Interfacing Serial EEPROM To DSP563xx

by

Ilan Naslavsky
Leonid Smolyansky

Motorola, Incorporated
Semiconductor Products Sector
6501 William Cannon Drive West
Austin, TX 78735-8598

Order this document by: APR38/D

CONTENTS

4 ESSI and SCI Configuration

5 Customization

Appendix A Assembly Equates

Appendix B Assembly Equates

Figures

Tables

Examples

1 Introduction

This application report describes how to interface Serial Electrically Erasable Programmable Memory (SEEPROM) devices with DSP56300 Family chips through either the Enhanced Synchronous Serial Interface (ESSI) or the Synchronous Communication Interface (SCI) of the DSP563xx chip. The ESSI and SCI are available in several derivatives of the DSP56300 family of microprocessors. See **Appendix B**.

The DSP56300 Family's ESSI and SCI are fully capable of interfacing to SEEPROM devices through a Serial Peripheral Interface (SPI) bus using the following family features:

- SPI industry-standard bus connection support through ESSI or SCI
- Application Program Interface (API) support for:
  - Read data dlock
  - Write data block
  - Write protection management
- Full serial clock rate support (up to 2MHz)

1.1 Scope

This application report describes the connection of DSP56300 Family devices to industry standard SPI-compatible SEEPROMs, such as SGS-THOMSON's ST95010/020/040 or National's NM25C020. It is recommended for the developer who has previous knowledge of Motorola's DSP56300 family, as well as the specification of the selected Serial EEPROM.

Section 2 describes the physical connection between the serial interface, ESSI or SCI, and a SEEPROM. **Section 3** details the implemented system conception. **Section 4** explains the configuration of ESSI and SCI registers; **Section 5** makes recommendations on system and code customization.

Appendix A lists the application's assembly equates; **Appendix B** lists relevant reference information.

1.2 Serial EEPROM Versus Parallel EEPROM

In comparison to Parallel EEPROMs, Serial EEPROMs have several advantages:

- Serial EEPROMs are cheaper.
- Serial EEPROMs are smaller and take up less area on the application board.
- Serial EEPROMs require fewer connection lines.

1.3 Application Example

Figure 1-1illustrates the use of SEEPROMs with DSP56300 Family devices. Here, a DSP56301 chip connects to a Peripheral Component Interconnect (PCI) bus through the HI32 Host Interface and to a SEEPROM through ESSI or SCI. The SEEPROM is used for downloading configuration data for HI32 and for storing run-time parameters that should be saved on non-volatile storage.

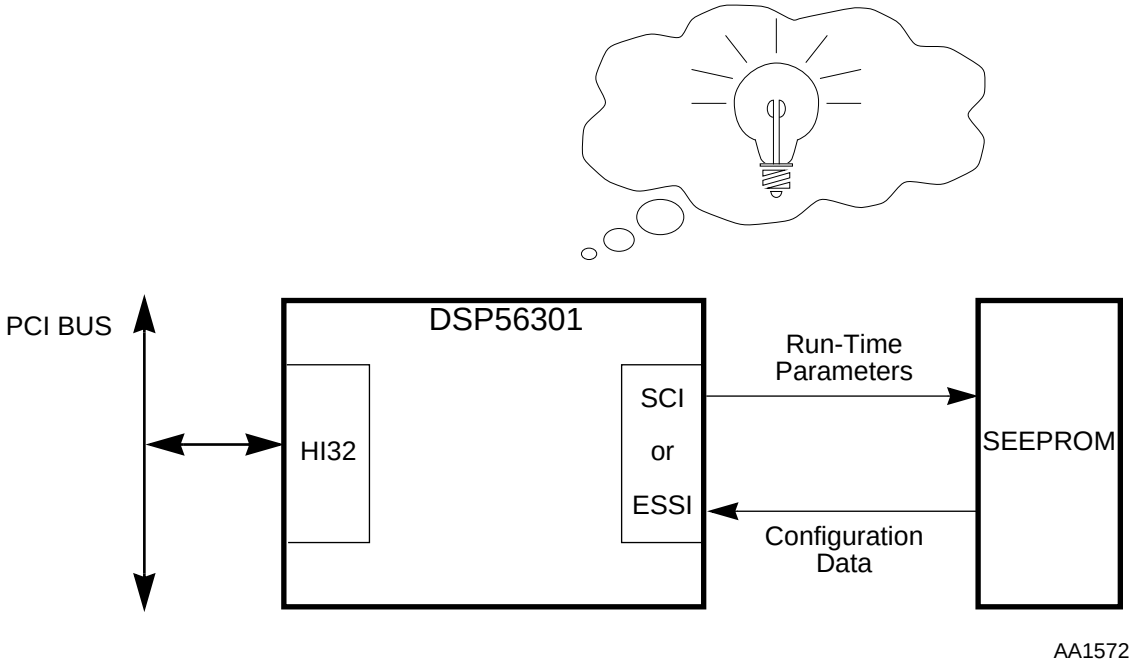

Figure 1-1 Application Example

1.4 Serial EEPROM / DSP Clock Ratio

Due to ESSI timing considerations, as explained in **Section 3.6**, the ratio between the Serial EEPROM Clock and DSP internal clock is limited to a minimum of 40:1.

$$\frac{\text{Serial EEPROM CLOCK}}{\text{DSP CLOCK}^{(\text{ESSI})}} > 40$$

For the SCI, the ratio is limited by specification to a minimum of 8:1.

$$\frac{\text{Serial EEPROM CLOCK}}{\text{DSP CLOCK}^{(\text{SCI})}} > 8$$

2 Physical Connection

This section describes the physical connection between the ESSI or SCI and a generic Serial EEPROM. The DSP563xx/Serial EEPROM connection suggested in this application report uses three ESSI pins or three SCI pins and one Port A Address Attribute pin, AAx, to provide all the data and control functions available in marketed Serial EEPROMs.

2.1 Serial EEPROM Pinout

Most SPI-compatible Serial EEPROMs present the user with eight pins: four for the serial interface, two for auxiliary control, and two for supply voltage and ground.

This report refers to an imaginary device with just such a configuration; we use general pin names, not necessarily those used in real devices.

On most Serial EEPROMs, pins with different names can have the same function. **Table 2-1** briefly describes each pin function and the corresponding connection on the application board or DSP.

Table 2-1 Serial EEPROM Pins

Pin	Description	ESSI Version	SCI Version
$\overline{CS}$	Chip Select	AAx (DSP Port A)[1]	
SI	Serial Data Input	STDx (ESSIx)[2]	TXD (SCI)
SO	Serial Data Output	SRDx (ESSIx)[2]	RXD (SCI)
SC	Serial Clock, provided by DSP (ESSIx[2] or SCI)	SCKx (ESSIx)[2]	SCLK (SCI)
WP	Write Protect, disables memory programming if asserted	pulled-up	
$\overline{HOLD}$	Halts Serial Communication if set	pulled-up	
VCC	Power Supply	board supplied	
GND	Ground	board supplied	

Notes: **1.** The application discussed here uses pin AA1. Any of the AAx pins (AA0-AA3) could be used. $\overline{CS}$ can also be achieved via any GPIO pin, as explained in **Section 2.3**.

2. This application addresses ESSI0, although it can run on ESSI1 with the appropriate register name changes.

ESSI/SCI and Port A act as the serial interface for the DSP while the two additional control pins ($\overline{HOLD}$ and $\overline{WP}$) are pulled-up. Power Supply and ground are provided by the board. All these lines should connect on the EEPROM according to the corresponding specification.

2.2 ESSI Pin Connections

Figure 2-1 outlines a DSP-Serial EEPROM connection using ESSI. The ESSI supplies the serial clock to the EEPROM through its Serial Clock (SCK) Pin, once the Port C P3 Pin is configured as ESSI. The Serial Data Input (SI) line is provided at the Port C P5 Pin once this pin is configured as the ESSI TX0 Serial Transmitter Output (STD) Pin.

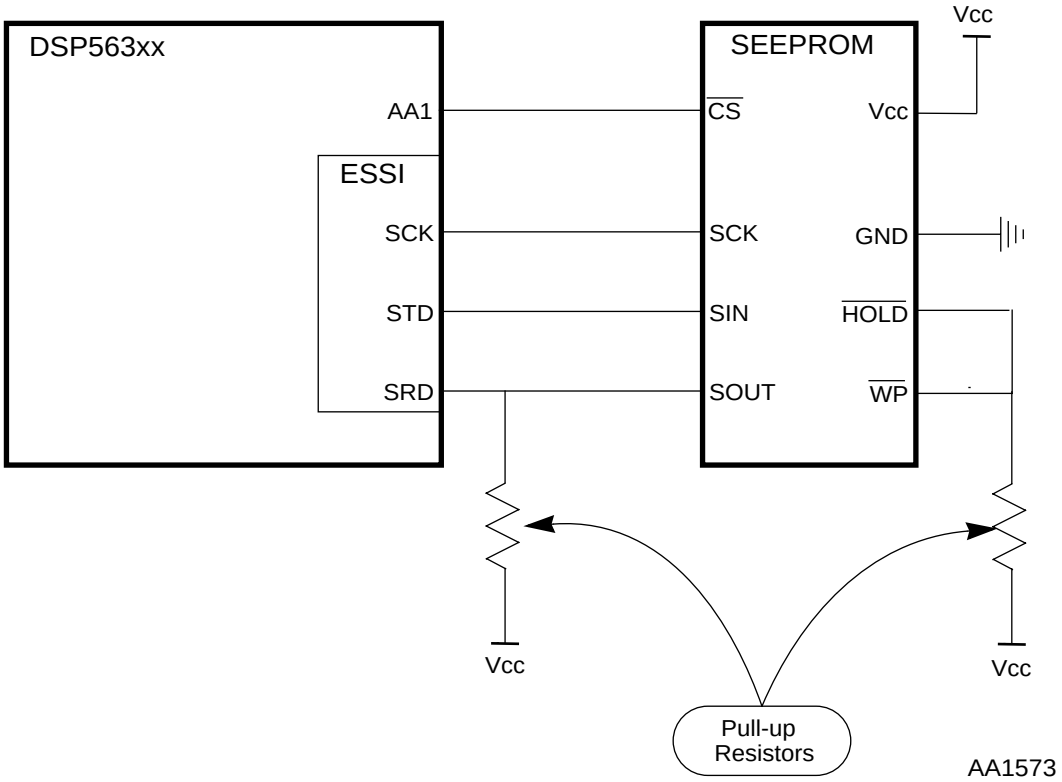

Figure 2-1 DSP - Serial EEPROM Connection to the ESSI

The Serial Data Output (SO) line is supplied by the Serial EEPROM, driving the Port C P4 Pin once it is configured as the ESSI Serial Receive Data (SRD) Pin. In read operations, most Serial EEPROMs keep this line tri-stated until the address byte is received. Since the ESSI works in synchronous mode reading dummy bytes during this period, we recommend pulling up this line to minimize power consumption.

2.3 SCI Pin Connections

Figure 2-2 outlines a DSP-Serial EEPROM connection using SCI.

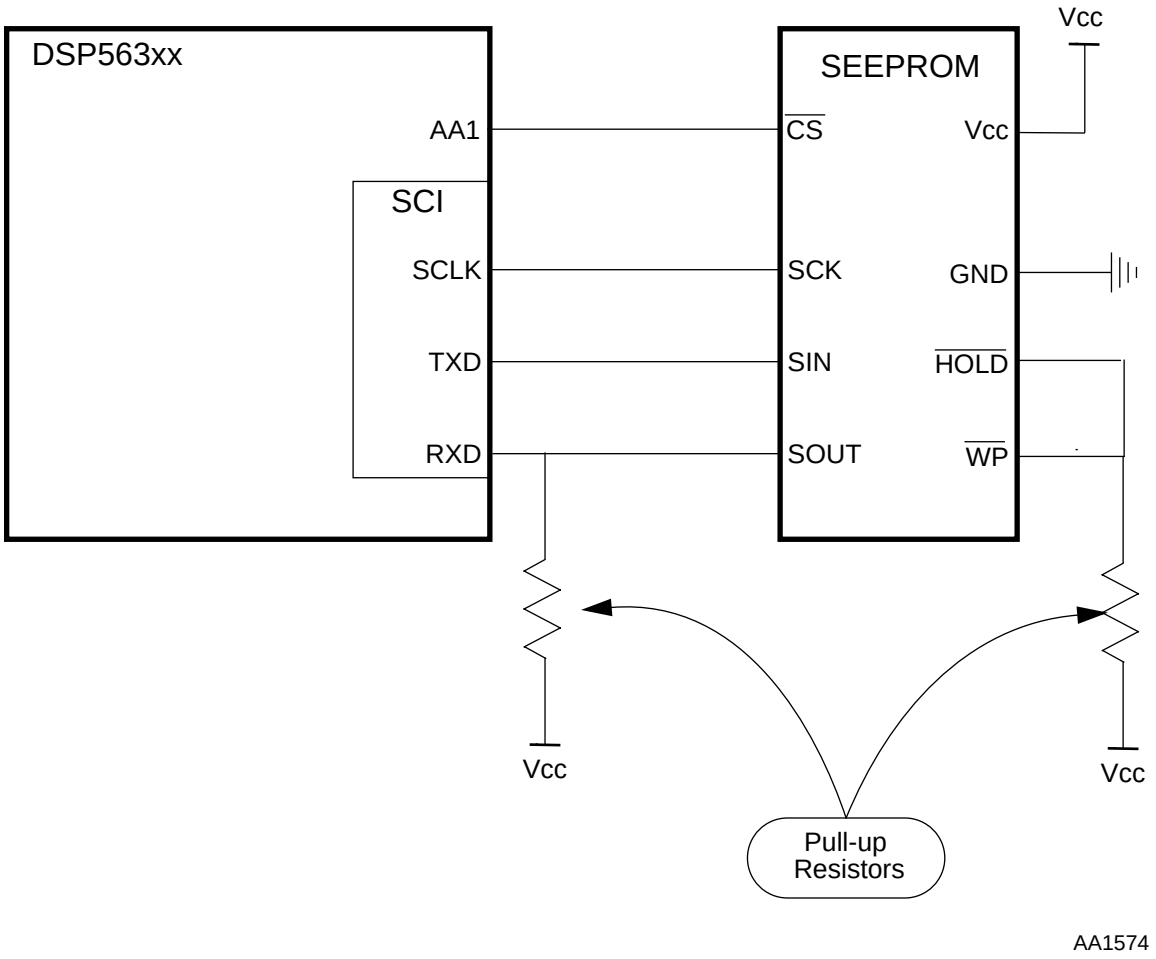

Figure 2-2 DSP - Serial EEPROM Connection with SCI

The SCI supplies the serial clock to the EEPROM through its Serial Clock (SCLK) Pin once the Port E P2 Pin is configured as SCI. A Serial Data Input (SI) line is provided at the Port E P1 Pin once it is configured as the SCI Transmit Data (TXD) Pin.

The Serial Data Output (SO) line is supplied by the Serial EEPROM, driving the Port E P0 Pin, configured as the SCI Receive Data (RXD) Pin. In read operations, most Serial EEPROMs keep this line tri-stated until the address byte is received. Since SCI works in synchronous mode reading dummy bytes during this period, we recommend pulling up this line to minimize power consumption.

