Order this document
by AN1227/D
Rev. 1.0

Motorola Semiconductor Application Note

AN1227

Using 9346 Series Serial EEPROMs with 6805 Series Microcontrollers

By William G. Grim

Abstract

This application note describes how the HC05 Family of microcontrollers (MCU) can be used with 93 x 6 series serial electrically erasable programmable read-only memories (EEPROMs). The MCUs are made by various manufacturers such as National Semiconductor, SGS Thompson, Catalyst, and Microchip. This series includes serial EEPROMs whose base numbers are 9346, 9347, 9356, 9357, 9366, 9367, 32C101, and 33C102. These EEPROMs are based on a loose standard; however, commands to initiate the basic functions are identical. This application note also can be helpful using I^2C EEPROMs when they are used in conjunction with the Motorola application note *Interfacing the MC68HC05C5 SIOP to an I^2C Peripheral* (AN1066/D) by Naji Naufel.

MOTOROLA

Introduction

Serial EEPROMs have become an inexpensive way to maintain small amounts of non-volatile data in microcontroller systems during power off. They commonly come in 1-K (128 x 8), 2-K (256 x 8), and 4-K (512 x 8) sizes. Unlike flash memory chips, they do not take special voltages, but on average they do require 4 milliseconds (ms) to execute each word-write operation.

Several series of serial EEPROMs are available. This application note describes a method to use 9346 series serial EEPROMs with HC05 Family microcontrollers. The 9346 series uses a serial 3-wire interface. Along with chip select (CS), the three communications wires are clock (CLK), data out (DO), and data in (DI).

In this application note, all seven basic 9346 commands are described in **Table 2** and source code is included in **Appendix H**, **Appendix I**, and **Appendix J**. These seven commands are erase enable (EWEN), erase disable (EWDS), write (WRITE), erase all (ERAL), write all (WRAL), erase a memory location (ERASE), and read a memory location (READ).

Different software algorithms that use serial EEPROMs are included. The first method uses polling and ordinary input/output (I/O) lines. The second method uses the serial peripheral interface (SPI) and polls it for status. The third method also uses the SPI communications port, but obtains status by using the SPI interrupt.

The first method of polling port pins requires four I/O lines; three of them can be shared with other peripherals. Three memory locations are also used. These locations can be shared by other tasks, also. This is a more appropriate implementation when reading and writing the EEPROM occurs infrequently or when a low-cost member of the HC05 Family is used.

The second implementation differs from the first because it uses the SPI and polls it for status. All of the bit shifting done in software in the first application is done by the SPI hardware in the second method.

The third implementation uses the SPI and the SPI interrupt to save processing time during WRITE. This is an appropriate approach when writing occurs frequently or when the processor cannot be occupied in a loop for the 4-ms interval required for each byte write.

Because differences exist among vendors, options to look for in 9346 series EEPROMs are described in the following section. The included source code in **Appendix H**, **Appendix I**, and **Appendix J** contains assembler switches to handle the various types of 9346 EEPROMs.

This application has been tested with EEPROMs made by Microchip, National Semiconductor, SGS Thompson, and ICT. The test used an M68HC05EVM evaluation module with an MC68HC705C8P C8-resident processor that was assembled using the P&E assembler, IASM05.

Available EEPROM Options

As of this writing, four base numbers of 9346 series EEPROMs exist representing four different sizes. Most manufacturers also offer versions that are autosequencing and autoerase. Packages typically are 8-pin dual-in-line packages (DIPs) or small outline integrated circuits (SOICs).

The oldest member of the 93 x 6 series is the 9306 EEPROM, which is not supported by this application note. The 9346 EEPROM has a 1-Kbit capacity, the 9356 EEPROM has a 2-Kbit capacity, and the 9366 EEPROM has a 4-Kbit capacity. Minor differences exist in the programming of these EEPROMs. The only direct replacement is a 9366 EEPROM for a 9356 EEPROM.

Older EEPROMs required erasure of each memory location before rewriting. Those that do not require erasure are autoerase EEPROMs, which can be programmed more quickly.

EEPROMs are now available in 3-volt versions and are ideal for applications that require memory retention during battery changes. Three-volt and 5-volt versions program in the same way.

Modes of EEPROM Operation

Serial EEPROMs have two formats and seven basic commands. EEPROMs can operate in an 8- or 16-bit format. This format is configured either by connecting the ORG pin to V_{CC} for a 16-bit format or by connecting the ORG pin to V_{SS} for an 8-bit format. Another option is to order the EEPROM from the factory preconfigured to the desired format. In the latter case, the ORG pin is not used.

Table 1 describes the seven EEPROM commands: erase enable (EWEN), erase disable (EWDS), write (WRITE), erase all (ERAL), write all (WRAL), erase a memory location (ERASE), and read a memory location (READ).

If an EEPROM is autosequencing, subsequent bits beyond the addressed cell will be read as long as the EEPROM is selected and clocks continue. EWEN, EWDS, and READ have no ready cycle. The EEPROM is ready for a new command immediately after any of these commands are executed. WRITE, WRAL, ERASE, and ERAL require that the EEPROM is opened by an EWEN operation and not subsequently closed by an EWDS operation. Although writing and erasing commands are limited by the writing cycle time, the time taken to read is limited only by microprocessor clock speed or the 1-MHz maximum EEPROM clock speed.

Table 1. Serial EEPROM Commands

Commands	Function	Description
EWEN	Erase write enable	Opens the EEPROM for writing or erasure
EWDS	Erase write disable	Write protects the EEPROM (power-on default)
WRITE	Writes a byte or word	Writes a byte in 8-bit format or word in 16-bit format to a specific memory location; this takes about 4 ms per word
WRAL	Write all	Writes the same byte or word to all EEPROM locations; this takes about 30 ms.
ERASE	Erases a location	Erases the addressed memory location; this takes about 4 ms
ERAL	Erase all	Erases the entire EEPROM; this takes about 15 ms
READ	Reads addressed cell	Reads the addressed memory location

Hardware Description

Two schematics, **Figure 1** and **Figure 2**, show the hardware configurations used to test the attached source code in **Appendix H**, **Appendix I**, and **Appendix J**. An MC68HC05EVM was used to test both designs with an HC705C8P resident processor. Any Motorola MCU or development system that can execute SPI code or I/O code can be used to test the design.

Appendix H (POL9346.asm) is used with **Figure 2**.

Appendix I and **Appendix J** (SPIP9346.asm and SPI9346.asm) are used with **Figure 1**.

The switch is for switching the EEPROM between 8- and 16-bit formats. In actual applications, the switch is replaced by a hard wire jumper to configure the EEPROM permanently for 8- or 16-bit operation.

In the polling application, ordinary I/O lines are used. Port A bit 0 and port C bits 5 and 6 are outputs. Port C bit 7 is an input. When port A bit 0 is low, the other ports are available for other services.

In the SPI application, the SPI is configured as a master. The SPI handles all communications with the EEPROM. Port A bit 5 handles chip select. When the EEPROM is not selected, the SPI is available for other services.

Port A bit 4 is used to keep the $\overline{SS}$ line in its inactive high state.

Source Code Description

The source code in **Appendix H**, **Appendix I**, and **Appendix J** was developed using the P&E assembler and a Motorola M68HC05EVM with a C8-resident processor. The EEPROM erased state is $FF. The software will invert all reads and writes to the EEPROM device. In other words, when writing $00 to the EEPROM, the software automatically will invert $00 to $FF before writing to the device.

The maximum clock frequency of the EEPROM is 1 MHz. For HC05 bus clock frequencies above 2 MHz, the CLOCK, EESEND, and RECEIVE subroutines that are used need to be adjusted with NOP commands or the SPI baud rate must be kept below 1 MHz.

Source code was developed to work with 9346 EEPROMs in an 8-bit configuration and 9346, 9356, and 9366 EEPROMs in the 16-bit configuration. Source code can handle newer EEPROMs that can erase the previous data automatically and those that can sequence to the next EEPROM memory location automatically.

To adapt the source code to a particular EEPROM and configuration, SET the configuration used, SETNOT the others, and assemble. **Table 2** shows how to handle the software switches.

Table 2. Software Switch Options

SWITCH	SWITCH OFF (#SETNOT)	SWITCH ON (#SET)
9346FORM8	One of the other FORM switches may be used.	Use with 9346 EEPROMS configured for bytes of data (ORG pin tied to V_{SS})
9346FORM16	One of the other FORM switches may be used.	Use with 9346 EEPROMS configured for words of data (ORG pin tied to V_{CC})
9356FORM16	One of the other FORM switches may be used.	Use with 9356 EEPROMS configured for words of data (ORG pin tied to V_{CC})
9366FORM16	One of the other FORM switches may be used.	Use with 9366 EEPROMS configured for words of data (ORG pin tied to V_{CC})
AUTOERASE	The software will erase an EEPROM location before writing data to it.	The software will NOT erase an EEPROM location before writing data to it
AUTOSEQ	In block READs, the software sends an address to the EEPROM for each address to be read.	In block READs, the software sends an address to the EEPROM only once for the first address to be read. The EEPROM automatically sequences to the next location to be read.

First Application: Appendix H I/O Polling to EEPROM Application Source

In the polling application, I/O lines are toggled by software to send the clocks, chip-selects, and data. Addresses are sent using the EESEND subroutine. Clocks are sent using a multi-entry CLOCK# routine. The read routines call a RECEIVE routine. RECEIVE uses the characteristic of the BRSET command, which copies the bit tested to the carry.

The first routine, WAIT, contains the loop where the microcontroller waits during writing and erasure until the EEPROM write cycle finishes.

Reading or writing:

1. Load location ee_start with the address where the block will start in the EEPROM. It is an EEPROM address.
2. Load location mem_start with the address where the block will start in the HC05 memory space.
3. Load location stor_len with the length in bytes of the block to be read or written.
4. Call the subroutine READ (or AUTORD) or WRITE.

