ; Call read_bd to read the first 4 words

;; Read the Block Header Data
       move.l  r5,d6                   ; d6<-Head Pointer Offset
       nop                             ;inserted due to restriction 6.4.4.a.2
       add     d4,d6,d6                ; d6<-Head Pointer Offset + RAMptr
       move.l  d6,r8                   ; r8<-Head Pointer Offset + RAMptr
       nop                             ;inserted due to restriction 6.4.4.a.2
       move.w  (r8),d7                 ;d7 <- CS Enabled bit | Block size
       extractu #$e,#0,d7,d7           ;remove the extend bits and CS

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;if Block size == 0 go to Boot_loaded (end)
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
       tsteq   d7                      ;compare d7 to 0
       jt      Boot_loaded             ;if d7 = 0->jmp to Boot Loaded
       nop                             ;inserted due to restriction 6.4.4.a.3

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; Read Block Data
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
       move.l  (r8+$4),r5              ;Address to save the data on DPRAM (offset
```

```
from
                                        ;SRAM)
        move.l  d8,d5                   ;d5<-Block data address on eeprom
        nop                             ;inserted due to restriction 6.4.4.a.2
        add     #$8,d5                  ;increment d5 so it points to data (first 8
bytes
                                        ;= Header)
        asll    #$8,d5                  ;Byte shift d5 (prepare for read_bd)

read_block
        move.l  #$0,d0                  ;Reset Status Bit in d0

read_block2
        move.l  r7,d4
        bsr     read_bd
        move.l  d2,r4                   ;r4<-Head Pointer
        nop                             ;inserted due to restriction 6.4.4.a.2
        move.w  (r4),d8                 ;d8<- CSE | Block size

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;Checksum Enabled ????
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

        bmtsts  #$8000,d8.l             ; Test if CS bit is set
        jt      calc_cs                 ; if checksum enable jump to calc CS
        nop                             ; insert due to restriction 6.4.4.a.3

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; Prepare Next Block Address
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

PrepareNextBAdd
        add     #$1,d10                 ;increment the BD counter
        move.w  (r4+$2),d8              ;d8<- Next Block Address
        move.l  d8,d5                   ;Restore address from d8
        asll    #$8,d5                  ;d5 <- Address for next Block (1 Byte
shifted)
        jmp     Get4FWord               ;jmp to Get 4 First words
        nop

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;Start Boot Execution
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

Boot_loaded
        move.l  #$80,d0                 ;Set Boot Loaded flag on d0
        move.l  (r8+$4),r3              ;r3 receive Boot Start Address

;;Reset all I2C Parameter to their default value
        move.l  #$0,r1
        move.b  r1,(r6+I2MOD)           ;Clear i2mod
        move.b  r1,(r6+I2ADD)           ;Slave Address= 00 (optional)
        move.b  r1,(r6+I2CMR)           ;Disable All Interrupt
        move.b  r1,(r6+I2BRG)           ;I2BRG[DIV]=6
        move.b  r1,(r6+I2COM)           ;I2COM[M/s]=1 (Master mode)
        move.w  #$0017,r1
        move.b  r1,(r6+I2CER)           ;Clear all previous events

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;Reload saved registers values from the backup area of DP ram
```

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
        move.l d9,r7
        nop
        move.l (r7+SCCRB),r1
        nop
        move.l r1,(r6+SCCR)
        move.l (r7+PODRBB),r1
        nop
        move.l r1,(r6+PODRB)
        move.l (r7+PSORBB),r1
        nop
        move.l r1,(r6+PSORB)
        move.l (r7+PPARBB),r1
        nop
        move.l r1,(r6+PPARB)
        move.l (r7+PDIRBB),r1
        nop
        move.l r1,(r6+PDIRB)
        jmp    end_

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;; END OF MAIN CODE FLOW;;;;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
```

How to Reach Us:

Home Page:
www.freescale.com

E-mail:
support@freescale.com

USA/Europe or Locations not listed:
Freescale Semiconductor
Technical Information Center, CH370
1300 N. Alma School Road
Chandler, Arizona 85224
+1-800-521-6274 or +1-480-768-2130
support@freescale.com

Europe, Middle East, and Africa:
Freescale Halbleiter Deutschland GMBH
Technical Information Center
Schatzbogen 7
81829 München, Germany
+44 1296 380 456 (English)
+46 8 52200080 (English)
+49 89 92103 559 (German)
+33 1 69 35 48 48 (French)
support@freescale.com

Japan:
Freescale Semiconductor Japan Ltd.
Headquarters
ARCO Tower 15F
1-8-1, Shimo-Meguro, Meguro-ku,
Tokyo 153-0064, Japan
0120 191014 or +81 3 5437 9125
support.japan@freescale.com

Asia/Pacific:
Freescale Semiconductor Hong Kong Ltd.
Technical Information Center
2 Dai King Street
Tai Po Industrial Estate
Tai Po, N.T. Hong Kong
+800 2666 8080

For Literature Requests Only:
Freescale Semiconductor Literature Distribution Center
P.O. Box 5405
Denver, Colorado 80217
1-800-441-2447 or 303-675-2140
Fax: 303-675-2150
LDCForFreescaleSemiconductor@hibbertgroup.com

Document Order No.: AN2549
Rev. 1
6/2005