2.4 Chip Select ($\overline{CS}$)

The Serial EEPROM receives a Chip Select ($\overline{CS}$) signal from the DSP. Any General-Purpose I/O (GPIO) pin can be used to implement Chip Select for the Serial EEPROM, as long as the pin is kept deasserted any time the EEPROM is not in use. Here, $\overline{CS}$ is driven at the Port A Address Attribute or Row Address Strobe Pin (AA1/$\overline{RAS}$1), configured as Address Attribute. Any activity on serial interface pins while $\overline{CS}$ is deasserted has no effect on the Serial EEPROM.

The $\overline{CS}$ line is asserted and deasserted by changing the pin polarity, with no relation to the Port A Address Attribute mechanism. This pin cannot be used with *real* Address Attribute functionality in applications implementing the currently-described connection. This procedure permits usage of the ESSI or SCI for other connections besides Serial EEPROM in the same application.

To configure the AA1/$\overline{RAS}$1 ($\overline{CS}$) Pin as an Address Attribute, DRAM access should *not* be defined for the external access type through the *External Access Type and Pin Definition Bits (BAT(1:0) = 10)*, at the corresponding Address Attribute Register (AAR1). The pin polarity is determined in AAR1, through the *AA Pin Polarity (BAAP) Bit*. Since AA1 is not used for external access, the AA1 pin always reflects an inactive status. A set BAAP bit gives an active high pin, so the AA1 ($\overline{CS}$) Pin is low and Chip Select is asserted. The default after reset is a cleared BAAP that provides an active low pin, or the deassertion of the Chip Select Pin (AA1 - $\overline{CS}$). If BAAP is set, the AA1 ($\overline{CS}$) pin is active high and $\overline{CS}$ is asserted.

2.5 $\overline{HOLD}$ And $\overline{WP}$ Lines

Asserting the $\overline{HOLD}$ pin halts serial communication without resetting the current sequence. This option is not implemented in this application report. This line is hard-wired deasserted through a pull-up resistor.

The $\overline{WP}$ pin disallows write operations to memory when it is asserted. The application discussed here provides writing functions, so this pin is kept deasserted by a pull-up resistor. Write protection can be achieved by software through a proper call to one of the write protection handling functions.

Active usage of these lines can be achieved by GPIO pins, but such usage is beyond the scope of this application.

3 System Implementation

This section presents a set of assembly routines that accomplish high-level functions for transparent access to the Serial EEPROM. These functions allow any application running on the DSP563xx to interact with the Serial EEPROM by subroutine calls with memory-mapped arguments. Along with the high-level functions, auxiliary lower-level routines compose a kernel to perform the Serial EEPROM tasks employed by the functions. These kernel functions are also available for direct call by any application.

3.1 Block Diagram

Figure 3-1 shows a block diagram of a general application that interacts with a Serial EEPROM. As **Section 2** shows, any application running on a DSP563xx can access an external Serial EEPROM connected to a serial interface, ESSI, or SCI. All interaction occurs through an Application Program Interface (API). No additional code is needed. Optionally, the application can call lower-level routines (kernel routines) for a direct interaction with the Serial EEPROM.

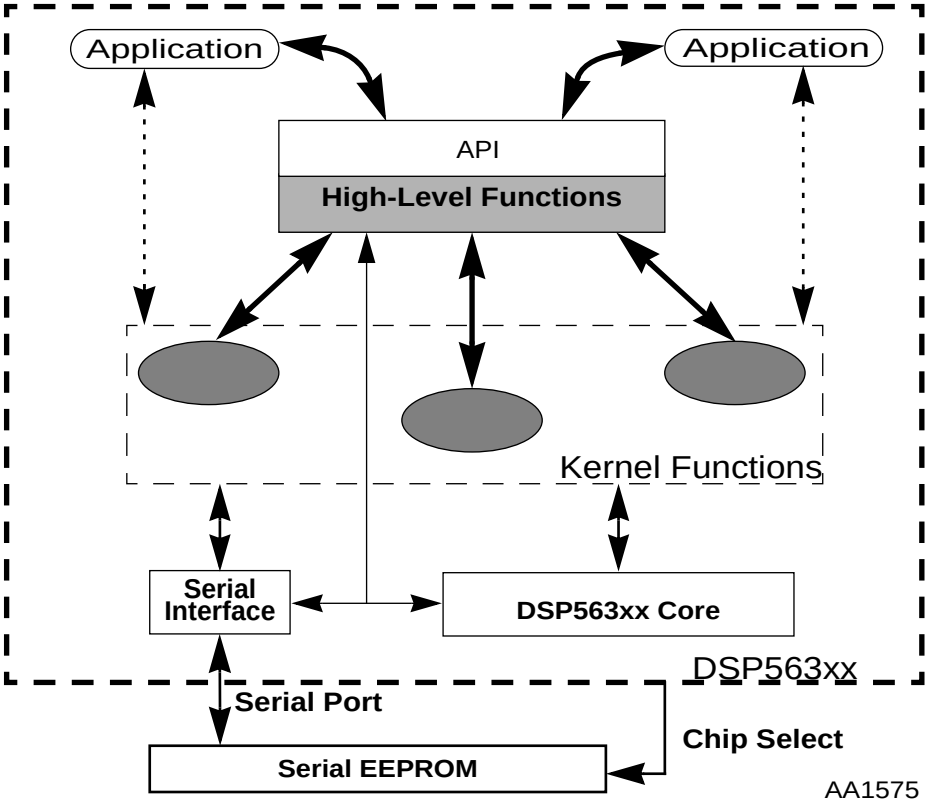

Figure 3-1 System Implementation

3.2 High-Level Functions

The high-level Serial EEPROM functions perform *write to SEEPROM, read from SEEPROM* and *SEEPROM write protection control*. **Table 3-1** summarizes these functions.

Table 3-1 High-Level Functions

Function	Description
WRITE_BLOCK	Copies a block of N x M-byte words from any DSP memory space to Serial EEPROM
READ_BLOCK	Copies a block of N x M-byte words from Serial EEPROM to any DSP memory space
PROTECT	Write-protects Serial EEPROM above any given address
PROTECT_ALL	Write-protects all Serial EEPROM
UNPROTECT	Write-unprotects Serial EEPROM below any given address
UNPROTECT_ALL	Write-unprotects all Serial EEPROM

3.3 Application Program Interface

A straightforward Application Program Interface (API) is provided for interaction with Serial EEPROM. The basic procedure consists of two steps:

1. DATA FEED: transferring arguments to data memory through a set of **move** instructions, DMA transfers, or previous Serial EEPROM download
2. SUBROUTINE CALL: any flow control instructions (**jumps** and **branches** to a correspondent subroutine)

Notes: **1.** PROTECT_ALL and UNPROTECT_ALL functions do not require step 1.

2. Memory protection handling is performed by the user's application through a suitable call to any of the write protection handling functions. The API does execute a call of WRITE_BLOCK to a protected address, although SEEPROM does not complete the write cycle because of the protection.

The following example shows the code for general accesses to the API:

Example 3-1 API Access Code

```
;-----------------------------------------------------------------------
; MACRO definition
;-----------------------------------------------------------------------
API        MACRO      ARGUMENT,VALUE
           move       #(VALUE),r0
           move       r0,x:ARGUMENT

           ENDM
;-----------------------------------------------------------------------
; call <FUNCTION> with parameters: PARAMETER_0,PARAMETER_1,...,PARAMETER_n
;-----------------------------------------------------------------------
           API        <PARAMETER_0>,<VALUE_FOR_PARAMETER_0>
           API        <PARAMETER_1>,<VALUE_FOR_PARAMETER_1>
           API        <PARAMETER_2>,<VALUE_FOR_PARAMETER_2>
           .
           .
           .
           API        <PARAMETER_n>,<VALUE_FOR_PARAMETER_n>
           bsr        <FUNCTION>
;-----------------------------------------------------------------------
; call <FUNCTION> without parameters
;-----------------------------------------------------------------------
           bsr        <FUNCTION>
;-----------------------------------------------------------------------
```

The sections that follow completely describe all the functions and respective parameters. Each description includes the function's flowchart, timing diagrams, and the routine's code. Major steps in the routine are indexed and referenced both in the flowchart and the timing diagrams. Functions that require calling parameters, are summarized in a table for each function. Descriptions conclude with an example function call.

The code provided is the same for both the ESSI and SCI versions. Particular portions corresponding to one peripheral or the other (ESSI or SCI) are selected during assembling through the equate SERIAL_INTERFACE. This equate is defined in **Appendix A**. To assemble the code for ESSI, SERIAL_INTERFACE should be equal to 'ESSI'. Similarly, to assemble the code for SCI, SERIAL_INTERFACE equate must be 'SCI'.

3.3.1 WRITE_BLOCK Function

The WRITE_BLOCK function permits the application to copy a block of words from DSP memory to the Serial EEPROM. The function performs Serial EEPROM page

management in a transparent way. **Figure 3-2** displays the WRITE_BLOCK function flowchart; **Figure 3-3** and **Figure 3-4** show the function's timing scheme.

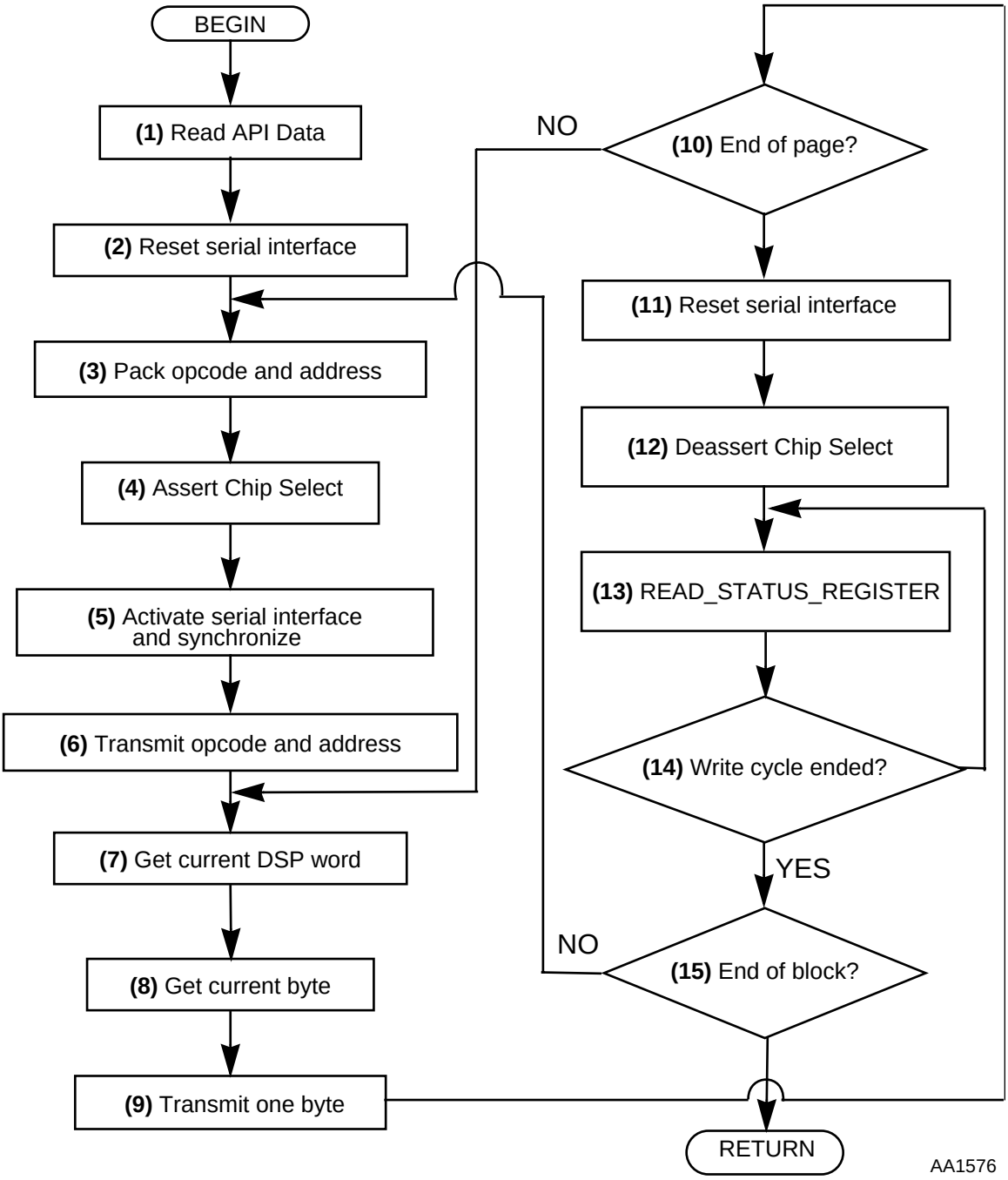

Figure 3-2 WRITE_BLOCK Flow-Chart

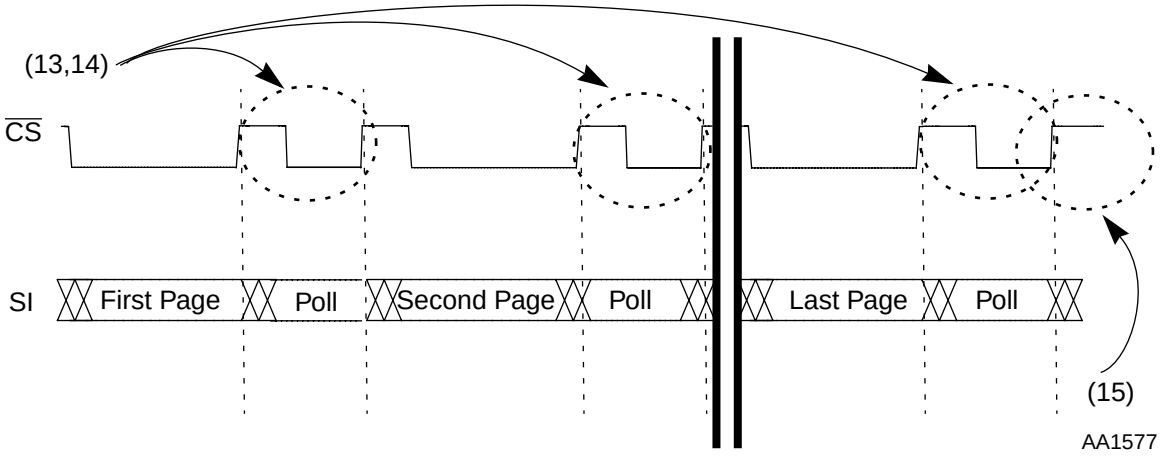

Figure 3-3 WRITE_BLOCK Timing for One Page

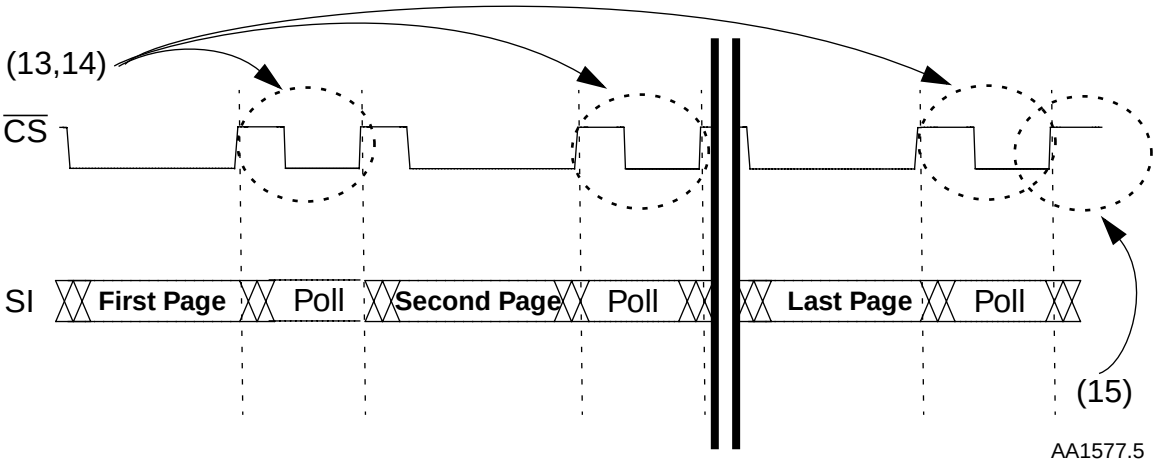

Figure 3-4 WRITE_BLOCK Timing

Table 3-2 depicts WRITE_BLOCK parameters.