To execute an ERASE command, perform these steps:

1. Load the accumulator with the address to be erased in the EEPROM. It is an EEPROM address.
2. Call the ERASE routine jsr ERASE.

To execute a WRAL command, perform the following steps:

1. Load the accumulator with the immediate value to be written to every byte of the EEPROM.
2. If the EEPROM is configured to read and write words, load the X register with the least significant byte of the word to be written. The value in the accumulator will be written to the most significant byte.
3. Call the WRAL routine jsr WRAL.

To execute an ERAL command, just call the ERAL routine jsr ERAL.

In the source code printouts in **Appendix H**, **Appendix I**, and **Appendix J**, calling examples are given under the area labeled START – Sample calling of routines.

- For reading, start at STARTRD
- For writing, start at STARTWR
- For erasing location 5, start at STARTERSE
- To write a $A5 or $A5C3 to every memory location in the EEPROM, start at STARTWRL

Second Application: Appendix I SPI Polling to EEPROM Application Source

In the SPI polling application, all CLOCK# routines have been eliminated and replaced with the SPI, which eliminates the need for them. The RECEIVE routine is merged into the EESEND routine. The WAIT and EESEND routines are changed to read and write the SPI by polling it for its condition. Clocks and data are shifted in and out by the special circuitry of the SPI.

The SPI polling application is used in a manner identical to the preceding I/O polling application.

Third Application: Appendix J SPI to EEPROM Using Interrupt Application Source

In the SPI application, the WAIT routine is eliminated entirely and the SPI periodically interrupts to check the EEPROM ready status. For reading, the SPI is polled exactly like the second application. Clocks and data are shifted in and out by the special circuitry of the SPI.

The SPI interrupt-driven application uses four memory locations. A WRBLOCK macro has been written to make writing blocks to EEPROM easier. The reader is left to write macros for the other functions.

To execute a READ or WRITE command, perform these steps:

1. Execute the CK_CLR subroutine, jsr CK_CLR. This will not allow the READ to proceed until any pending WRITE, WRAL, ERASE, or ERAL finishes.
2. Load location ee_start with the address where the block will start in EEPROM. It is an EEPROM address.
3. Load location mem_start with the address where the block will start in the HC05 memory space.
4. Load location stor_len with the length in bytes of the block to be read or written.
5. Call the READ, AUTORD, or WRITE subroutine.

To execute an ERASE command, perform these steps:

1. Execute the CK_CLR subroutine, jsr CK_CLR . This will not allow the READ to proceed until any pending WRITE, WRAL, ERASE, or ERAL finishes.
2. Load location ee_start with the address where the block to be erased will start in EEPROM. It is an EEPROM address.
3. Load location stor_len with the length in bytes of the block to be erased.
4. Call the ERASE routine jsr ERASE.

To execute a WRAL command, perform these steps:

1. Load the accumulator with the immediate value to be written to every byte of the EEPROM. If the EEPROM is configured to read and write words, the value in the accumulator will be written to the more significant byte.
2. Execute the CK_CLR subroutine, jsr CK_CLR. This subroutine will not allow the WRAL to proceed until any pending WRITE, WRAL, ERASE, or ERAL finishes.
3. Call the WRAL routine jsr WRAL.

To execute an ERAL command, perform these steps:

1. Execute the CK_CLR subroutine, jsr CK_CLR. This will not allow the WRAL to proceed until any pending WRITE, WRAL, ERASE, or ERAL finishes.
2. Call the ERAL routine jsr ERAL.

In the source code printouts in **Appendix H**, **Appendix I**, and **Appendix J**, calling examples are given under the area labeled START – `Sample calling of routines`.

- For reading, start at STARTRD.
- For writing, start at STARTWR.
- For erasing locations 5, 6, and 7, start at STARTERSE.
- To write a $A5 or $A5C3 to every memory location in the EEPROM, start at STARTWRL.

Common Problems

The most common EEPROM problem is that it will not be accessible after writing or erasing.

This list describes additional EEPROM problems:

1. Not erasing an EEPROM that does not have the autoerase feature. Most EEPROMs now have autoerase; however, some older designs do not have this feature. SETNOT the autoerase switch and re-assemble.
2. Interference in the WRITE command by another task, such as a task that shares the SPI or I/O lines. For the EEPROM to respond properly to a command, that command must be received in the correct order of bits. Delays are allowable, but stray bits are not.

3. Not having the correct assembler switches set, such as programming a 9346 EEPROM as a 9356 EEPROM. A 9346 EEPROM requires a different number of clocks than the 9356 and 9366 EEPROMs. Form 8 and form 16 configurations take different numbers of clocks, also. If the number is not right, the EEPROM will not come ready.
4. Some EEPROMs have a ready-disable mode triggered by writing a high to the DI line when selected. Avoid this operation.
5. Some EEPROMs, such as the SGS Thompson version of the EEPROMs, do not support ERASE or ERAL. Because these EEPROMs are autoerase, this function is never needed. Any attempt to write an ERASE or ERAL command to these EEPROMs will cause them to not come ready.
6. After a WRITE, ERAL, or WRAL instruction is sent, an inquiry of status is required. This is done merely by reselecting the EEPROM. The software does this in the WAIT routine.

Another problem is caused by interrupts. Interrupt problems are described in the following list:

1. Interrupts can change memory locations during block writes. The result can be an inconsistent collection of values saved to EEPROM. When the values are read back, the HC05 program may crash. Be careful with interrupts, especially during write operations. EEPROMs can take up to 15 ms to write a large block of data, a long time on a microcontroller scale.
2. A similar but potentially more damaging problem is the one created by powering down during a write cycle. A designer might have shipped a product only to find that this problem occurs on rare occasions. This problem can be more easily solved than the interrupt problem cited above by making two copies, each with an age tag. This task may seem wasteful, but it will ensure that at least one usable copy will be available for the next power-up, if the other copy was in the process of updating.