Table 3-2 WRITE_BLOCK Parameters

Parameter	Description	Range	API (1)
WR_N_SRC_N	number of words to be written to SEEPROM	1 to 16M-words (24 bits)	x:$8
WR_N_SRC_SPC	block source memory space	X,Y or P, case insensitive	x:$9
WR_N_SRC_ADD	block source base address	any mapped DSP memory address	x:$A
WR_N_DEST_ADD	Serial EEPROM address for LSB of first word	$00 to $FF (8 bits)	x:$B
WR_N_PAGE_SIZE	Serial EEPROM page size minus 1	Device dependent, must be a power of 2 (-1)	x:$C
WR_N_WRD_SZ	word size in bytes	1 for byte, 2 for 16-bit words, 3 for 24-bit words	x:$D
WR_N_STAT_REG	internal use		x:$E
WR_N_COUNTER	internal use		x:$F

Note: These addresses are determined by the assembler equates. The values correspond to those in **Appendix A** and can be changed by modifying the equates appropriately, with no additional change needed in the code.

Example 3-2 presents the assembly code for the WRITE_BLOCK routine.

Example 3-2 WRITE_BLOCK Routine Assembly Code

```
WRITE_BLOCK
            ;-------------------------------------------------------------
            ; (1) READ API DATA
            ;-------------------------------------------------------------
            move        #$0,r0
            move        r0,x:WR_N_COUNTER      ; clear counter
            move        x:WR_N_SRC_ADD,r0
            move        x:WR_N_DEST_ADD,b
            move        x:WR_N_PAGE_SIZE,m2
            move        #$1,r3
            move        x:WR_N_WRD_SZ,m3
            move        x:WR_N_DEST_ADD,r4
            move        m2,x0
            and         x0,b
            clr         a
            move        b,r2
            ;-------------------------------------------------------------
            ; loop WRITE_PAGE until all DSP words have been transmitted
            ;-------------------------------------------------------------
WRITE_PAGE
            ;-------------------------------------------------------------
            ; Write Enable
            ;-------------------------------------------------------------
            bsr         WRITE_ENABLE
            ;-------------------------------------------------------------
            ; (2) RESET SERIAL INTERFACE
            ;-------------------------------------------------------------
            bsr         SERIAL_INTERFACE_RESET
            ;-------------------------------------------------------------
            ; (3) PACK OPCODE and ADDRESS
            ;-------------------------------------------------------------
            move        r4,a2
            asr         #24,a,a
            move        #WRITE_OPCODE,a2
            asr         #8,a,a                 ; now we have WR_N_DEST_ADD i
                                               ; A0 and WRITE_OPCODE in A1
            ;-------------------------------------------------------------
            ; (4) ASSERT CHIP SELECT
            ;-------------------------------------------------------------
            movep       #$4,x:M_AAR1           ; set AA1 low
            ;-------------------------------------------------------------
            ; (5) ACTIVATE SERIAL INTERFACE and SYNCHRONIZE
            ;-------------------------------------------------------------
            bsr         SYNCHRONIZE
            ;-------------------------------------------------------------
```

Example 3-2 WRITE_BLOCK Routine Assembly Code (Continued)

```
        ;-----------------------------------------------------------
        ; (6) TRANSMIT OPCODE and ADDRESS
        ;-----------------------------------------------------------
        IF SERIAL_INTERFACE=='ESSI'

        movep     a0,x:M_TX00                     ; load 2nd valid byte to be
                                                  ; TXed (address, B2)
        brclr     #M_RDF,x:M_SSISR0,*             ; wait until byte is TXed
                                                  ; (opcode, B1)
        movep     x:M_RX0,n5                      ; clear RDF bit

        ELSE
        IF SERIAL_INTERFACE=='SCI'

        brclr     #M_TDRE,x:M_SSR,*               ; wait until byte is TXed
                                                  ; (opcode, B0)
        movep     a0,x:M_STXH                     ; load 2nd valid byte to be TXed
                                                  ; (address, B1)
        brclr     #M_RDRF,x:M_SSR,*               ; clean receiver
        movep     x:M_SRXH,a1
        nop                                       ; pipeline delay

        ENDIF
        ENDIF
        ;-----------------------------------------------------------
        ; which space?
        ;-----------------------------------------------------------
        clr       b
        clr       a          x:WR_N_COUNTER,b0
        move      #>$20,x0
        move      x:WR_N_SRC_SPC,a
        cmp       #$70,a                          ; is it lowercase?
        sub       x0,a       ifge                 ; capitalize
        move      x:WR_N_SRC_N,x0
        cmp       #'X',a
        move      #(_xin_word-_end+1),r5
        beq       _xin_word
        cmp       #'Y',a
        move      #(_yin_word-_end+1),r5
        beq       _yin_word
        move      #(_pin_word-_end+1),r5
        ;-----------------------------------------------------------
        ; transmit page
        ;-----------------------------------------------------------
        ;-----------------------------------------------------------
        ; (7) GET CURRENT DSP WORD
        ;-----------------------------------------------------------
_pin_word
        move      p:(r0),a1                       ; case P
        bra       _cont
_yin_word
        move      y:(r0),a1                       ; case Y
        bra       _cont
_xin_word
        move      x:(r0),a1                       ; case X
_cont
```

Example 3-2 WRITE_BLOCK Routine Assembly Code (Continued)

```
            ;-------------------------------------------------------------
            ; (8) GET CURRENT BYTE
            ;-------------------------------------------------------------
            do          r3,_cut1byte
            asr         #$8,a,a
_cut1byte
            move        (r3)+                           ; updates byte pointer in
                                                        ; current DSP word
            ;-------------------------------------------------------------
            ; (9) TRANSMIT ONE BYTE
            ;-------------------------------------------------------------
            IF SERIAL_INTERFACE=='ESSI'

            movep       a0,x:M_TX00                     ; load Nth valid byte to be TXed
                                                        ; (data, B.n)
            brclr       #M_RDF,x:M_SSISR0,*             ; wait until byte is TXed
                                                        ; (address/data, B.n-1)
            movep       x:M_RX0,n5                      ; clear RDF bit
            nop                                         ; pipeline delay
            nop                                         ; pipeline delay

            ELSE
            IF SERIAL_INTERFACE=='SCI'

            brclr       #M_TDRE,x:M_SSR,*               ; wait until byte is TXed
            movep       a0,x:M_STXH                     ; load Nth valid byte to be TXed
                                                        ; (data, B.n)
            brclr       #M_RDRF,x:M_SSR,*               ; clean receiver
            movep       x:M_SRXH,a1

            ENDIF
            ENDIF

            move        r3,a
            move        (r4)+
            tst         a
            bne         _upd                            ; current DSP word finished?
            move        (r3)+                           ; resets r3 to 1 (byte pointer)
            move        (r0)+                           ; points to next DSP word
            inc         b
            move        x0,a0
            cmp         a,b                             ; compare number of TXed words
                                                        ; to number of DSP words
            beq         _end                            ; end transaction in case all
                                                        ; DSP word have been TX
_upd
            move        (r2)+                           ; points to next in-page address
            move        r2,a
            tst         a
            bne         r5                              ; continues until END_OF_PAGE
_end        nop
```

Example 3-2 WRITE_BLOCK Routine Assembly Code (Continued)

```
;-----------------------------------------------------------
; (10) END OF PAGE
;-----------------------------------------------------------
IF SERIAL_INTERFACE=='ESSI'

brclr      #M_RDF,x:M_SSISR0,*              ; wait until last byte is TXed
movep      x:M_RX0,n5                       ; clear RDF bit
move       b0,x:WR_N_COUNTER                ; save number of TXed words

ELSE
IF SERIAL_INTERFACE=='SCI'

brclr      #M_TDRE,x:M_SSR,*                ; wait until byte is TXed
                                            ; (address, B1)
brclr      #M_RDRF,x:M_SSR,*                ; clean receiver
movep      x:M_SRXH,a1
move       b0,x:WR_N_COUNTER                ; save number of TXed words

ENDIF
ENDIF
;-----------------------------------------------------------
; (11) RESET SERIAL INTERFACE
;-----------------------------------------------------------
bsr        SERIAL_INTERFACE_RESET
;-----------------------------------------------------------
; (12) DEASSERT CHIP SELECT
;-----------------------------------------------------------
bclr       #M_BAAP,x:M_AAR1                 ; set AA1 high
;-----------------------------------------------------------
; (13) READ_STATUS_REGISTER
;-----------------------------------------------------------
bsr        POLL_SR
;-----------------------------------------------------------
; (14) WRITE CYCLE ENDED?
;-----------------------------------------------------------
clr        b          x:WR_N_SRC_N,x0
clr        a          x:WR_N_COUNTER,b0
move       x0,a0
nop
nop
;-----------------------------------------------------------
; (15) END OF BLOCK?
;-----------------------------------------------------------
cmp        a,b                              ; compare number of transmitted
                                            ; words to number of DSP words
bne        WRITE_PAGE
nop
;-----------------------------------------------------------
; RETURN
;-----------------------------------------------------------
rts
```

Example 3-3 shows a call to WRITE_BLOCK for copying a 24-word block of 24-bit words (3-byte words) from DSP X memory, address $204, to address $0 of an 8-byte page Serial EEPROM.

Example 3-3 A WRITE_BLOCK Call

```
;-----------------------------------------------------------------------
; a WRITE_BLOCK call
;-----------------------------------------------------------------------
          API       WR_N_SRC_N,$18                  ; 24 words
          API       WR_N_SRC_SPC,"'x'"              ; from X memory
          API       WR_N_SRC_ADD,$204               ; Block begins on address $204
          API       WR_N_DEST_ADD,$0                ; Block is written to address $0
          API       WR_N_PAGE_SIZE,$7               ; SEEPROM's page size is 8 bytes
          API       WR_N_WRD_SZ,$3                  ; word size is 3 bytes = 24 bits
          bsr       WRITE_BLOCK                     ; branch to WRITE_BLOCK subroutine
```

3.3.2 READ_BLOCK Function

A call to READ_BLOCK reads a block of words from Serial EEPROM to DSP memory. **Figure 3-5** shows the READ_BLOCK function flowchart; **Figure 3-6** shows the function's timing scheme. **Figure 3-3** displays READ_BLOCK parameters.