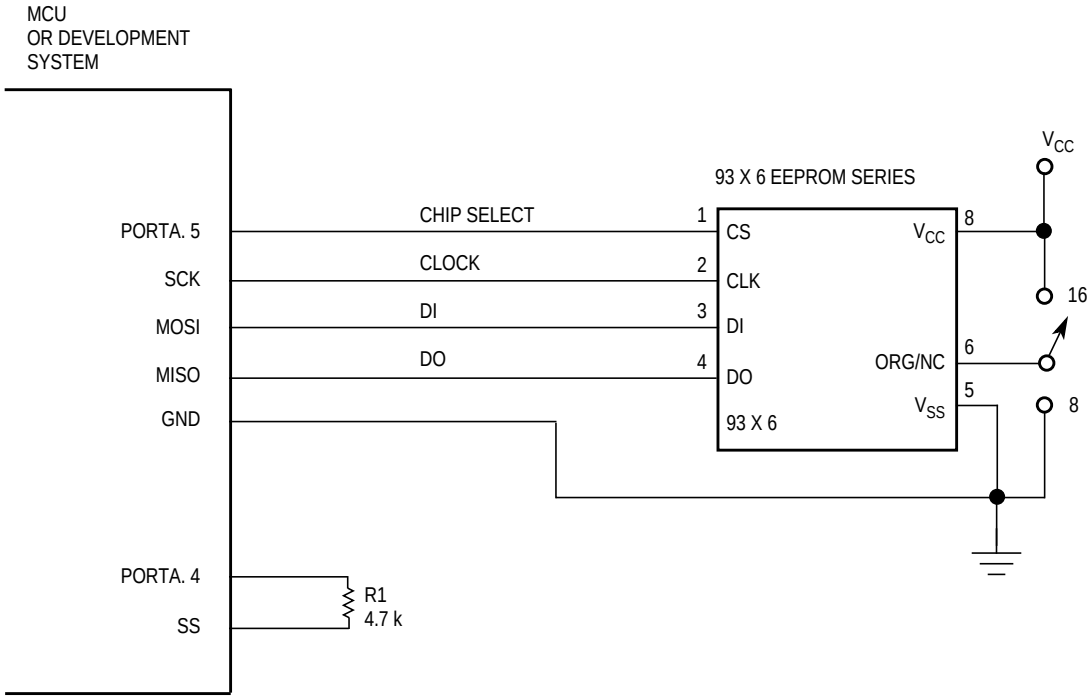


Figure 1. SPI to EEPROM Connection

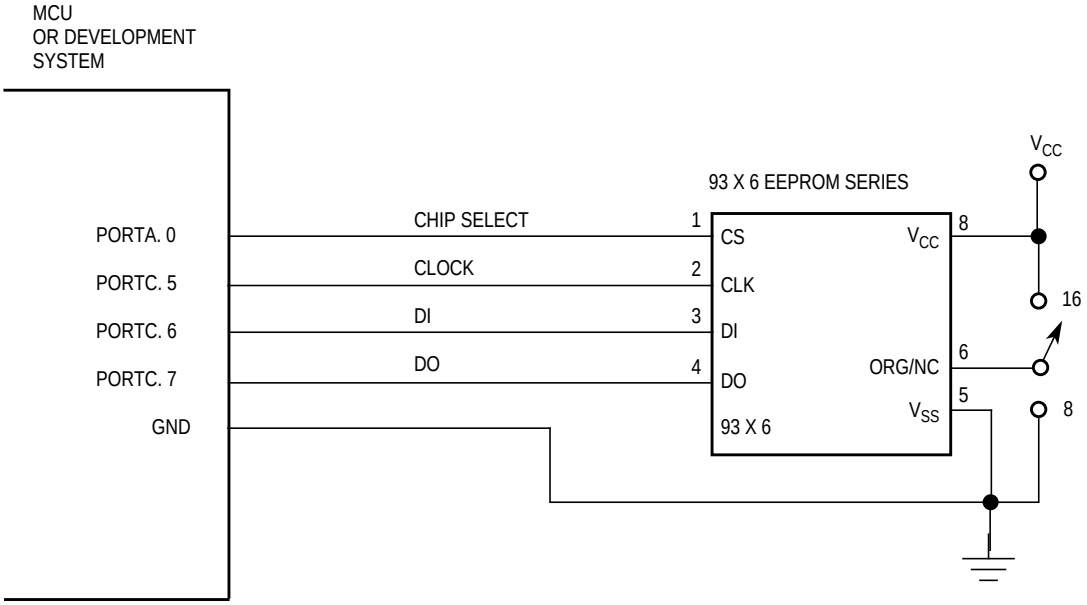
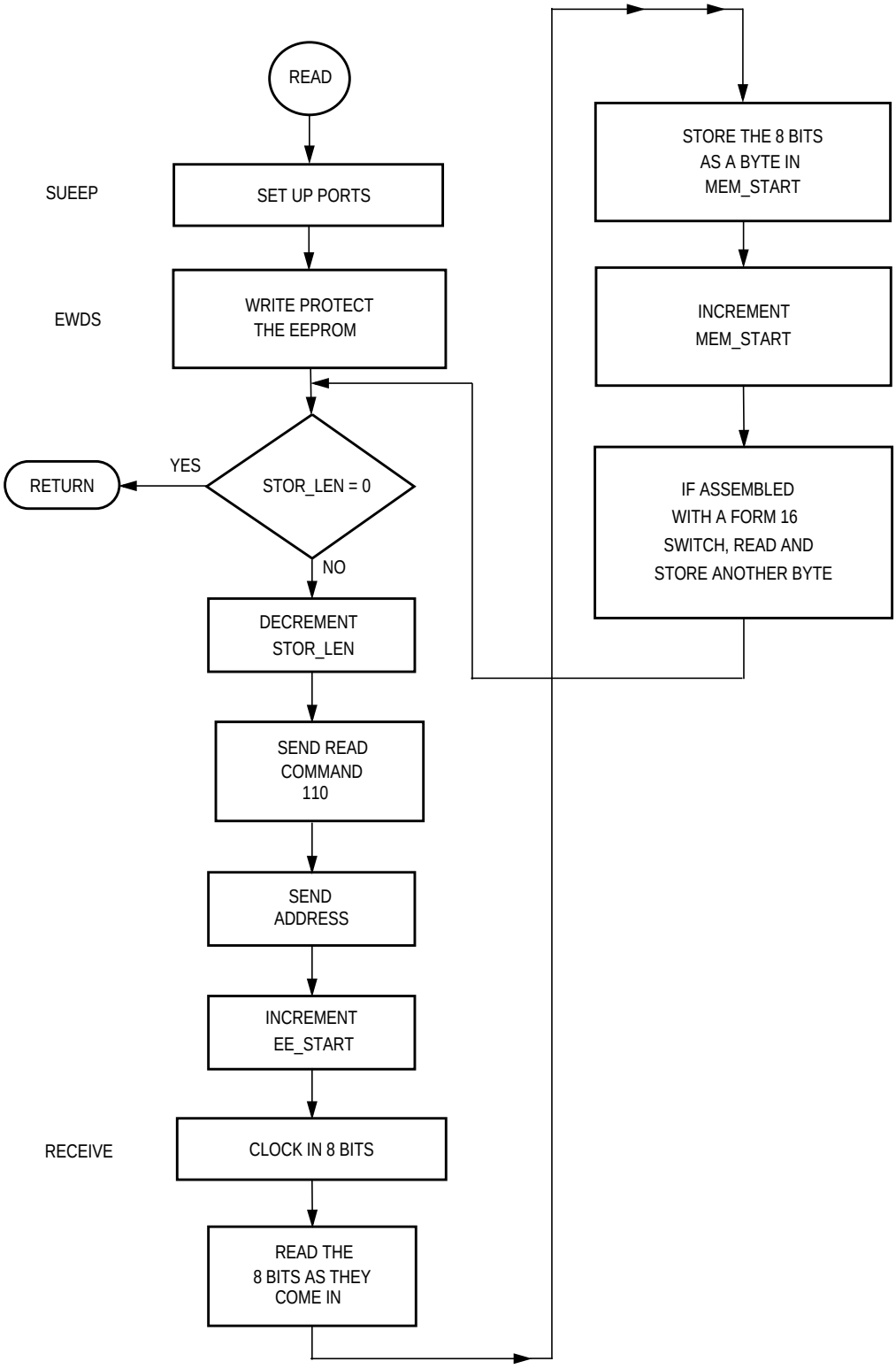
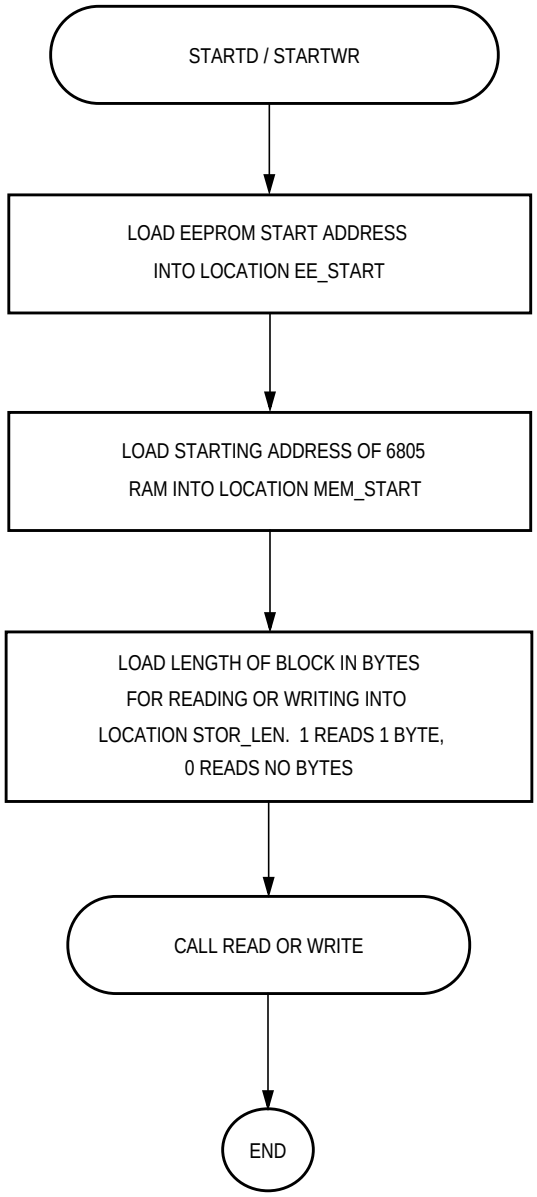