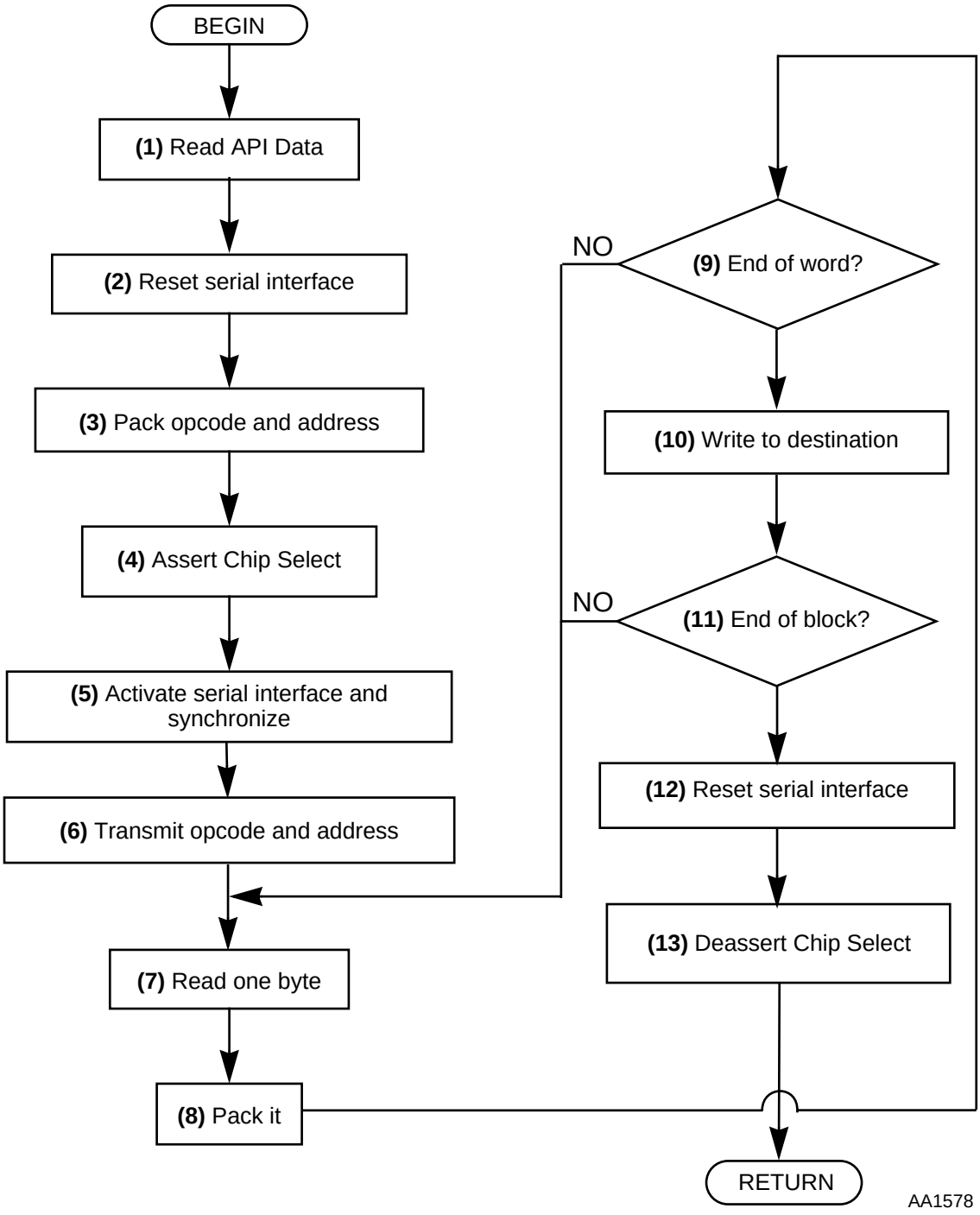

Figure 3-5 READ_BLOCK Flow Chart

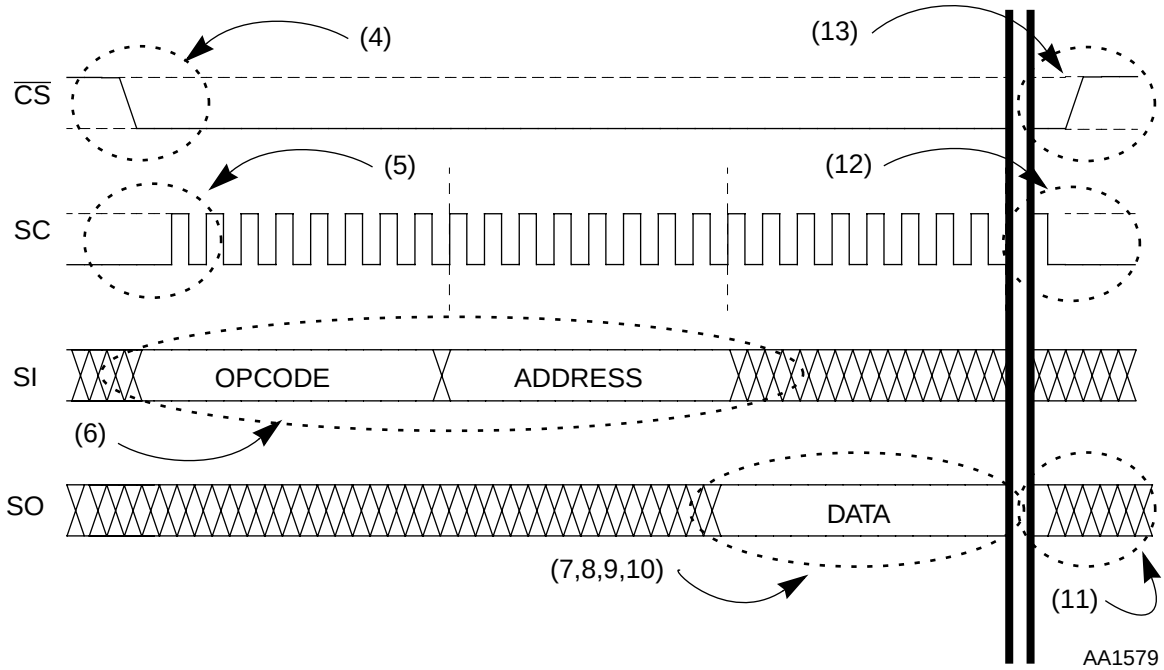

Figure 3-6 READ_BLOCK Timing

Table 3-3 READ_BLOCK Parameters

Parameter	Description	Range	API (1)
RD_N_SRC_N	Number of words to be read from Serial EEPROM	1 to 16M-words (24 bits)	x:$0
RD_N_SRC_ADD	Serial EEPROM address for LSB of first word	$00 to $FF (8 bits)	x:$1
RD_N_DEST_SPC	Block destination memory space	X,Y or P, case insensitive	x:$2
RD_N_DEST_ADD	Block destination base address	Any mapped DSP memory address	x:$3
RD_N_WRD_SZ	Word size in bytes	1 for byte, 2 for 16-bit words, 3 for 24-bit words	x:$4

Note: These addresses are determined by assembler equates. The values correspond to those in **Appendix A** and can be changed by modifying the equates appropriately, with no additional change needed in the code.

Example 3-4 shows the assembly code for the READ_BLOCK routine.

Example 3-4 READ_BLOCK Assembly Code

```
READ_BLOCK
          ;-------------------------------------------------------------
          ; (1) READ API DATA
          ;-------------------------------------------------------------
          clr       a
          move      #>$20,x0
          move      x:RD_N_SRC_ADD,a2
          move      x:RD_N_DEST_ADD,r0
          move      x:RD_N_DEST_SPC,b
          move      x:RD_N_WRD_SZ,x1
          cmp       #$70,b                          ; is it lowercase?
          sub       x0,b       ifge                 ; capitalize
          move      x:RD_N_SRC_N,x0
          ;-------------------------------------------------------------
          ; (2) RESET SERIAL INTERFACE
          ;-------------------------------------------------------------
          bsr       SERIAL_INTERFACE_RESET
          ;-------------------------------------------------------------
          ; (3) PACK OPCODE and ADDRESS
          ;-------------------------------------------------------------
          asr       #24,a,a
          move      #READ_OPCODE,a2
          asr       #8,a,a                          ; now we have RD_N_SRC_ADD in A0
                                                    ; and READ_OPCODE in A1
          ;-------------------------------------------------------------
          ; (4) ASSERT CHIP SELECT
          ;-------------------------------------------------------------
          movep     #$4,x:M_AAR1                    ; change AA1 low
          ;-------------------------------------------------------------
          ; (5) ACTIVATE SERIAL INTERFACE and SYNCHRONIZE
          ;-------------------------------------------------------------
          bsr       SYNCHRONIZE
          ;-------------------------------------------------------------
          ; (6) TRANSMIT OPCODE and ADDRESS
          ;-------------------------------------------------------------
          IF SERIAL_INTERFACE=='ESSI'

          movep     a0,x:M_TX00                     ; load 2nd valid byte to be
                                                    ; TXed (address, B2)
          brclr     #M_RDF,x:M_SSISR0,*             ; wait until byte is TXed
                                                    ; (opcode, B1)
          movep     x:M_RX0,n5                      ; clear RDF bit
          brclr     #M_RDF,x:M_SSISR0,*             ; wait until byte is TXed
                                                    ; (address, B2)
          movep     x:M_RX0,n5                      ; clear RDF bit
```

Example 3-4 READ_BLOCK Assembly Code (Continued)

```
            ELSE
            IF SERIAL_INTERFACE=='SCI'

            brclr      #M_TDRE,x:M_SSR,*                ; wait until byte is TXed
                                                        ; (opcode, B0)
            movep      a0,x:M_STXH                      ; load 2nd valid byte to be
                                                        ; TXed (address, B1)
            brclr      #M_RDRF,x:M_SSR,*                ; clean receiver
            movep      x:M_SRXH,a1
            nop                                         ; pipeline delay

            ;-------------------------------------------------------------
            brclr      #M_TDRE,x:M_SSR,*                ; wait until byte is TXed
                                                        ; (address, B1)
            movep      #$ff0000,x:M_STXH                ; keep transmitting to maintain
                                                        ; clock
            brclr      #M_RDRF,x:M_SSR,*                ; clean receiver
            movep      x:M_SRXH,a1

            ENDIF
            ENDIF
            ;-------------------------------------------------------------
            ; which space?
            ;-------------------------------------------------------------
            cmp        #'X',b
            move       #(_x-_p+1),r5
            beq        _sp_end
            cmp        #'Y',b
            move       #(_y-_p+1),r5
            beq        _sp_end
            move       #$1,r5
_sp_end
            ;-------------------------------------------------------------
            ; read words and write to DSP memory
            ;-------------------------------------------------------------
            do         x0,_rd_n_ws                      ; read N words
            do         x1,_rd_bytes                     ; read bytes

            IF SERIAL_INTERFACE=='ESSI'

            brclr      #M_RDF,x:M_SSISR0,*              ; wait until ESSI0's receiver is
                                                        ; full
            ;-------------------------------------------------------------
            ; (7) READ ONE BYTE
            ;-------------------------------------------------------------
            movep      x:M_RX0,a1

            ELSE
            IF SERIAL_INTERFACE=='SCI'

            brclr      #M_TDRE,x:M_SSR,*                ; wait until byte is TXed
            movep      #$ff0000,x:M_STXH                ; keep transmitting to maintain
                                                        ; clock
```

Example 3-4 READ_BLOCK Assembly Code (Continued)

```
            ;-------------------------------------------------------------
            ; (7) READ ONE BYTE
            ;-------------------------------------------------------------
            brclr      #M_RDRF,x:M_SSR,*                ; read received byte
            movep      x:M_SRXH,a1

            ENDIF
            ENDIF
            ;-------------------------------------------------------------
            ; (8) PACK IT
            ;-------------------------------------------------------------
            lsr        #16,a
            asr        #8,a,a
_rd_bytes
            ;-------------------------------------------------------------
            ; (9) END OF WORD?
            ;-------------------------------------------------------------
            rep        x1
            asl        #8,a,a
            ;-------------------------------------------------------------
            ; (10) WRITE TO DESTINATION
            ;-------------------------------------------------------------
            bra        r5
_p
            move       a1,p:(r0)+
            bra        _rd_end
_x
            move       a1,x:(r0)+
            bra        _rd_end
_y
            move       a1,y:(r0)+
_rd_end
            nop
_rd_n_ws
            ;-------------------------------------------------------------
            ; (11) END OF BLOCK?
            ;-------------------------------------------------------------
            ;-------------------------------------------------------------
            ; (12) RESET SERIAL INTERFACE
            ;-------------------------------------------------------------
            bsr        SERIAL_INTERFACE_RESET
            ;-------------------------------------------------------------
            ; (13) DEASSERT CHIP SELECT
            ;-------------------------------------------------------------
            bclr       #M_BAAP,x:M_AAR1                 ; set AA1 high
            ;-------------------------------------------------------------
            ; RETURN
            ;-------------------------------------------------------------
            rts
```

Example 3-5 shows a call to the READ_BLOCK function for copying a 6-word block of 8-bit words (1-byte words) from address $0 of a Serial EEPROM to DSP X memory, address $104. This example actually reads the first two 24-bit words written to the Serial EEPROM in **Example 3-3** as six independent bytes that are each written on a different DSP X memory address. The LSB of the first 24-bit word written to the Serial EEPROM is read to the LSB of X:$104, and so on.

Example 3-5 A READ_BLOCK Call

```
;-----------------------------------------------------------------------
; a READ_BLOCK call
;-----------------------------------------------------------------------
          API       RD_N_SRC_N,$6                   ; 6-word block
          API       RD_N_SRC_ADD,$0                 ; from SEEPROM's address $0
          API       RD_N_DEST_SPC,"'x'"             ; to DSP X memory
          API       RD_N_DEST_ADD,$100              ; at address $100
          API       RD_N_WRD_SZ,$1                  ; 1-byte words
          bsr       READ_BLOCK
```

3.3.3 PROTECT Function

The PROTECT function protects the Serial EEPROM from writing from a given address' section to the top of the memory. A Serial EEPROM's section corresponds to one fourth of its address range. An address' section is the one to which the address to be protected belongs. **Figure 3-7** displays the PROTECT function flowchart; **Table 3-4** shows the function's parameters.

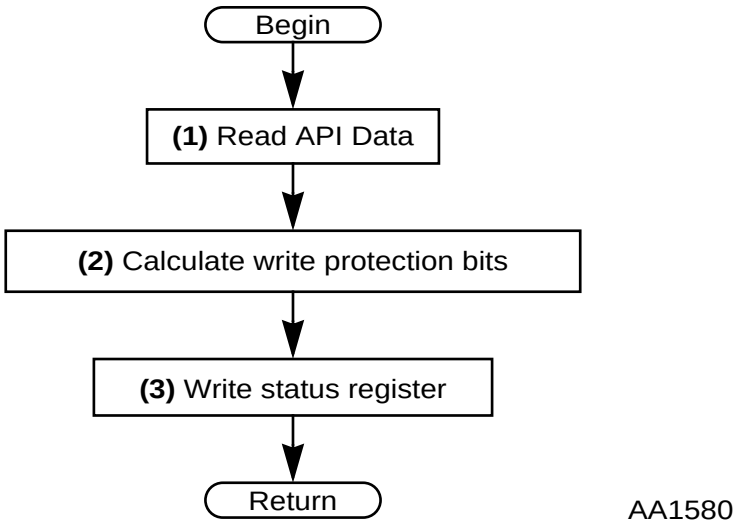

Figure 3-7 PROTECT/UNPROTECT Flow-Chart

Table 3-4 PROTECT Parameters

Parameter	Description	Range	API (1)
PRF_BASE_ADD	Serial EEPROM base address for protection	One byte, $00 to $FF	x:$10
PRF_MEM_SZ	Serial EEPROM's size in Kbits	1 for 1K-bits or 2 for 2K-bits	x:$11

Note: These addresses are determined by assembler equates. The values correspond to those in **Appendix A** and can be changed by modifying the equates, with no additional change needed on the code.

Example 3-6 shows the assembly code for the PROTECT routine.

Example 3-6 PROTECT Function Assembly Code

```
PROTECT
        ;-------------------------------------------------------------
        ; (1) READ API DATA
        ;-------------------------------------------------------------
        clr     a
        clr     b
        move    #>1,y0
        move    #>$40,a1
        move    #>$60,a0
        move    x:PRF_MEM_SZ,b
        dec     b
        asl     b1,a,a
        move    a1,x1
        move    a0,x0
        clr     a
        move    x:PRF_BASE_ADD,a1
        clr     b
        move    #>$3,b
        ;-------------------------------------------------------------
        ; (2) CALCULATE WRITE PROTECTION BITS
        ;-------------------------------------------------------------
        cmp     x1,a
        sub     y0,b    ifge
        cmp     x0,a
        sub     y0,b    ifge
        lsl     #2,b
        ;-------------------------------------------------------------
        ; (3) WRITE STATUS REGISTER
        ;-------------------------------------------------------------
        move    b1,r0
        move    r0,x:WRSR_DATA
        bsr     WRITE_STATUS_REG
        nop
        ;-------------------------------------------------------------
        ; RETURN
        ;-------------------------------------------------------------
        rts
```

The following example calls the PROTECT function in order to write-protect all addresses above address $d0 on a 2K-bit SEEPROM. The routine protects all addresses above $d0 location's section, including the section itself—that is, all addresses above $c0.

Example 3-7 A PROTECT Call

```
;-----------------------------------------------------------------------
; a PROTECT call
;-----------------------------------------------------------------------
          API        PRF_BASE_ADD,$d0
          API        PRF_MEM_SZ,$2
          bsr        PROTECT
```

3.3.4 UNPROTECT Function

Calling the UNPROTECT function cancels write protection for the Serial EEPROM from memory's bottom address to a given address' section. The UNPROTECT function flowchart is the same as for PROTECT, shown in **Figure 3-7**. **Table 3-5** displays the parameters of the UNPROTECT function.

Table 3-5 UNPROTECT Parameters

Parameter	Description	Range	API[(1)]
UPRF_BASE_ADD	Serial EEPROM base address for protection	One byte, $00 to $FF	x:$12
UPRF_MEM_SZ	Serial EEPROM's size in Kbits	1 for 1K-bits or 2 for 2K-bits	x:$13

Note: These addresses are determined by assembler equates. The values correspond to those in **Appendix A.** These values can be changed by modifying the equates appropriately, with no additional change needed in the code.

Example 3-8 shows the assembly code for the UNPROTECT routine.