Figure 2. I/O Lines to EEPROM Schematic

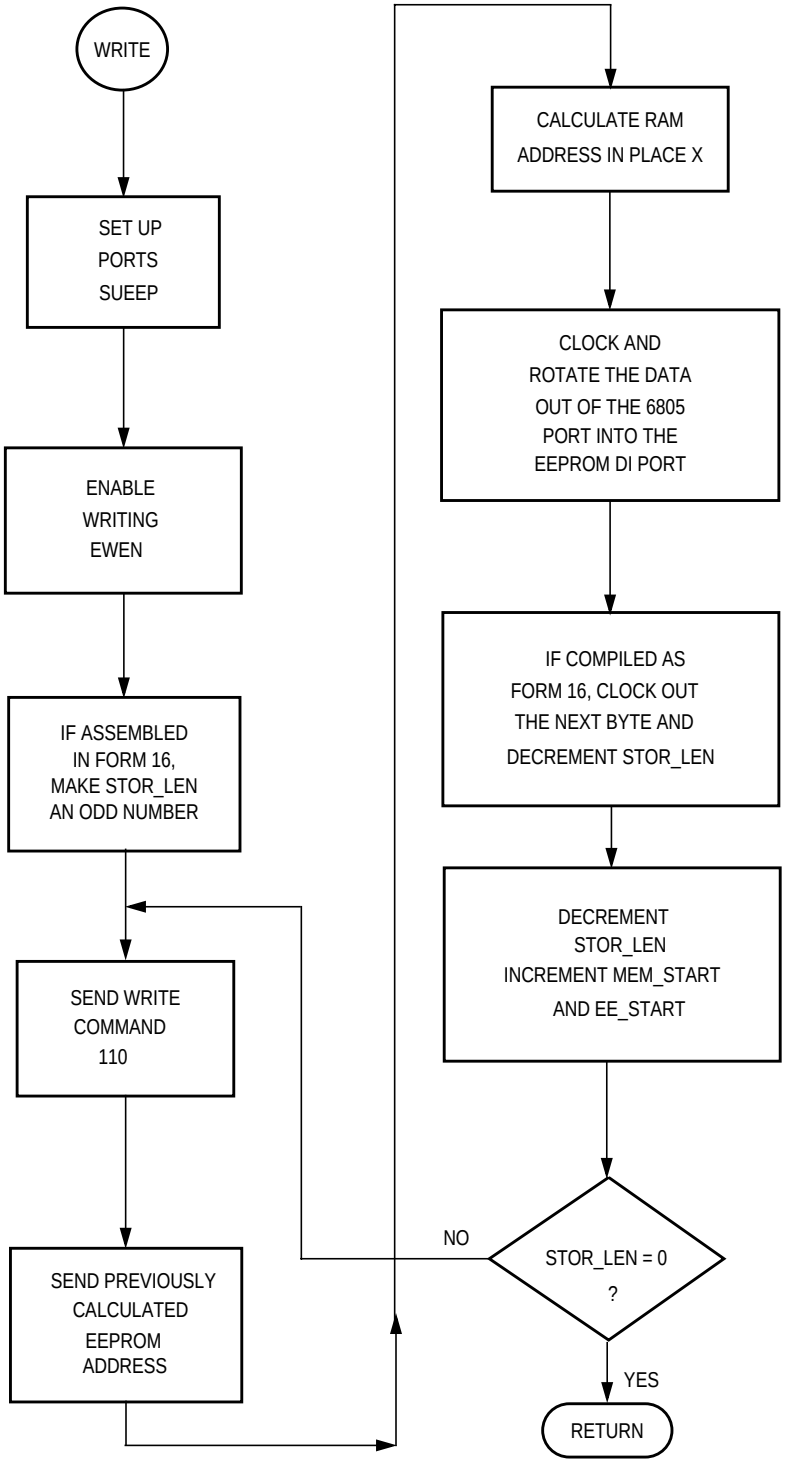
Appendix A — READ Application Flowchart



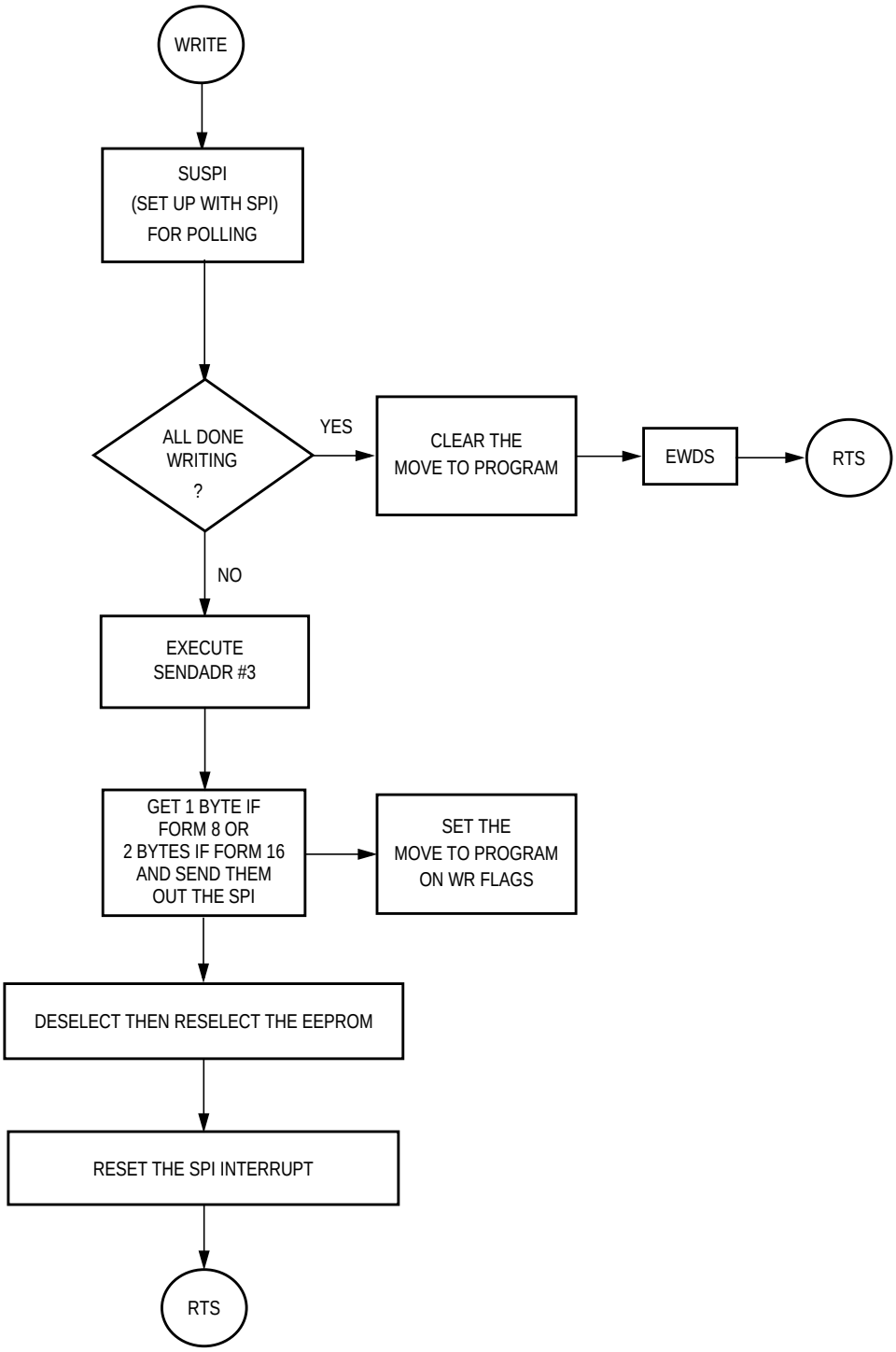
Appendix B — Application Calling Reading or Writing Flowchart



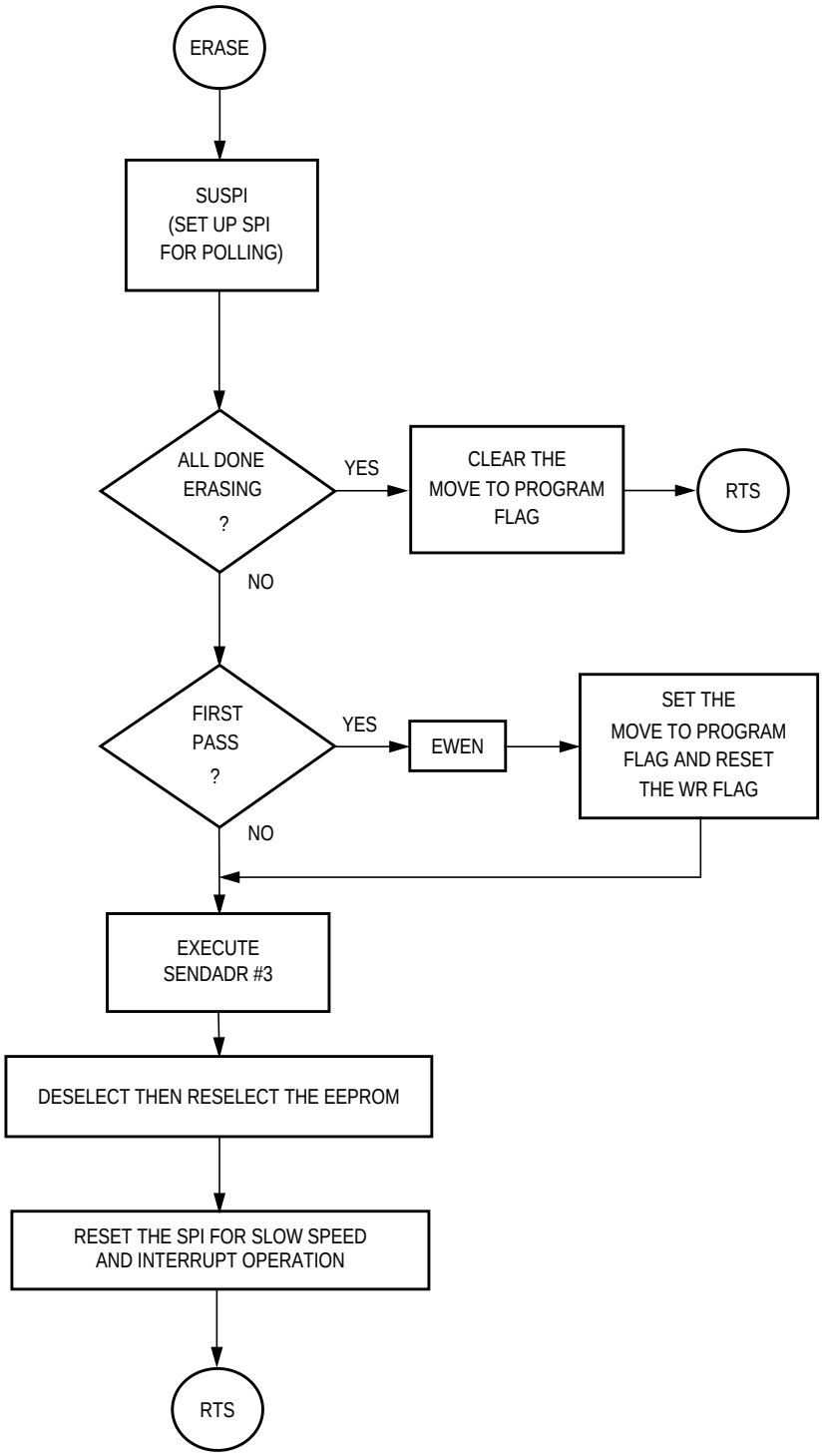
Appendix C — I/O and SPI Polling Application Flowchart



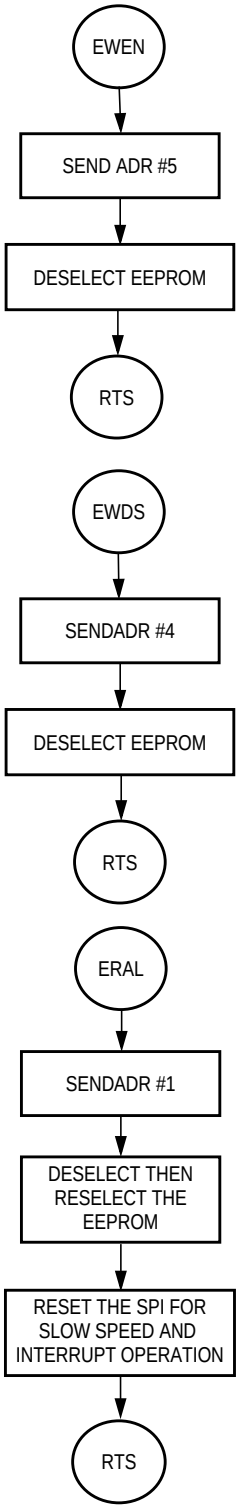
Appendix D — SPI Interrupt WRITE Application Flowchart

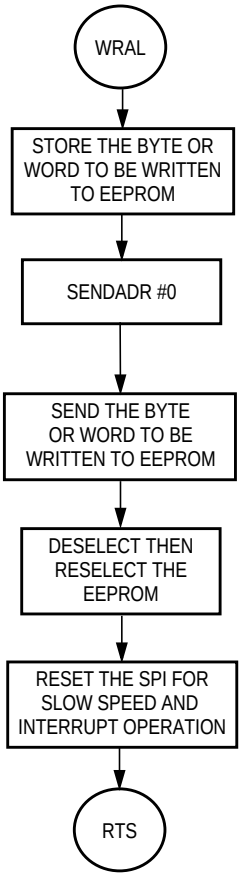


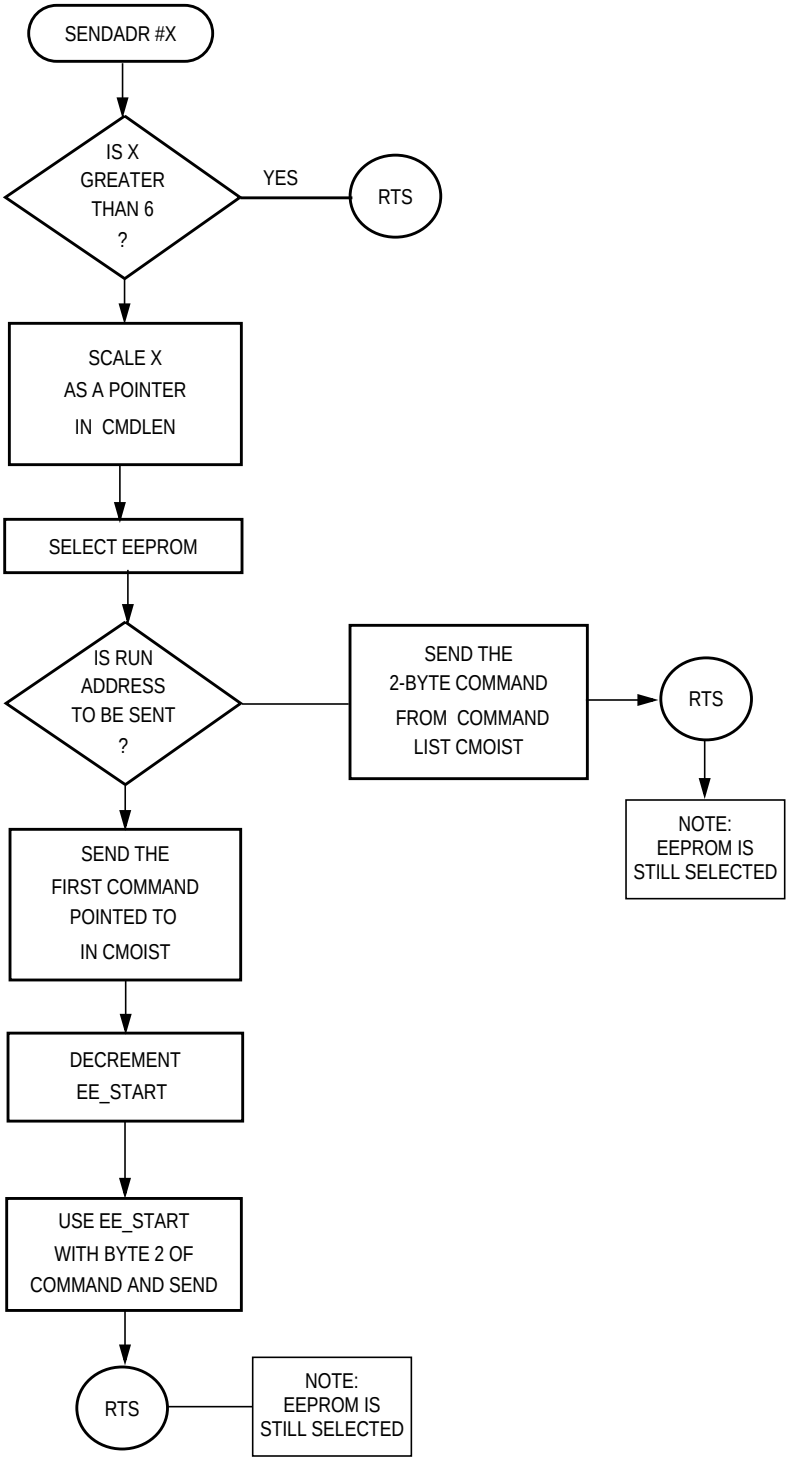
Appendix E — SPI Interrupt ERASE Application Flowchart

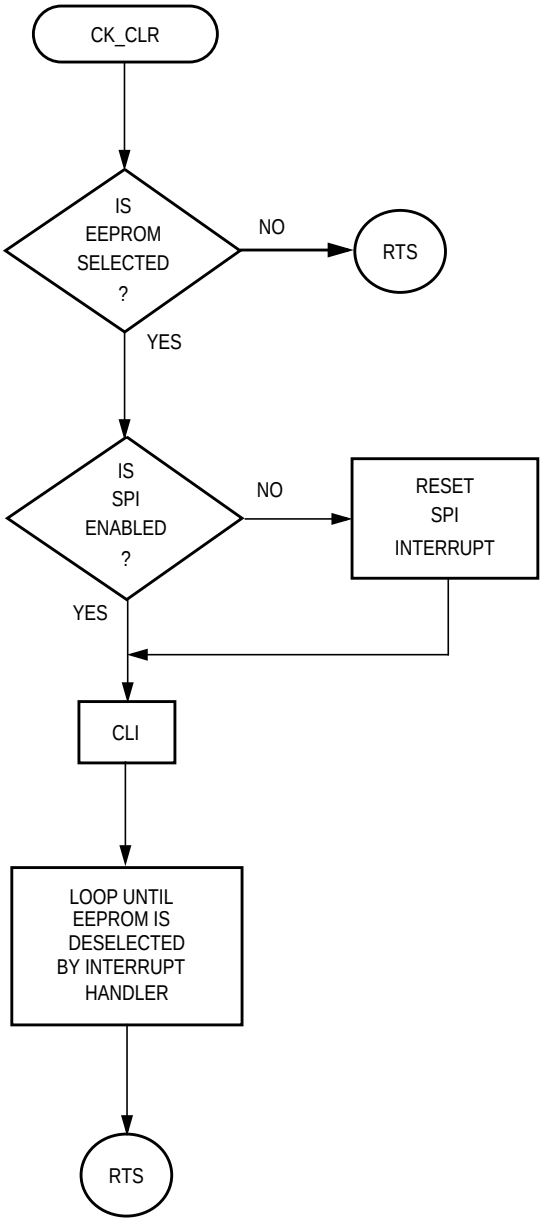


Appendix F — SPI Interrupt Application Flowcharts

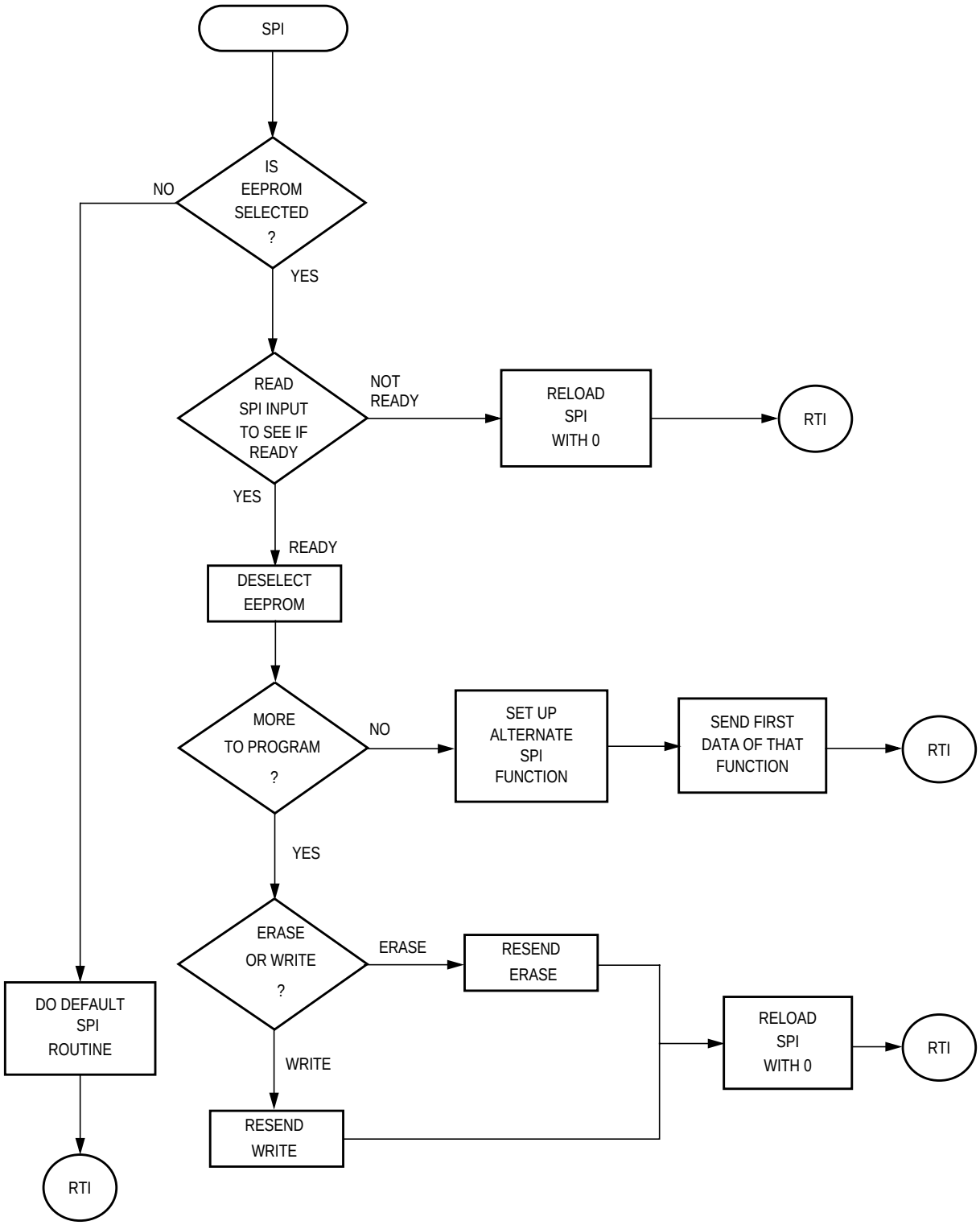





SENDADR #X
IS X GREATER THAN 6 ?
YES
RTS
SCALE X AS A POINTER IN CMDLEN
SELECT EEPROM
IS RUN ADDRESS TO BE SENT ?
SEND THE 2-BYTE COMMAND FROM COMMAND LIST CMOIST
RTS
NOTE: EEPROM IS STILL SELECTED
SEND THE FIRST COMMAND POINTED TO IN CMOIST
DECREMENT EE_START
USE EE_START WITH BYTE 2 OF COMMAND AND SEND
RTS
NOTE: EEPROM IS STILL SELECTED