Example 3-8 UNPROTECT Function Assembly Code

```
UNPROTECT
        ;-------------------------------------------------------------
        ; (1) READ API DATA
        ;-------------------------------------------------------------
        clr     a
        clr     b
        move    #>1,y0
        move    #>$40,a1
        move    #>$60,a0
        move    x:UPRF_MEM_SZ,b
        dec     b
        asl     b1,a,a
        move    a1,x1
        move    a0,x0
        clr     a
        move    x:UPRF_BASE_ADD,a1
        clr     b
        move    #$0,b
        ;-------------------------------------------------------------
        ; (2) CALCULATE WRITE PROTECTION BITS
        ;-------------------------------------------------------------
        cmp     x0,a
        add     y0,b    iflt
        cmp     x1,a
        add     y0,b    iflt
        lsl     #2,b
        ;-------------------------------------------------------------
        ; (3) WRITE STATUS REGISTER
        ;-------------------------------------------------------------
        move    b1,r0
        move    r0,x:WRSR_DATA
        bsr     WRITE_STATUS_REG
        nop
        ;-------------------------------------------------------------
        ; RETURN
        ;-------------------------------------------------------------
        rts
```

Example 3-9 calls for UNPROTECT enabling a 2K-bit Serial EEPROM to be written on all addresses below address $b0. The routine unprotects all addresses below the $d0 location's section as well, including the section itself, that is, all addresses below $c0.

Example 3-9 An UNPROTECT Call

```
;-----------------------------------------------------------------------
; a UNPROTECT call
;-----------------------------------------------------------------------
          API       UPRF_BASE_ADD,$b0
          API       UPRF_MEM_SZ,$2
          bsr       UNPROTECT
```

3.3.5 PROTECT_ALL Function

Calling the PROTECT_ALL function protects the whole Serial EEPROM from being written. No parameters are needed. The function writes an adequate value to the SEEPROM Status Register, as shown in the following code.

Example 3-10 The PROTECT_ALL Function

```
PROTECT_ALL
          ;-------------------------------------------------------
          ; call WRSR with this correspondent data
          ;-------------------------------------------------------
          move      #$c,r0
          move      r0,x:WRSR_DATA
          bsr       WRITE_STATUS_REG
          nop
          ;-------------------------------------------------------
          ; RETURN
          ;-------------------------------------------------------
rts
```

To call PROTECT_ALL it is enough to branch to the subroutine, as **Example 3-11** shows.

Example 3-11 A PROTECT_ALL Call

```
;-----------------------------------------------------------------------
; a PROTECT_ALL call
;-----------------------------------------------------------------------
          bsr       PROTECT_ALL
```

3.3.6 UNPROTECT_ALL Function

Calling the UNPROTECT_ALL function cancels write protection for the whole Serial EEPROM. No parameters are needed. The function writes an adequate value to the SEEPROM Status Register, as the following code shows.

Example 3-12 The UNPROTECT_ALL Function

```
UNPROTECT_ALL
            ;-------------------------------------------------------------
            ; call WRSR with this correspondent data
            ;-------------------------------------------------------------
            move        #$0,r0
            move        r0,x:WRSR_DATA
            bsr         WRITE_STATUS_REG
            nop
            ;-------------------------------------------------------------
            ; RETURN
            ;-------------------------------------------------------------
rts
```

To call UNPROTECT_ALL it is enough to branch to the subroutine, as **Example 3-13** shows.

Example 3-13 An UNPROTECT_ALL Call

```
;-----------------------------------------------------------------------
; a UNPROTECT_ALL call
;-----------------------------------------------------------------------
            bsr         UNPROTECT_ALL
```

3.4 Serial EEPROM Functions

Serial EEPROMs accept one-byte serial opcodes delivering the memory basic functionality. The general Serial EEPROM complete instruction set is implemented in this application report. Part of the instruction set is achieved as a sub-set of the high-level functions described in **Section 3.2** and **Section 3.3**. Other instructions accomplished by kernel routines that serve those high-level functions are similarly available for direct call by any application.

Table 3-6 summarizes these low-level functions and the way they work. The following paragraphs describe them. These functions are called in much the same way as the high-level functions described in **Section 3.3**, some with parameters and some without.

Table 3-6 Serial EEPROM Functions

Function	Description	Implementation
WREN	Enables Serial EEPROM for writing	Kernel: WRITE_ENABLE
WRDI	Disables Serial EEPROM for writing	Kernel: WRITE_DISABLE
RDSR	Reads Serial EEPROM's Status Register	Kernel: READ_STATUS_REG
WRSR	Programs Serial EEPROM's Status Register	Kernel: WRITE_STATUS_REG
WRITE	Writes to Serial EEPROM	Sub-set of WRITE_BLOCK
READ	Reads from Serial EEPROM	Sub-set of READ_BLOCK

3.4.1 WRITE_ENABLE and WRITE_DISABLE Functions

The WRITE_ENABLE and WRITE_DISABLE functions do not require parameters. WRITE_ENABLE prepares the Serial EEPROM for writing. It is called by higher-level functions before any programming task, data writing, or status register writing. WRITE_DISABLE disables the Serial EEPROM for writing and is not called by any high-level function. The same routine executes both functions, as **Figure 3-8** shows.

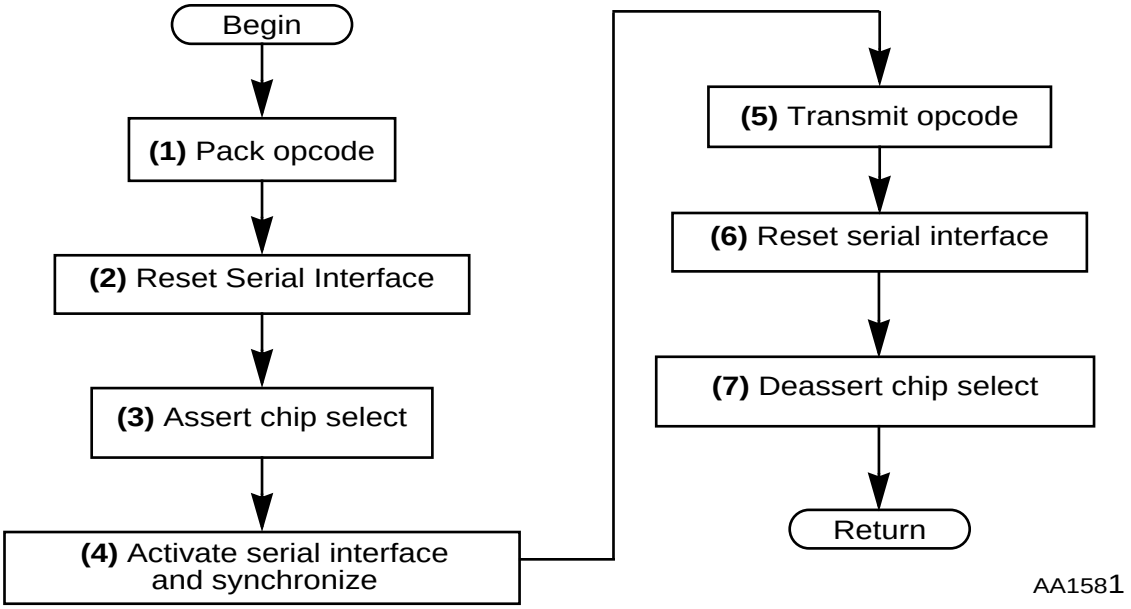

Figure 3-8 WRITE_ENABLE/WRITE_DISABLE Flowchart

Example 3-14 shows the complete assembly code for the WRITE_ENABLE/ WRITE_DISABLE routine.

Example 3-14 WRITE_ENABLE/_DISABLE Routine Assembly Code

```
WRITE_ENABLE
            ;-----------------------------------------------------------
            ; (1) PACK OPCODE and ADDRESS
            ;-----------------------------------------------------------
            move        #$0,a2
            asr         #24,a,a
            move        #WREN_OPCODE,a2
            asr         #8,a,a                          ; now we have WREN_OPCODE in A1
            bra         endis
WRITE_DISABLE
            ;-----------------------------------------------------------
            ; (1) PACK OPCODE and ADDRESS
            ;-----------------------------------------------------------
            move        #$0,a2
            asr         #24,a,a
            move        #WRDI_OPCODE,a2
            asr         #8,a,a                          ; now we have WRDI_OPCODE in A1
endis
            ;-----------------------------------------------------------
            ; (2) RESET SERIAL INTERFACE
            ;-----------------------------------------------------------
            bsr         SERIAL_INTERFACE_RESET
            ;-----------------------------------------------------------
            ; (3) ASSERT CHIP SELECT
            ;-----------------------------------------------------------
            movep       #$4,x:M_AAR1                    ; set AA1 low
            ;-----------------------------------------------------------
            ; (4) ACTIVATE SERIAL INTERFACE and SYNCHRONIZE
            ;-----------------------------------------------------------
            bsr         SYNCHRONIZE
            ;-----------------------------------------------------------
            ; (5) TRANSMIT OPCODE
            ;-----------------------------------------------------------
            IF SERIAL_INTERFACE=='ESSI'

            jclr        #M_RDF,x:M_SSISR0,*             ; wait until byte is TXed
                                                        ; (opcode, B1)
            movep       x:M_RX0,n5                      ; clear RDF bit

            ELSE
            IF SERIAL_INTERFACE=='SCI'

            brclr       #M_TDRE,x:M_SSR,*               ; wait until byte is TXed
                                                        ; (opcode, B0)
            movep       #$ff0000,x:M_STXH               ; keep transmitting to maintain
                                                        ; clock
            brclr       #M_RDRF,x:M_SSR,*               ; clean receiver
            movep       x:M_SRXH,a1
```

Example 3-14 WRITE_ENABLE/_DISABLE Routine Assembly Code (Continued)

```
        ENDIF
        ENDIF
        ;-----------------------------------------------------------
        ; (6) RESET SERIAL INTERFACE
        ;-----------------------------------------------------------
        bsr        SERIAL_INTERFACE_RESET
        ;-----------------------------------------------------------
        ; (7) DEASSERT CHIP SELECT
        ;-----------------------------------------------------------
        bclr       #M_BAAP,x:M_AAR1                ; set AA1 high
        ;-----------------------------------------------------------
        ; RETURN
        ;-----------------------------------------------------------
        rts
```

The WRITE_ENABLE and WRITE_DISABLE functions can be called by branching to the subroutine, with no need for parameters.

Example 3-15 WRITE_ENABLE/_DISABLE Calls

```
;-------------------------------------------------------------------------
; a WRITE_ENABLE call
;-------------------------------------------------------------------------
        bsr        WRITE_ENABLE
;-------------------------------------------------------------------------
; a WRITE_DISABLE call
;-------------------------------------------------------------------------
        bsr        WRITE_DISABLE
```

3.4.2 READ_STATUS_REG Function

The READ_STATUS_REG function reads the SEEPROM Status Register and writes it on the Least Significant Byte of a given address in a given DSP memory space. **Figure 3-9** shows the READ_STATUS_REG function timing scheme; **Figure 3-10** displays the function's flowchart; **Table 3-7** lists the parameters of READ_STATUS_REG.

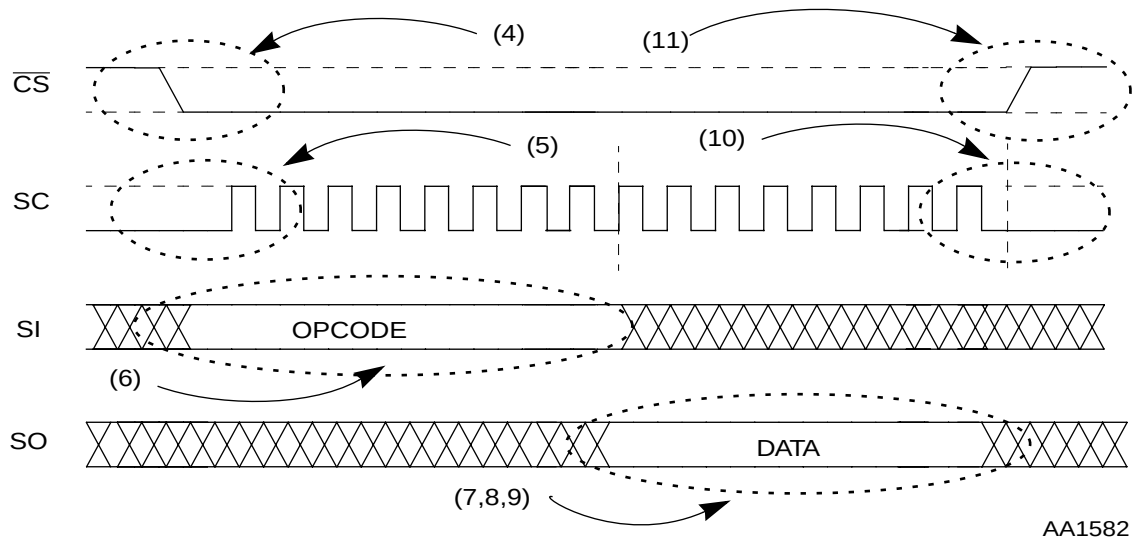

Figure 3-9 READ_STATUS_REG Timing

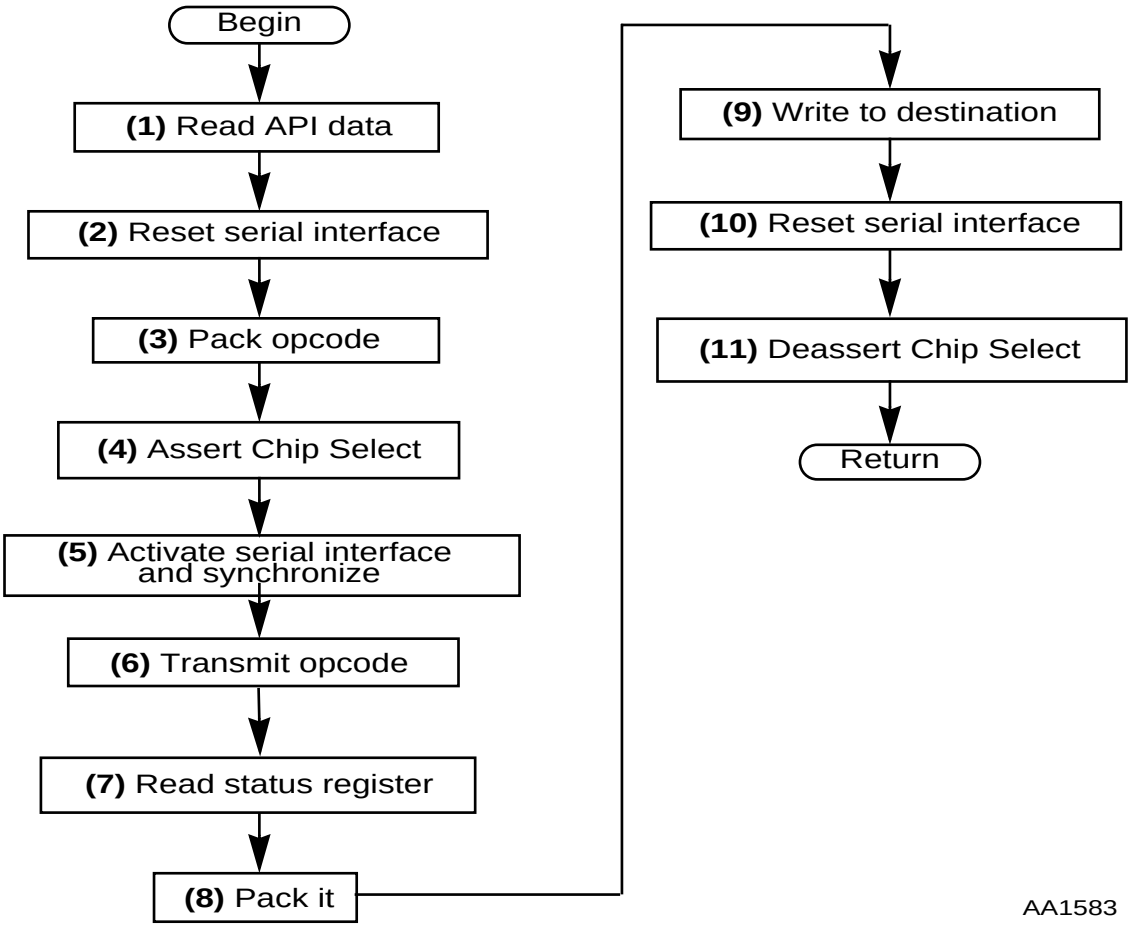

Figure 3-10 READ_STATUS_REG Flow-Chart

Table 3-7 READ_STATUS_REG Parameters

Parameter	Description	Range	Address[1]
RDSR_DEST_SPC	destination memory space	X,Y or P	x:$5
RDSR_DEST_ADD	destination address	any mapped DSP memory address	x:$6

Note: These addresses are determined by assembler equates. The values correspond to those in **Appendix A**. These values can be changed by modifying the equates appropriately, with no additional change needed on the code.