CK_CLR
IS EEPROM SELECTED ?
NO
RTS
YES
IS SPI ENABLED ?
NO
RESET SPI INTERRUPT
YES
CLI
LOOP UNTIL EEPROM IS DESELECTED BY INTERRUPT HANDLER
RTS

Appendix G — SPI Interrupt Handler INTERRUPT Application Flowchart



Appendix H — I/O Polling to EEPROM Application Source

```
RAM         EQU   $50         ; Ram space starts at $50
ROM         EQU   $100        ; program space starts at $0100
VECTORS     EQU   $1ff4       ; Reset+ interrupt vectors start at $1ff4


****************************************
* Eeprom type and configuration switches
*
#SETNOT     9346FORM8         ; 9346 eeprom, 1 byte format.
#SETNOT     9346FORM16        ; 9346, 2 byte word format.
#SETNOT     9356FORM16        ; 9356, 2 byte word format.
#SETNOT     9366FORM16        ; 9366, 2 byte word format.

#SET        AUTOERASE         ; For eeproms that do not need to
                              ; erase before writing.
#SET        AUTOSEQ           ; For eeproms that automatically
                              ; sequence to the next cell when
                              ; being read.
*
****************************************
*
* RAM - variables
*
*
*

      ORG  RAM

ee_addr     ds  1             ; eeprom address stored here.
mem_addr    ds  1             ; Block index stored here.
block_to_go ds  1             ; Block length stored here.

data        ds    $ad         ; Rest of data space is data to be stored.

*
* PROGRAM
*
* The main subroutines are READ, EWEN, EWDS,
* WRITE, WRAL, ERASE, and ERAL.  SLACK, WAIT,
* CLOCK, and SHUFFLE support
* these.
*
* Port locations follow.
*
*

DIPORT      EQU   2           ; Eeprom "DI."
DILINE      EQU   6           ; portc.6, an output line.

DOPORT      EQU   2           ; Eeprom "DO."
DOLINE      EQU   7           ; portc.7, an input line.
```

```
CLKPORT     EQU   2           ; Eeprom CLocK.
CLKLINE     EQU   5           ; portc.5, an output line.

CSPORT      EQU   0           ; Eeprom Chip Select.
CSLINE      EQU   0           ; porta.0, an output line.


      ORG   ROM

*****************************************;
* WAIT - This routine delays the next command
*        to the eeprom until the most recent
*        write or erase has finished.
*        If in a write or erase
*        cycle the routine loops.  One
*        write or erase takes 4
*        milliseconds.
*
* INPUTS    - none
* OUTPUTS   - none
* DESTROYS  - nothing
*
WAIT:
            bset  CSLINE,CSPORT     ; Select.

            brclr DOLINE,DOPORT,$   ; Loop here until eeprom ready.
            bclr  CSLINE,CSPORT     ; De-select.
            rts

*****************************************;
* CLOCK# - clock data in to or out of the eeprom
*          using the # number of clocks.
*          "D_CARE" is used to handle the
*          'don't care' clocks required of
*          some commands.  It is conditionally
*          defined.  The required number of
*          'don't care' clocks is a function of
*          eeprom type and form.
* INPUTS    - none
* OUTPUTS   - none
* DESTROYS  - nothing
*

CLOCK6:                           ; Clocks six clocks to the eeprom.
#IF         9366FORM16
D_CARE:                           ; 9366 Form 16 uses 6 don't care bits.
#ENDIF

#IF         9356FORM16
D_CARE:                           ; 9356 Form 16 uses 6 don't care
                                  ; bits.
#ENDIF
            bset  CLKLINE,CLKPORT   ; Active clock.
            bclr  CLKLINE,CLKPORT   ; Inactive clock.
```

```
#IF         9346FORM8
D_CARE:                             ; 9346 Form 8 uses 5 don't care
                                ; bits.
#ENDIF

CLOCK5:                             ; Clocks five clocks to the eeprom.
            bset  CLKLINE,CLKPORT   ; Active clock.
            bclr  CLKLINE,CLKPORT   ; Inactive clock.

#IF         9346FORM16
D_CARE:                             ; 9346 Form 16 uses 4 don't care
                                    ; bits.
#ENDIF

CLOCK4:                             ; Clocks four clocks to the eeprom.
            bset  CLKLINE,CLKPORT   ; Active clock.
            bclr  CLKLINE,CLKPORT   ; Inactive clock.
CLOCK3:
                                    ; Clocks three clocks to the eeprom.
            bset  CLKLINE,CLKPORT   ; Active clock.
            bclr  CLKLINE,CLKPORT   ; Inactive clock.
CLOCK2:                             ; Clocks two clocks to the eeprom.
            bset  CLKLINE,CLKPORT   ; Active clock.
            bclr  CLKLINE,CLKPORT   ; Inactive clock.
CLOCK:                              ; Clocks one clock to the eeprom.
            bset  CLKLINE,CLKPORT   ; Active clock.
            bclr  CLKLINE,CLKPORT   ; Inactive clock.
            rts

*****************************************;
* EESEND    - sends the complement of the carry
*             to the eeprom and rotates the
*             accumulator left through the carry.
*
* INPUTS    - accumulator
* OUTPUTS   - accumulator left rotated through
*             carry, and one bit to the Eeprom.
* DESTROYS  - nothing
*
EESEND:
            bcc   OPU1              ; If carry clear jump to set.
            bclr  DILINE,DIPORT     ; If carry set clear the output to
                        ; eeprom.
            bra   OPU0
OPU1:
            bset  DILINE,DIPORT     ; Clear carry means set output to           ;ee-
prom.
OPU0:                               ; Clock the complement of the carry
                                    ; eeprom.
            bset  CLKLINE,CLKPORT   ; Active clock.
            bclr  CLKLINE,CLKPORT   ; Inactive clock.
            rola                    ; ready the next bit to be sent by
                                    ; rotating to carry.
            rts
```

```
*****************************************;
* SENDADR   - Send a 6,7, or 8 bit address to
*             the serial eeprom depending on its
*             type and form.
*  -or-
* SENDDAT   - Send 8 bits of data.
*
* INPUTS    - Byte address in accumulator.
*             In 16 bit format bit 0 is ignored.
* OUTPUTS   - none
* DESTROYS  - Accumulator
*
SENDDAT:
            rola                    ; ready the first data bit to be
                                ; sent by rotating to carry.
            jsr   EESEND            ; Send data bit 7 or 15.
            jsr   EESEND            ; Send data bit 6 or 14.
            bra   RTTZ
SENDADR:
            coma                    ; Addresses are inverted twice
                                    ; before being sent!
#IF         9346FORM8
            rola                    ; Rotate address extra bit through
                                    ; carry.
            rola                    ; ready the first address bit to be
                                    ; sent by rotating to carry.
            jsr   EESEND            ; Send address bit 6.
#ENDIF

#IF         9346FORM16
            rola                    ; Rotate extra address bit through
                                    ; carry.
            rola                    ; Rotate extra address bit through
                                    ; carry.
            rola                    ; ready the first address bit to be
                                    ; sent by rotating to carry.
#ENDIF

#IF         9356FORM16
            ora   #$80              ; Set the Don't care bit.
            rola                    ; Rotate the Don't care bit to the
                                ; carry.
            jsr   EESEND            ; Send 1 Don't care bit.
            jsr   EESEND            ; Send address bit 6.
#ENDIF

#IF         9366FORM16
            rola                    ; ready the first address bit to be
                                ; sent by rotating to carry.
            jsr   EESEND            ; Send address bit 7.
            jsr   EESEND            ; Send address bit 6.
```

```
#ENDIF

RTTZ:
            jsr   EESEND            ; Send bit 5 or 13.
            jsr   EESEND            ; Send bit 4 or 12.
            jsr   EESEND            ; Send bit 3 or 11.
            jsr   EESEND            ; Send bit 2 or 10.
            jsr   EESEND            ; Send bit 1 or 9.
            jsr   EESEND            ; Send bit 0 or 8.
            rts

*****************************************;
* SUEEP - Set up the eeprom ports.  Called
*         frequently to ensure the ports are
*         set up for the eeprom and so that
*         other tasks can share the ports.
*
* INPUTS   - none
* OUTPUTS  - DDRA,DDRB
* DESTROYS - nothing
*
SUEEP:
            bset  CSLINE,CSPORT+4    ; Chip Select port is
            bclr  CSLINE,CSPORT      ; output and low.
            bset  CLKLINE,CLKPORT+4 ; Clock is output.
            bclr  DOLINE,DOPORT+4   ; DO is an input.
            bset  DILINE,DIPORT+4   ; DI is an output.
            rts

*****************************************;
* EWEN - This subroutine enables erase and write
*        operations.  It in effect unlocks the
*        eeprom so that its cells may be
*        changed.
*
* INPUTS    - none
* OUTPUTS   - none
* DESTROYS  - nothing
*
EWEN:
            bset  CSLINE,CSPORT     ; Select the Eeprom
            bset  DILINE,DIPORT     ; Send 1.
            jsr   CLOCK             ; Clock it into the eeprom.
            bclr  DILINE,DIPORT     ; Send 00.
            jsr   CLOCK2            ; Clock them into the eeprom.
            bset  DILINE,DIPORT     ; Send 11.
            jsr   CLOCK2            ; Clock them into the eeprom.
            bclr  DILINE,DIPORT     ; DI line low.
            jsr   D_CARE            ; Clock the Don't care clocks.
            bclr  CSLINE,CSPORT     ; deselect the Eeprom
            rts
```

```
*****************************************;
* EWDS - This subroutine disables erase and
*        write operations so that data cannot be
*        inadvertently corrupted.  It in effect
*        locks the eeprom so that its cells
*        cannot be changed.
*

* INPUTS    - none
* OUTPUTS   - none
* DESTROYS  - nothing
*
EWDS:
            bset  CSLINE,CSPORT     ; Select the Eeprom
            bset  DILINE,DIPORT     ; Send 1.
            jsr   CLOCK             ; Clock it into the eeprom.
            bclr  DILINE,DIPORT     ; Send 0000.
            jsr   CLOCK4            ; Clock them into the eeprom.
            bclr  DILINE,DIPORT     ; DI line low.
            jsr   D_CARE            ; Clock the Don't care clocks.
            bclr  CSLINE,CSPORT     ; deselect the Eeprom
            rts

*****************************************;
* ERAL - This subroutine erases the entire
*        eeprom.  An erased cell will put a high
*        level on the DO line when read, but
*        due to inverting in READ, the result
*        will arrive as 0x00 in 6805 memory.
*        ERAL calls EWEN to allow erasure.
*
* INPUTS    - none
* OUTPUTS   - none
* DESTROYS  - all contents of eeprom
*
ERAL:
            jsr   SUEEP             ; Set up the ports for the eeprom.
            jsr   EWEN              ; Open the eeprom for writing.
            bset  CSLINE,CSPORT     ; Select the Eeprom
            bset  DILINE,DIPORT     ; Send 1.
            jsr   CLOCK             ; Clock it into the eeprom.
            bclr  DILINE,DIPORT     ; Send 00.
            jsr   CLOCK2            ; Clock them into the eeprom.
            bset  DILINE,DIPORT     ; Send 1.
            jsr   CLOCK             ; Clock it into the eeprom.
            bclr  DILINE,DIPORT     ; Send 0.
            jsr   CLOCK             ; Clock it into the eeprom.
            jsr   D_CARE            ; Clock the Don't care clocks.
            bclr  CSLINE,CSPORT     ; deselect the Eeprom
            jsr   WAIT              ; Pause until the eeprom comes
                                    ; ready.
            rts
```

```
*****************************************;
* WRAL - In FORM8 eeproms this subroutine
*        writes the byte in the accumulator to
*        every byte of the Eeprom.  In FORM16
*        eeproms the accumulator is written to
*        the most significant byte the X
*        register is written to the less
*        significant byte.
*
* INPUTS    - Accumulator ( and X for FORM16)
* OUTPUTS   - none
* DESTROYS  - accumulator in FORM16 applications.
*
#MACRO      WRAL16
            txa                     ; Put more significant byte into X.
            jsr   SENDDAT           ; Send that byte to the eeprom for
                                    ; writing.
#MACROEND

WRAL:
            jsr   SUEEP             ; Set up the ports for the eeprom.

#IFNOT AUTOERASE
            jsr   ERAL              ; Erase and open the eeprom.
#ELSEIF
            jsr   EWEN              ; Open the eeprom for writing.
#ENDIF
            bset  CSLINE,CSPORT     ; Select the Eeprom
            bset  DILINE,DIPORT     ; Send 1.
            jsr   CLOCK             ; Clock it into the eeprom.
            bclr  DILINE,DIPORT     ; Send 000.
            jsr   CLOCK3            ; Clock them into the eeprom.
            bset  DILINE,DIPORT     ; Send 1.
            jsr   CLOCK             ; Clock it into the eeprom.
            bclr  DILINE,DIPORT     ; Send DI low.
            jsr   D_CARE            ; Clock the Don't care clocks.

            jsr   SENDDAT           ; Send a byte for writing.

#IF         9346FORM16
            WRAL16                  ; Send a second byte if form 16.
#ENDIF

#IF         9356FORM16
            WRAL16                  ; Send a second byte if form 16.
#ENDIF

#IF         9366FORM16
            WRAL16                  ; Send a second byte if form 16.
```

```
#ENDIF
            bclr  DILINE,DIPORT     ; Send DI line low.
            bclr  CSLINE,CSPORT     ; deselect the Eeprom

            jsr   WAIT              ; <- Waits here for erasure to
                                    ; finish.

            jsr   EWDS              ; Close the eeprom for writing.
            rts

*****************************************;
* ERASE - This subroutine Erases an eight
*         cell byte or 16 cell word in the
*         Eeprom.  The address of the cell is
*         located in the accumulator.  The
*         accumulator is returned unchanged.
*
* INPUTS    - Eeprom address for erasure in Acc.
* OUTPUTS   - none
* DESTROYS  - X.
*
ERASE:
            tax                     ; Store address in X.
            jsr   SUEEP             ; Set up the ports for the eeprom.
            jsr   EWEN              ; Open the eeprom for writing.
            bset  CSLINE,CSPORT     ; Select the Eeprom
            bset  DILINE,DIPORT     ; Send 111.
            jsr   CLOCK3            ; Clock them into the eeprom.

            jsr   SENDADR           ; Send the 6,7, or 8 bit address.

            bclr  DILINE,DIPORT     ; DI line low.
            bclr  CSLINE,CSPORT     ; deselect the Eeprom

            jsr   WAIT              ; <- Waits here for erasure to
                                    ; finish.

            txa                     ; Return the address to accumulator.
            rts

*****************************************
* Write macros ------------------------------

#MACRO WRBYTE
            ldx   mem_addr          ; Bring in the address pointer
                                    ; of the byte to be written.
            lda   ,x                ; Bring the byte to be written
                                    ; into the accumulator.
            incx                    ; Increment the address pointer.
            stx   mem_addr          ; Update the address pointer.
            jsr   SENDDAT           ; Send accumulator to eeprom for
                                    ; writing.
            lda   block_to_go       ; load block length.
            deca                    ; Decrement length and check if
                                    ; done.
```

```
            sta   block_to_go       ; Update block length.
#MACROEND

#MACRO INIT16
        lda       block_to_go       ; Check for Zero length.
        beq       WRDONE            ; Abort if Zero.
        rora                        ; Place LS bit of address in carry.
        bcc       LEN_OK            ; Ensure that block_to_go
        rola                        ; starts as an even number.
        inca                        ; increment if not.
        sta       block_to_go       ; Update to new even value.
LEN_OK:
#MACROEND

*****************************************;
* WRITE - This subroutine Writes a block of
*         eight cell bytes to the Eeprom.
*         Writing starts at low memory value
*         in both eeprom, ee_addr, and 6805
*         memory, mem_addr, and increments
*         upward as block_to_go is decremented
*         downward.
*
* INPUTS    - The following memory locations
*             set up as follows.
*         ee_addr   -> contains the absolute
*                      address of where the
*                      data will start in
*                      the eeprom.
*         mem_addr  -> contains the absolute
*                      starting address of the
*                      block of memory
*                      to be written to eeprom.
*         block_to_go  -> The length of the block,
*                      1 writes one byte,
*                      0 writes none.
* OUTPUTS   - none
* DESTROYS  - ee_addr, mem_addr, block_to_go,
*             Acc. and X
*
WRITE:
            jsr   SUEEP             ; Set up the ports for the eeprom.

#IF AUTOERASE
            jsr   EWEN              ; Open the eeprom for writing.
#ENDIF

#IF         9346FORM16
            INIT16                  ; Even the block to be written.
#ENDIF

#IF         9356FORM16
            INIT16                  ; Even the block to be written.
#ENDIF
```

```
#IF         9366FORM16
            INIT16                  ; Even the block to be written.
#ENDIF

WRLP:
            lda   ee_addr           ; eeprom address to be written,
            inca                    ; Update for the
            sta   ee_addr           ; next address to be written.
            deca                    ; Restore address to be written
                                    ; for this time.

#IFNOT AUTOERASE
            jsr   ERASE             ; Erase the cell if not autoerase.
#ENDIF

            bset  CSLINE,CSPORT     ; Select the Eeprom
            bset  DILINE,DIPORT     ; Send 1.
            jsr   CLOCK             ; Clock it into the eeprom.
            bclr  DILINE,DIPORT     ; Send 0.
            jsr   CLOCK             ; Clock it into the eeprom.
            bset  DILINE,DIPORT     ; Send 1.
            jsr   CLOCK             ; Clock it into the eeprom.

            jsr   SENDADR           ; Send eeprom address to eeprom.

            WRBYTE                  ; Send a byte to be written to the
                                    ; eeprom.
#IF         9346FORM16
            WRBYTE                  ; Send a byte to be written to the eeprom.
#ENDIF

#IF         9356FORM16
            WRBYTE                  ; Send a byte to be written to the
                                    ; eeprom.
#ENDIF

#IF         9366FORM16
            WRBYTE                  ; Send a byte to be written to the
                                    ; eeprom.
#ENDIF
            bclr  DILINE,DIPORT     ; DI low.
            bclr  CSLINE,CSPORT     ; deselect the Eeprom

            jsr   WAIT              ; <- Waits here until the byte
                                    ; is written.
            tsta                    ; Acc still has block_to_go.
            bne   WRLP              ; If not done loop again.
WRDONE:
            jsr   EWDS              ; Close the eeprom for writing.
            rts
```

```
*******************************************
*  reading - The following are used by the
*            reading routine.
RECEIVE:
            ldx   #$8
RCVLP:
            bset  CLKLINE,CLKPORT   ; Active clock.
            bclr  CLKLINE,CLKPORT   ; Inactive clock.
            brset DOLINE,DOPORT,RTTY; Bit from eeprom
                                    ; comes in carry.
RTTY:                               ; Not really a branch.
            rola                    ; Rotate new bit from carry to
                                    ; accumulator.
            decx                    ; decrease bit count.
            bne   RCVLP             ; If bit count = 0 then acc has
                                    ; received byte.
            coma                    ; Complement the whole thing, All
                                    ; bits come out of the eeprom
                                ; complemented.
            rts

#MACRO    READ8
            jsr   RECEIVE           ; read 1 byte from eeprom.