Example 3-16 shows the complete assembly code for the READ_STATUS_REG routine.

Example 3-16 READ_STATUS_REG Routine Assembly Code

```
READ_STATUS_REG
        ;-------------------------------------------------------------
        ; (1) READ API DATA
        ;-------------------------------------------------------------
        clr     a
        move    #>$20,x0
        move    x:RDSR_DEST_ADD,r1
        move    x:RDSR_DEST_SPC,b
        cmp     #$70,b                          ; is it lowercase?
        sub     x0,b      ifge                  ; capitalize
        ;-------------------------------------------------------------
        ; (2) RESET SERIAL INTERFACE
        ;-------------------------------------------------------------
        bsr     SERIAL_INTERFACE_RESET
        ;-------------------------------------------------------------
        ; (3) PACK OPCODE
        ;-------------------------------------------------------------
        move    #RDSR_OPCODE,a2
        asr     #8,a,a                          ; now we have READ_OPCODE in A1
        ;-------------------------------------------------------------
        ; (4) ASSERT CHIP SELECT
        ;-------------------------------------------------------------
        movep   #$4,x:M_AAR1                    ; set AA1 low
        ;-------------------------------------------------------------
        ; (5) ACTIVATE SERIAL INTERFACE and SYNCHRONIZE
        ;-------------------------------------------------------------
        bsr     SYNCHRONIZE
        ;-------------------------------------------------------------
        ; (6) TRANSMIT OPCODE
        ;-------------------------------------------------------------
```

Example 3-16 READ_STATUS_REG Routine Assembly Code (Continued)

```
        IF SERIAL_INTERFACE=='ESSI'

        brclr      #M_RDF,x:M_SSISR0,*              ; wait until byte is TXed
                                                    ; (opcode, B1)
        movep      x:M_RX0,n5                       ; clear RDF bit

        ELSE
        IF SERIAL_INTERFACE=='SCI'

        brclr      #M_TDRE,x:M_SSR,*                ; wait until byte is TXed
                                                    ; (opcode, B0)
        movep      #$ff0000,x:M_STXH                ; keep transmitting to maintain
                                                    ; clock
        brclr      #M_RDRF,x:M_SSR,*                ; clean receiver
        movep      x:M_SRXH,a1
        nop                                         ; pipeline delay

        ENDIF
        ENDIF
        ;-------------------------------------------------------------
        ; which space?
        ;-------------------------------------------------------------
        cmp        #'X',b
        move       #(_x-_p+1),r5
        beq        _sp_end
        cmp        #'Y',b
        move       #(_y-_p+1),r5
        beq        _sp_end
        move       #$1,r5
_sp_end
        ;-------------------------------------------------------------
        ; read SR and write to DSP memory
        ;-------------------------------------------------------------
        ;-------------------------------------------------------------
        ; (7) READ STATUS REGISTER
        ;-------------------------------------------------------------
        IF SERIAL_INTERFACE=='ESSI'

        brclr      #M_RDF,x:M_SSISR0,*              ; wait until ESSI0's receiver is
                                                    ; full
        movep      x:M_RX0,a1

        ELSE
        IF SERIAL_INTERFACE=='SCI'

        brclr      #M_RDRF,x:M_SSR,*                ; read received byte
        movep      x:M_SRXH,a1

        ENDIF
        ENDIF
```

Example 3-16 READ_STATUS_REG Routine Assembly Code (Continued)

```
            ;-------------------------------------------------------------
            ; (8) PACK IT
            ;-------------------------------------------------------------
            lsr        #16,a
            ;-------------------------------------------------------------
            ; (9) WRITE TO DESTINATION
            ;-------------------------------------------------------------
            bra        r5
_p
            move       a1,p:(r1)+
            bra        _rd_end
_x
            move       a1,x:(r1)+
            bra        _rd_end
_y
            move       a1,y:(r1)+
_rd_end
            ;-------------------------------------------------------------
            ; (10) RESET SERIAL INTERFACE
            ;-------------------------------------------------------------
            bsr        SERIAL_INTERFACE_RESET
            ;-------------------------------------------------------------
            ; (11) DEASSERT CHIP SELECT
            ;-------------------------------------------------------------
            bclr       #M_BAAP,x:M_AAR1                 ; set AA1 polarity high
            ;-------------------------------------------------------------
            ; RETURN
            ;-------------------------------------------------------------
            rts
```

Example 3-17 shows an example of a READ_STATUS_REG call. The call reads the SEEPROM's Status Register and writes its value to DSP X data memory at address $300.

Example 3-17 READ_STATUS_REG Call

```
;-----------------------------------------------------------------------
; a READ_STATUS_REG call
;-----------------------------------------------------------------------
            API        RDSR_DEST_SPC,"'x'"
            API        RDSR_DEST_ADD,$300
            bsr        READ_STATUS_REG
```

3.4.3 WRITE_STATUS_REG Function

The WRITE_STATUS_REG function programs the SEEPROM Status Register with a given value. **Figure 3-11** displays the WRITE_STATUS_REG timing scheme; **Figure 3-12** shows the flowchart; **Table 3-8** lists the parameters of WRITE_STATUS_REG.

Table 3-8 WRITE_STATUS_REG Parameters

Parameter	Description	Range	API
WRSR_DATA	Data to be written into the SEEPROM Status Register	One byte, at LSB of DSP memory location	x:$7

Note: These addresses are determined by assembler equates. The values correspond to those in **Appendix A** and can be changed by modifying the equates appropriately, with no additional change needed on the code.

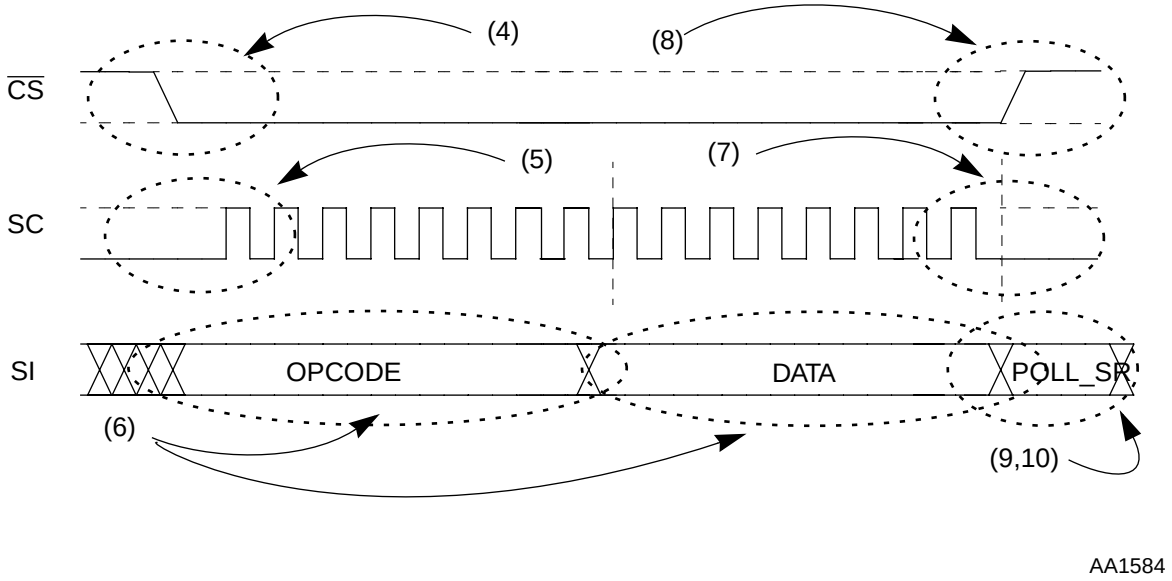

Figure 3-11 WRITE_STATUS_REG Timing

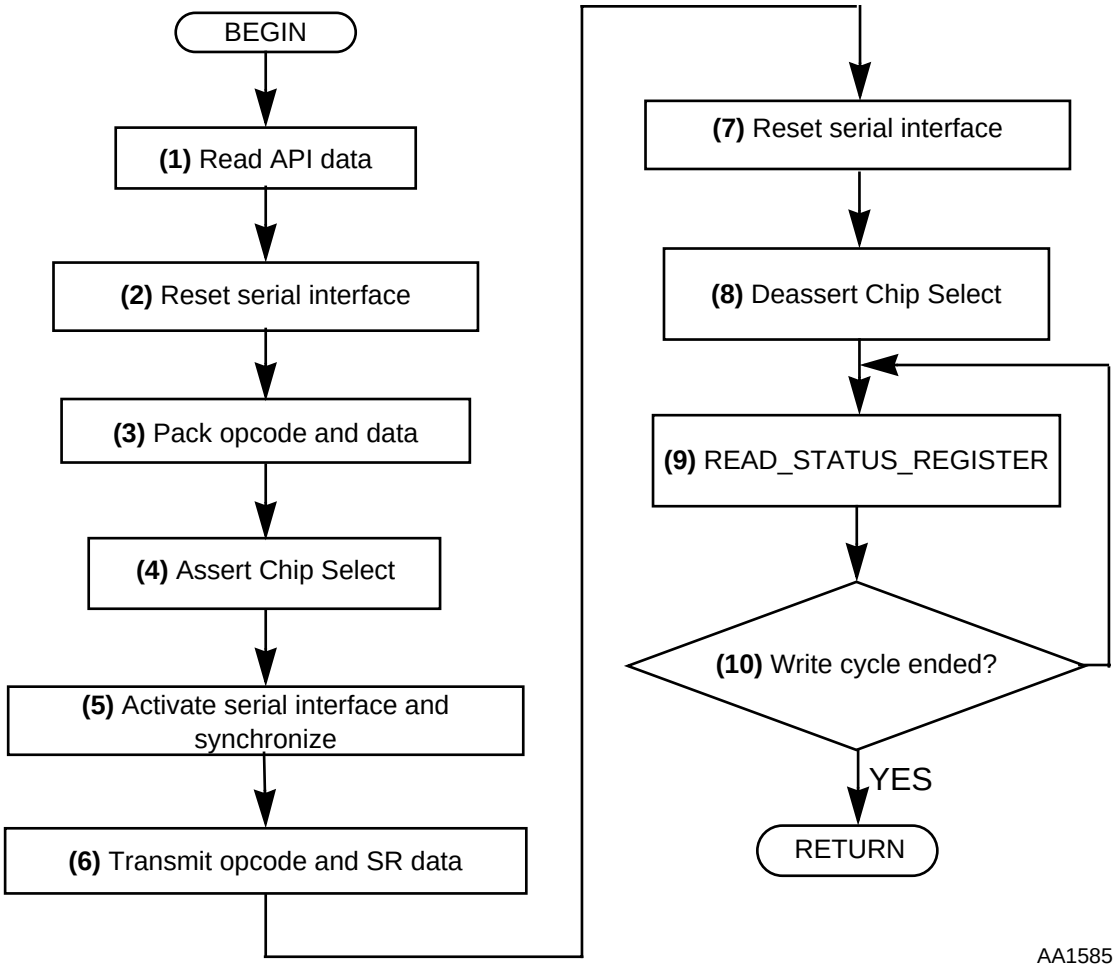

Figure 3-12 WRITE_STATUS_REG Flow-Chart

Example 3-18 presents the assembly code for the WRITE_STATUS_REG routine.

Example 3-18 WRITE_STATUS_REG Assembly Code

```
WRITE_STATUS_REG
          ;-------------------------------------------------------------
          ; Write Enable
          ;-------------------------------------------------------------
          bsr        WRITE_ENABLE
          ;-------------------------------------------------------------
          ; (1) READ API DATA
          ;-------------------------------------------------------------
          clr        a
          move       x:WRSR_DATA,a1
          ;-------------------------------------------------------------
          ; (2) RESET SERIAL INTERFACE
          ;-------------------------------------------------------------
          bsr        SERIAL_INTERFACE_RESET
          ;-------------------------------------------------------------
          ; (3) PACK OPCODE and DATA
          ;-------------------------------------------------------------
          move       #WRSR_OPCODE,a2
          asr        #8,a,a                          ; now we have WRSR_DATA in A0
                                                     ; and WRSR_OPCODE in A1
          ;-------------------------------------------------------------
          ; (4) ASSERT CHIP SELECT
          ;-------------------------------------------------------------
          movep      #$4,x:M_AAR1                    ; change AA1 polarity,in order
                                                     ; to set it low
          ;-------------------------------------------------------------
          ; (5) ACTIVATE SERIAL INTERFACE and SYNCHRONIZE
          ;-------------------------------------------------------------
          bsr        SYNCHRONIZE
          ;-------------------------------------------------------------
          ; (6) TRANSMIT OPCODE and SR DATA
          ;-------------------------------------------------------------
          IF SERIAL_INTERFACE=='ESSI'

          movep      a0,x:M_TX00                     ; load 2nd valid byte to be
                                                     ; TXed (SR data, B2)
          brclr      #M_RDF,x:M_SSISR0,*             ; wait until byte is TXed
                                                     ; (opcode, B1)
          movep      x:M_RX0,n5                      ; clear RDF bit
          brclr      #M_RDF,x:M_SSISR0,*             ; wait until byte is TXed
                                                     ;(SR data, B2)
          movep      x:M_RX0,n5                      ; clear RDF bit
```

Example 3-18 WRITE_STATUS_REG Assembly Code (Continued)

```
ELSE
IF SERIAL_INTERFACE=='SCI'

brclr      #M_TDRE,x:M_SSR,*                ; wait until byte is TXed;(opcode, B0)
movep      a0,x:M_STXH                      ; load 2nd valid byte to be TXed
                                            ; (data, B1)
brclr      #M_RDRF,x:M_SSR,*                ; clean receiver
movep      x:M_SRXH,a1
nop                                         ; pipeline delay
brclr      #M_TDRE,x:M_SSR,*                ; wait until byte is TXed
                                            ; (data, B1)
brclr      #M_RDRF,x:M_SSR,*                ; clean receiver
movep      x:M_SRXH,a1

ENDIF
ENDIF
;-------------------------------------------------------------
; (7) RESET SERIAL INTERFACE
;-------------------------------------------------------------
bsr        SERIAL_INTERFACE_RESET
;-------------------------------------------------------------
; (8) DEASSERT CHIP SELECT
;-------------------------------------------------------------
bclr       #M_BAAP,x:M_AAR1                 ; change AA1 polarity, in order
                                            ; to set it high
;-------------------------------------------------------------
; (9) READ_STATUS_REGISTER
;-------------------------------------------------------------
jsr        POLL_SR
;-------------------------------------------------------------
; (10) WRITE CYCLE ENDED?
;-------------------------------------------------------------
;-------------------------------------------------------------
; RETURN
;-------------------------------------------------------------
rts
```

Example 3-19 shows an example of a WRITE_STATUS_REG call, which writes a value of $f2 to the SEEPROM Status Register.

Example 3-19 A WRITE_STATUS_REG Call

```
;-----------------------------------------------------------------------
; a WRITE_STATUS_REG call
;-----------------------------------------------------------------------
          API        WRSR_DATA,$f2
          bsr        WRITE_STATUS_REG
```

3.4.4 WRITE Function

The typical Serial EEPROM presents a WRITE function that enables writing to the SEEPROM from one byte up to one memory page. The WRITE function is thus a subset of the WRITE_BLOCK function described in **Section 3.3.1 WRITE_BLOCK Function** on page 3-3. With appropriate calls to WRITE_BLOCK, you can run any WRITE function. For example, the following call writes one byte to address $10 in the Serial EEPROM.

Example 3-20 Single Byte WRITE Call

```
API     WR_N_SRC_N,$1
API     WR_N_SRC_SPC,"'x'"          ; The data to be written is
API     WR_N_SRC_ADD,$200           ; taken from DSP memory x:$200
API     WR_N_DEST_ADD,$10
API     WR_N_PAGE_SIZE,$3           ; The used SEEPROM has a 4-byte; page
API     WR_N_WRD_SZ,$1
bsr     WRITE_BLOCK
```

For writing a page, the call is as follows:

Example 3-21 Single Page WRITE Call

```
API     WR_N_SRC_N,$4               ; The page is 4-byte long
API     WR_N_SRC_SPC,"'x'"
API     WR_N_SRC_ADD,$200
API     WR_N_DEST_ADD,$10           ; $10 is base address of a page
API     WR_N_PAGE_SIZE,$3
API     WR_N_WRD_SZ,$1
bsr     WRITE_BLOCK
```

3.4.5 READ Function

As with the WRITE function, a READ function is available on the typical Serial EEPROM, and it permits reading of any number of bytes from the SEEPROM. Calling the READ_BLOCK function as detailed in **Section 3.3.2** runs the READ function. Since READ_BLOCK can read 8-,16- and 24-bit words, the Serial EEPROM READ function is a subset of READ_BLOCK. A one-byte SEEPROM READ access is performed, for example, with the following call:

Example 3-22 One-Byte SEEPROM READ Call

```
API     RD_N_SRC_N,$1               ; read one byte
API     RD_N_SRC_ADD,$0             ; from SEEPROM address $0
API     RD_N_DEST_SPC,"'y'"         ; write to DSP memory y:$100
API     RD_N_DEST_ADD,$100
API     RD_N_WRD_SZ,$1              ; read byte mode
bsr     READ_BLOCK
```

3.5 Auxiliary Routines

Other available auxiliary routines are SERIAL_INTERFACE_RESET, SYNCHRONIZE, and POLL_SR.

3.5.1 Serial Interface Reset

The SERIAL_INTERFACE_RESET routine puts the serial interface, either ESSI or SCI, in the Personal Reset state while programming the relevant control registers to initial values. See **Example 3-23**:

Example 3-23 SERIAL_INTERFACE_RESET Code

```
SERIAL_INTERFACE_RESET
          IF SERIAL_INTERFACE=='ESSI'

          movep   #DEFAULT_PCR,x:M_PCRC
          movep   #DEFAULT_PDR,x:M_PDRC
          movep   #DEFAULT_PRR,x:M_PRRC
          movep   #DEFAULT_CRA,x:M_CRA0
          movep   #DEFAULT_CRB,x:M_CRB0

          ELSE
          IF SERIAL_INTERFACE=='SCI'

          movep   #DEFAULT_PCRE,x:M_PCRE
          movep   #DEFAULT_PDRE,x:M_PDRE
          movep   #DEFAULT_PRRE,x:M_PRRE
          movep   #DEFAULT_SCR,x:M_SCR
          movep      #DEFAULT_SCCR,x:M_SCCR
          movep      #ACTV_PCRE_1,x:M_PCRE            ; avoid initial 1 DSP clock
                                                      ; spike at SCLK
          ENDIF
          ENDIF
          rts
```

3.5.2 Synchronize Serial Interface

The SYNCHRONIZE routine activates the serial interface in a synchronized way in order to guarantee timing constraints of the SPI protocol. **Example 3-24** shows the routine's assembly listing:

Example 3-24 SYNCHRONIZE Code

```
SYNCHRONIZE
          IF SERIAL_INTERFACE=='ESSI'

          movep     #$ffffff,x:M_TX00          ; load first byte to be TXed (dummy, B0)
          movep     #ACTV_PCR_1,x:M_PCRC       ; enable ESSI
          movep     #ACTV_CRB,x:M_CRB0         ; activate ESSI's TX and RX
          brclr     #M_TFS,x:M_SSISR0,*        ; wait until frame status bit occurs
          movep     a1,x:M_TX00                ; load 1st valid byte to be TXed
                                               ; (opcode, B1)
          brclr     #M_RDF,x:M_SSISR0,*        ; wait until byte is TXed; (dummy, B0)
          movep     x:M_RX0,n5                 ; clear RDF bit
          rep       #(CLOCK_RATIO/2)
          nop
          movep     #ACTV_PCR_2,x:M_PCRC       ; enable SCK and STD

          ELSE
          IF SERIAL_INTERFACE=='SCI'

          movep     #ACTV_PCRE_2,x:M_PCRE      ; enable SCI
          movep     a1,x:M_STXH                ; load 1st valid byte to be TXed
                                               ; (opcode, B0)
          movep     #ACTV_SCR,x:M_SCR          ; activate SCI's TX and RX

          ENDIF
          ENDIF
          rts
```

3.5.3 Poll Status Register

The POLL_SR routine polls the SEEPROM Status Register's READY bit to determine the end of a Serial EEPROM write cycle. The routine is called by the WRITE_BLOCK routine at the end of each page write of the Serial EEPROM. It is also called by the WRITE_STATUS_REGISTER routine after it writes to the SEEPROM Status Register. **Example 3-25** shows the routine's assembly listing.

Example 3-25 POLL_SR Code

```
POLL_SR
          move      #>WR_N_STAT_REG,a
          move      a,x:RDSR_DEST_ADD
          move      #>'X',a
          move      a,x:RDSR_DEST_SPC
          jsr       READ_STATUS_REG
          move      #WR_N_STAT_REG,r1
          jset      #$0,x:(r1),POLL_SR
          rts
```

3.5.4 Read Modify Write Operation

The following example performs a READ_MODIFY_WRITE operation on a 10-word block in a 1K-bit SEEPROM, followed by write protection of the block. The modification part of the operation switches the first word and the second, the third and the fourth, and so on until the ninth and tenth words.

Example 3-26 READ_MODIFY_WRITE Operation

```
;-----------------------------------------------------------------------
; Read the block from SEEPROM address $d0 to Y memory, address $100
;-----------------------------------------------------------------------
          API       RD_N_SRC_N,$a                   ; read ten words
          API       RD_N_SRC_ADD,$d0                ; from SEEPROM address $c0
          API       RD_N_DEST_SPC,"'y'"             ; write to DSP memory y:$100
          API       RD_N_DEST_ADD,$100
          API       RD_N_WRD_SZ,$3                  ; read 3-byte words
          bsr       READ_BLOCK
;-----------------------------------------------------------------------
; Modify the block
;-----------------------------------------------------------------------
          move      #$100,r0
          do        #$5,_end_modify
          move      y:(r0)+,a0
          move      y:(r0)-,a1
          move      a1,y:(r0)+
          move      a0,y:(r0)+
_end_modify
;-----------------------------------------------------------------------
; Write-unprotect the block
;-----------------------------------------------------------------------
          API       UPRF_BASE_ADD,$d0
          API       UPRF_MEM_SZ,$1
          bsr       UNPROTECT
```

Example 3-26 READ_MODIFY_WRITE Operation (Continued)

```
;-----------------------------------------------------------------------
; Write back the block
;-----------------------------------------------------------------------
          API        WR_N_SRC_N,$a
          API        WR_N_SRC_SPC,"'y'"
          API        WR_N_SRC_ADD,$100
          API        WR_N_DEST_ADD,$d0
          API        WR_N_PAGE_SIZE,$3
          API        WR_N_WRD_SZ,$3
          bsr        WRITE_BLOCK
;-----------------------------------------------------------------------
; Write-protect the block
;-----------------------------------------------------------------------
          API        PRF_BASE_ADD,$d0
          API        PRF_MEM_SZ,$1
          bsr        PROTECT
;-----------------------------------------------------------------------
; stop
;-----------------------------------------------------------------------
          nop
          nop
          nop
          stop
          nop
          nop
          nop
;-----------------------------------------------------------------------
; end of example
;-----------------------------------------------------------------
```

3.6 ESSI Timing Considerations

A set of polling routines synchronize ESSI SCLK, enabling with the first valid bit of Serial Data Out, emulating a gated clock. **Figure 3-13** and **Figure 3-14** show Synchronization events for enabling and disabling the ESSI in the worst case (a READ_BLOCK call).

To guarantee this synchronization scheme, the ratio between the SEEPROM period and DSP period should be greater than 40, which guarantees the gated clock when the ESSI SCLK is enabled, while avoiding extra forbidden clocks after the last valid bit on the ESSI SCLK is disabled.

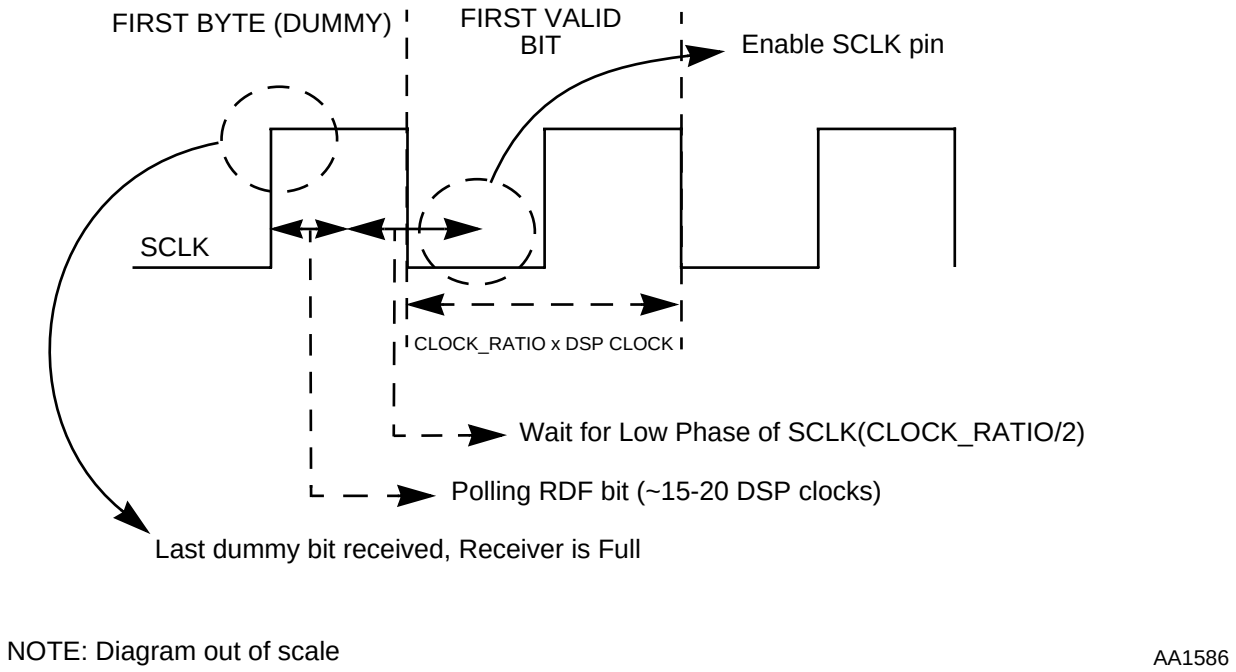

Figure 3-13 ESSI Enabling Synchronization

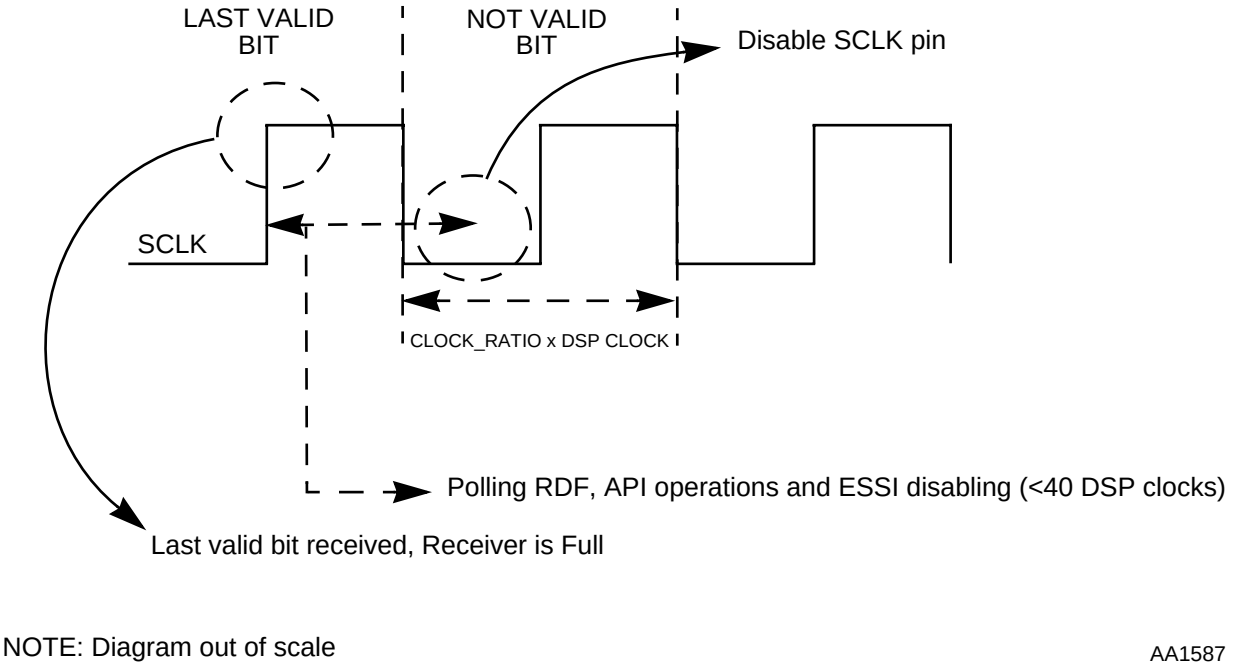

Figure 3-14 ESSI Disabling Synchronization

4 ESSI and SCI Configuration

This section details how the serial interface, whether ESSI or SCI, is configured.

4.1 ESSI Configuration

The following paragraphs describe ESSI programming.

4.1.1 ESSI Control Register A

ESSI Control Register A (CRA) is initialized with a value of **$000018**, corresponding to the following configuration

11	10	9	8	7	6	5	4	3	2	1	0
PSR				PM7	PM6	PM5	PM4	PM3	PM2	PM1	PM0
0	0	0	0	0	0	0	1	1	0	0	0
0				1				8			

23	22	21	20	19	18	17	16	15	14	13	12
	SSC1	WL2	WL1	WL0	ALC		DC4	DC3	DC2	DC1	DC0
0	0	0	0	0	0	0	0	0	0	0	0
0				0				0			

AA1588

Figure 4-2 Control Register A

- PM7-PM0 and PSR: Prescale Modulus Select and Prescale Range bits are configured so that a 1MHz serial clock is generated for a 400 MHz chip.
- DC4-DC0: Frame Rate Divider Control bits are cleared providing continuous data transfer in Normal mode.
- ALC: the Alignment Control bit is cleared, so that transmitted and received bytes would be aligned to bit 23 in the corresponding data registers.
- WL2-WL0: Word Length Control bits are programmed for 8-bit words.
- SSC1: this bit is irrelevant to the application discussed in this report.

4.2.1 ESSI Control Register B

ESSI Control Register B (CRB) is initialized with a value of **$001920**. A value of a value of **$031920** activates the ESSI.

11	10	9	8	7	6	5	4	3	2	1	0
CKP	FSP	FSR	FSL1	FSL0	SHFD	SCKD	SCD2	SCD1	SCD0	OF1	OF0
1	0	0	1	0	0	1	0	0	0	0	0
9				2				0			

23	22	21	20	19	18	17	16	15	14	13	12
REIE	TEIE	RLIE	TLIE	RIE	TIE	RE	TE0	TE1	TE2	MOD	SYN
0	0	0	0	0	0	0 -> 1	0 -> 1	0	0	0	1
0				0 -> 3				1			

AA1589

Figure 4-3 Control Register B

These values correspond to the following configurations:

- OF0 and OF1: Output Flags bits are not used in this implementation.
- SCD2-SCD0: Serial Control Direction bits are irrelevant for this application.
- SCKD: The internal clock is used, so the Clock Source Direction bit is set.
- SHFD: Data is shifted in and out MSB first, so the Shift Direction bit is cleared.
- FSL1-FSL0: Frame Sync Length bits are set to 10 (bit-length) according to the ESSI specification for continuous periodic data transfers.
- FSR and FSP: Frame Sync Relative Timing and Frame Sync Polarity bits are irrelevant for the current implementation.
- CKP: The Clock Polarity bit is set so that data is clocked out on the falling edge of bit clock and latched in on its rising edge.
- SYN: ESSI works in its synchronous mode, so the Synchronous/Asynchronous bit is set.
- MOD: Clearing the ESSI Mode Select bit selects Normal mode for the application.

- TE2 and TE1: Transmitters 2 and 1 are not used, so corresponding Transmit Enable bits are zeroed.
- TE0: the Transmit 0 Enable bit is set whenever transmitter 0 is to be used.
- RE: the Receive Enable bit is set whenever the ESSI receiver is to be used.
- REIE, TEIE, RLIE, TLIE, RIE and TIE: No interrupt is used on the application, so all the interrupt enabling bits are cleared.

4.3.1 ESSI Port Control, Direction and Data Registers

The ESSI Port Control Register (PCRC), Port Direction Register (PRRC) and Port Data Register (PDRC) mirror the pin functionality, direction, and state required in every task performed throughout the application.

Table 4-1 condenses all the combinations used.

Table 4-1 PCRC, PRRC and PDRC Values

	Pin Number	**P5**	**P4**	**P3**	**P2**	**P1**	**P0**	**HEXA**
	ESSI Function	STD	SRD	SCK	SC2	SC1	SC0	
DEFAULT	Function (PCRC)	GPIO	GPIO	GPIO	GPIO	GPIO	GPIO	00
	Direction (PRRC)	OUTPUT	OUTPUT	OUTPUT	OUTPUT	OUTPUT	OUTPUT	3F
	Data (PDRC)	1	1	0	1	1	1	37
ACTIVE	Function (PCRC) Enable ESSI	GPIO	ESSI	GPIO	GPIO	GPIO	GPIO	20
	Function (PCRC) Enable SCK and STD	ESSI	ESSI	ESSI	GPIO	GPIO	GPIO	38

4.4 SCI Configuration

The following paragraphs describe SCI programming.

4.4.1 SCI Control Register

The SCI Control Register (SCR) is initialized with a value of **$008008**. A value of **$008308** simultaneously enables both the receiver and transmitter 1.

11	10	9	8	7	6	5	4	3	2	1	0
CD11	CD10	CD9	CD8	CD7	CD6	CD5	CD4	CD3	CD2	CD1	CD0
0	0	0	0	0	0	1	1	0	0	0	1
0				3				1			

23	22	21	20	19	18	17	16	15	14	13	12
								TCM	RCM	SCP	COD
								0	0	0	0
0				0				0			

AA1591

Figure 4-5 SCI Control Register

These values correspond to the following configuration:

- REIE, STIR, TMIE, TIE, RIE and ILIE: no interrupt is used in this implementation, so these bits are programmed with zero.
- SCKP: Negative Clock Polarity is used, so this bit is set to one.
- TE: the enable bit is set when the SCI transmitter is to be used.
- RE: the Receive Enable bit is set when the SCI receiver is to be used.
- WOMS, RWU, WAKE and SBK: These bits are irrelevant in the current application.
- SSFTD: the Most Significant Bit is shifted first, so this bit should be set to one.
- WDS2-WDS0: the Word Select Bits are all zeroed, configuring SCI to its Synchronous Mode (Mode 0).

4.5.1 SCI Clock Control Register

A value of **$000031** initializes the SCI Clock Control Register (SCCR).

11	10	9	8	7	6	5	4	3	2	1	0
CD11	CD10	CD9	CD8	CD7	CD6	CD5	CD4	CD3	CD2	CD1	CD0
0	0	0	0	0	0	1	1	0	0	0	1
0				3				1			

23	22	21	20	19	18	17	16	15	14	13	12
								TCM	RCM	SCP	COD
								0	0	0	0
0				0				0			

AA1591

Figure 4-6 SCI Clock Control Register

This value corresponds to the following configuration:

- TCM and RCM: An internal clock is used, so the Transmitter Clock Mode and Receiver Clock Mode bits are zeroed.
- SCP: SCI Clock Prescaler divides by one, so this bit is set to zero.
- COD: this bit is irrelevant in Synchronous mode, since the output divider is fixed at divide by 2.
- CD11-CD0: the Clock Divider bits are programmed to **$31**, providing a 1/400 ratio between the serial clock and the DSP clock.

4.6.1 SCI Port Control, Direction, and Data Registers

The SCI Port Control Register (PCRE), Port Direction Register (PRRE), and Port Data Register (PDRE) mirror the pin functionality, direction, and state required in every task performed throughout the application.

Table 4-1 condenses all the combinations used.

Table 4-1 PCRE PRRE and PDRE Values

	Pin Number	**P2**	**P1**	**P0**	**HEXA**
	ESSI Function	**SCLK**	**TXD**	**RXD**	
DEFAULT	Function (PCRE)	GPIO	GPIO	GPIO	0
	Direction (PRRE)	OUTPUT	OUTPUT	OUTPUT	7
	Data (PDRE)	0	1	1	3
ACTIVE	Function (PCRE) Enable SCI	GPIO	GPIO	SCI	1
	Function (PCRE) Enable SCLK and TXD	SCI	SCI	SCI	7

5 Customization

This section explains how to customize the routines provided in the earlier sections.

5.1 Code Optimization

Much of the assembly code provided in Section 3 is consumed in processing API parameters and in running API routines. This includes DSP memory space selection, Serial EEPROM page management, word packing, and so on. Although it gives great transparency and flexibility, the code needed for performing these features consumes DSP cycles and program memory, which can be optimized by customizing all the API functions for the specific user's application. Below is a list of some features that can be customized to reduce DSP memory and processing cycles. The numbers in parentheses address to correspondent steps on the WRITE_BLOCK routine where the applicable changes should be made. Refer to the function code in **Section 3.3.1**.

- Using a fixed DSP memory space permits the DSP memory space processing from any function to be cut (1,6-7).
- Using a fixed word size or/and fixed SEEPROM addresses permits reduce packing routines (1,3,8,9).
- Isolated use of some functions may permit you to avoid branching to common routines (1-2,5,11,13).
- If you do not need to write large blocks, the page management mechanism can be extracted (1,7,8,10,14,15).
- In case some functions that are called one after another, SERIAL_INTERFACE_RESET can be ignored at the beginning of any function after the second one (2).

5.2 Larger-Capacity Serial EEPROM

The application discussed in this report provides a one-byte addressing mechanism that covers any Serial EEPROM up to 2K-bit density. Serial EEPROMs of greater density use an additional address line for accessing memory locations higher than $FF. This additional address bit is usually one of the unused bits of the one-byte opcode and is device-dependent. If your application needs to access larger-density Serial EEPROMs, you should modify some code in order to fit the present addressing routines to the selected device. We suggest modifying the code on its highest level, at the API subroutines call, with minor low-level routine changes for opcode and address determination.

Appendix A Assembly Equates

This Appendix presents the assembly code and defined equates for this application.

A.1 EQUATES

In addition to the I/O and Interrupt Equates of the respective DSP56300 derivatives, the AC-link application assembly code uses the equates defined below.

A.1.1 General Equates

```
;-----------------------------------------------------------------------
;          General
;-----------------------------------------------------------------------
START                   equ        $100                  ; Main Program Starting Address
CLOCK_RATIO             equ        $190                  ; ratio (EEPROM period / DSP
                                                         ; period = 1/400)
SERIAL_INTERFACE        equ        "ESSI"                ; This equate should determine
                                                         ; which Serial Interface will
                                                         ; be used on the connection. Set
                                                         ; it to “SCI” in case it is
                                                         ; intended to be used as Serial
                                                         ; Interface
```

A.1.2 ESSI Configuration Equates

```
;-----------------------------------------------------------------------
;           ESSI EQUATES
;-----------------------------------------------------------------------
DEFAULT_PCR             equ        $000000
DEFAULT_PRR             equ        $00003F
DEFAULT_PDR             equ        $000037
DEFAULT_CRA             equ        $000018
DEFAULT_CRB             equ        $001920
;-----------------------------------------------------------------------
ACTV_CRB                equ        $031920               ; TX & RX enabled
ACTV_PCR_1              equ        $000020               ; ESSI activation
ACTV_PCR_2              equ        $000038               ; SCK and STD enabling
```

A.1.3 SCI Configuration Equates

```
;-----------------------------------------------------------------------
;           SCI EQUATES
;-----------------------------------------------------------------------
DEFAULT_PCRE          equ        $000000
DEFAULT_PRRE          equ        $000007
DEFAULT_PDRE          equ        $000003
DEFAULT_SCR           equ        $008008
DEFAULT_SCCR          equ        $000031              ; 400MHz (400   : 8 : 1 : 50)
                                                      ; = 1MHz
;-----------------------------------------------------------------------
ACTV_SCR              equ        $008308              ; TX & RX enabled
ACTV_PCRE  _1         equ        $000001
ACTV_PCRE  _2         equ        $000007
```

A.1.4 SEEPROM Opcodes

```
;-----------------------------------------------------------------------
;           EEPROM OPCODES
;-----------------------------------------------------------------------
WRSR_OPCODE           equ        $01
WRITE_OPCODE          equ        $02
READ_OPCODE           equ        $03
WRDI_OPCODE           equ        $04
WREN_OPCODE           equ        $06
RDSR_OPCODE           equ        $05
```

A.1.5 API

```
;-----------------------------------------------------------------------
;           API
;-----------------------------------------------------------------------
; RD_N :    Read N words from EEPROM
;-----------------------------------------------------------------------
RD_N_SRC_N          equ        $0                    ; number of words to be read
                                                     ; from SEEPROM
RD_N_SRC_ADD        equ        $1                    ; SEEPROM address for MSB byte
                                                     ; of first word
RD_N_DEST_SPC       equ        $2                    ; Destination memory space
RD_N_DEST_ADD       equ        $3                    ; Destination memory address
RD_N_WRD_SZ         equ        $4                    ; word size em bytes
;-----------------------------------------------------------------------
; RDSR :    Read STATUS REGISTER from EEPROM
;-----------------------------------------------------------------------
RDSR_DEST_SPC       equ        $5
RDSR_DEST_ADD       equ        $6
;-----------------------------------------------------------------------
; WRSR :    Write STATUS REGISTER to EEPROM
;-----------------------------------------------------------------------
WRSR_DATA           equ        $7
;-----------------------------------------------------------------------
; WR_N :    Write a block of N DSP words to EEPROM
;-----------------------------------------------------------------------
WR_N_SRC_N          equ        $8                    ; number of words to be written
                                                     ; to EEPROM
WR_N_SRC_SPC        equ        $9
WR_N_SRC_ADD        equ        $a                    ; DSP first word address
WR_N_DEST_ADD       equ        $b                    ; EEPROM address for LSB byte
                                                     ; of first word
WR_N_PAGE_SIZE      equ        $c                    ; (page size -1) for used
                                                     ; EEPROM
WR_N_WRD_SZ         equ        $d
WR_N_STAT_REG       equ        $e                    ; dest to STATUS REG polling
WR_N_COUNTER        equ        $f
;-----------------------------------------------------------------------
; PRF :     Write protect above address
;-----------------------------------------------------------------------
PRF_BASE_ADD        equ        $10
PRF_MEM_SZ          equ        $11
;-----------------------------------------------------------------------
; UPRF :    Write unprotect above address
;-----------------------------------------------------------------------
UPRF_BASE_ADD       equ        $12
UPRF_MEM_SZ         equ        $13
```

Appendix B Assembly Equates

The following manuals, which may contain data pertinent to this application, can be viewed or downloaded at the indicated web sites.

- http://www.mot.com/SPS/DSP/documentation/DSP56300.html
  - DSP56300 Digital Signal Processor Family Manual
  - DSP563xx Digital Signal Processor User's Manual
  - DSP563xx Digital Signal Processor Data Sheet
  - Application note APR20/D, *DSP56300 / DSP56600 Application Optimization for the Digital Signal Processors*
- http://www.st.com
  - ST95040, ST95020, ST95010 Data Sheets
- http://www.national.com
  - NM25C020 Data Sheet

Order by this number: APR38/D

How to reach us:

USA/Europe/Locations Not Listed:
Motorola Literature Distribution
P.O. Box 5405
Denver, Colorado 80217
1 (800) 441-2447
1 (303) 675-2140

Asia/Pacific:
Motorola Semiconductors H.K. Ltd.
8B Tai Ping Industrial Park
51 Ting Kok Road
Tai Po, N.T., Hong Kong
852-26629298

Japan:
Nippon Motorola Ltd
SPD, Strategic Planning Office
4-32-1, Nishi-Gotanda
Shinagawa-ku, Tokyo 141, Japan
81-3-5487-8488

Motorola Fax Back System (Mfax™):
TOUCHTONE (602) 244-6609
1 (800) 774-1848
RMFAX0@email.sps.mot.com

Technical Resource Center:
1 (800) 521-6274

DSP Helpline
dsphelp@dsp.sps.mot.com

Internet:
http://www.motorola-dsp.com/

MOTOROLA