            ldx   block_to_go       ; Check if finished.
            beq   NOSAVE            ; Throw byte away if done.
            decx                    ; If kept decrement the length
                                    ; counter.
            stx   block_to_go       ; Update the block length counter.
            ldx   mem_addr          ; Load the address to store into X.
            sta   ,x                ; Store read byte to memory.
            incx                    ; Increment the address pointer.
            stx   mem_addr          ; Update the address pointer.
NOSAVE:                             ; Branch around storage.
#MACROEND

*******************************************;
* READ - This subroutine reads a block of
*        data out of the eeprom and places it
*        in a block of 6805 memory.  It is used
*        with eeproms that do not have the
*        autosequence feature.
*
* INPUTS    - The following memory locations
*             set up as follows.
*         ee_addr   -> contains the eeprom
*                      address where the data
*                      block starts.
*
*         mem_addr  -> contains the absolute
*                      starting
*                      address of the 6805
*                      memory block
*                      destination.
*
```

```
*         block_to_go  -> The length of the block,
*                      1 reads one byte,
*                      0 reads none.
* OUTPUTS   - a block of updated memory
* DESTROYS  - ee_addr, mem_addr, block_to_go,
*             Acc. and X
*
*
READ:
            jsr   SUEEP             ; Set up the ports for the eeprom.

            jsr   EWDS              ; Close the eeprom for writing.
RDNLP:
            ldx   block_to_go       ; Read length of block.
            beq   RDNDONE           ; If done exit.
            bset  CSLINE,CSPORT     ; Select the Eeprom
            bset  DILINE,DIPORT     ; Send 11.
            jsr   CLOCK2            ; Clock them into the eeprom.
            bclr  DILINE,DIPORT     ; Send 0.
            jsr   CLOCK             ; Clock it into the eeprom.

            lda   ee_addr           ; Bring in eeprom address.
            inca                    ; Update eeprom address
            sta   ee_addr           ; for next reading.
            deca                    ; Restore eeprom address for this
                                    ; reading.
            jsr   SENDADR           ; send eeprom address

            bclr  DILINE,DIPORT     ; Bring low for the extra clock
                                    ; of read cycle.
            READ8                   ; Read a byte out of the eeprom and
                                    ; into the accumulator.
#IF         9346FORM16
            READ8                   ; Read byte 2 out of the eeprom and
                                    ; into the accumulator.
#ENDIF

#IF         9356FORM16
            READ8                   ; Read byte 2 out of the eeprom and
                                    ; into the accumulator.
#ENDIF

#IF         9366FORM16
            READ8                   ; Read byte 2 out of the eeprom and
                                    ; into the accumulator.
#ENDIF

            bclr  CSLINE,CSPORT     ; deselect the Eeprom
            bra   RDNLP             ; Go back to see if done reading.
RDNDONE:                            ; All done reading.
            rts

#IF AUTOSEQ
```

```
*****************************************;
* RDAUTO - This subroutine reads a block of
*        data out of the eeprom and places it in
*        a block of 6805 memory.  It functions
*        faster than the READ routine above and
*        can only be used with the newer eeproms
*        that automatically cycle to the next
*        register, the "autosequence" or
*        "autoincrement" feature.
*
* INPUTS    - The following memory locations
*             set up as follows.
*         ee_addr   -> contains the eeprom
*                    address where the data
*                    block starts.
*
*         mem_addr  -> contains the absolute
*                    starting address of the 6805
*                    memory block destination.
*
*         block_to_go  -> The length of the block,
*                    1 writes one byte, 0 writes
*                    none.
* OUTPUTS   - a block of updated memory
* DESTROYS  - ee_addr, mem_addr, block_to_go,
*             Acc. and X.
*
* USES - "RECEIVE" which is defined above.
*
RDAUTO:
            jsr   SUEEP             ; Set up the ports for the eeprom.

            jsr   EWDS              ; Close the eeprom for writing.
            bset  CSLINE,CSPORT     ; Select the Eeprom
            bset  DILINE,DIPORT     ; Send 11.
            jsr   CLOCK2            ; Clock them into the eeprom.
            bclr  DILINE,DIPORT     ; Send 0.
            jsr   CLOCK             ; Clock it into the eeprom.

            lda   ee_addr           ; Bring in eeprom address.
            jsr   SENDADR           ; Send it out.

            bclr  DILINE,DIPORT     ; Bring low for the extra clock
                                    ; of read cycle.
RDALP:
            ldx   block_to_go       ; Bring in length left to send.
            tstx                    ; Check for done.
            beq   RDADONE           ; If done exit.
            decx                    ; Decrement length left.
            stx   block_to_go       ; Update length left.
            jsr   RECEIVE           ; Receive a byte from the eeprom.
            ldx   mem_addr          ; Load in the place in memory to
                                    ; put the byte.
```

```
            sta  ,x                 ; <- Change store command here if
                                    ; memory is above $100.
            incx                    ; increment the memory pointer.
            stx   mem_addr          ; Update the memory pointer.
            bra   RDALP             ; Loop back until done.
RDADONE:
            bclr  CSLINE,CSPORT     ; deselect the Eeprom
            rts
#ENDIF

*****************************************;
* START - Sample calling of routines.

BSTART      EQU   0                 ; Start eeprom addresses for these
                                    ; examples.
BL_LEN      EQU   $80               ; Length of block for these
                                    ; examples.
STARTRD:
            lda   #BSTART           ; Start reading eeprom at
                                    ; address BSTART.
            sta   ee_addr           ; Place first eeprom address in                ;
                                    ; ee_addr.
            lda   #data             ; Load in start address of receiving
                                    ; memory.
            sta   mem_addr          ; Place start address in mem_addr.
            lda   #BL_LEN           ; Length of block to read.
            sta   block_to_go       ; Store block length.

#IF AUTOSEQ
            jsr   RDAUTO            ; Read the eeprom using
                                    ; autosequencing.
#ELSEIF
            jsr   READ              ; Read the W/O Autosequencing
                                    ; eeprom.
#ENDIF
            bra   $                 ; jump to this location
                                    ; (do nothing else).

STARTWR:
            lda   #BSTART           ; Start writing eeprom at
                                    ; address BSTART.
            sta   ee_addr           ; Place first eeprom address in
                                    ; ee_addr.
            lda   #data             ; Load in start address of block in
                                    ; memory.
            sta   mem_addr          ; Place start address in mem_addr.
            lda   #BL_LEN           ; Length of block to write.
            sta   block_to_go       ; Store block length.
            jsr   WRITE             ; WRITE the block.
            bra   $                 ; (do nothing else).

STARTERAL:
            jsr   ERAL              ; Erases the entire serial eeprom
LOOP3:
            bra   $                 ; (do nothing else).
```

```
STARTWRL:
            lda   #$a5              ; (write $a5 to form 8 eeprom.)

#IF         9346FORM16
            ldx   #$c3              ; write $a5c3 to form 16 eeprom.
#ENDIF

#IF         9356FORM16
            ldx   #$c3              ; write $a5c3 to form 16 eeprom.
#ENDIF

#IF         9366FORM16
            ldx   #$c3              ; write $a5c3 to form 16 eeprom.
#ENDIF
            jsr   WRAL              ; 0xa5 to all memory locations in
                                    ; the eeprom.
*     bra   $                       ; (do nothing else).

STARTERSE:
            lda   #$05              ; Bring in 5 as location to be
                                    ; erased.
            jsr   ERASE             ; Erases memory location 5 of the
                                    ; eeprom.
            bra   $                 ; (do nothing else).

      ORG VECTORS

VECSPI:     fdb   STARTRD           ; SPI VECTOR
VECSCI:     fdb   STARTRD           ; SCI VECTOR
VECTMR:     fdb   STARTRD           ; TIMER VECTOR
VECIRQ:     fdb   STARTRD           ; IRQ VECTOR
VECSWI:     fdb   STARTRD           ; SWI VECTOR
VECRST:     fdb   STARTRD           ; START VECTOR
```

Appendix I — SPI Polling to EEPROM Application Source

```
RAM         EQU   $50               ; RAM starts at $50
ROM         EQU   $100              ; ROM Starts at $100
VECTORS     EQU   $1ff4             ; Reset and interrupt vectors start
                                    ; at $1ff4

****************************************
* Eeprom type and configuration switches
*
#SETNOT     9346FORM8               ; 9346 eeprom, 1 byte format.
#SETNOT     9346FORM16              ; 9346, 2 byte word format.
#SETNOT     9356FORM16              ; 9356, 2 byte word format.
#SET        9366FORM16              ; 9366, 2 byte word format.

#SET        AUTOERASE               ; For eeproms that do not need to
                                    ; erase before writing.
#SET        AUTOSEQ                 ; For eeproms that automatically
                                    ; sequence to the next cell when
                                    ; being read.

****************************************
* RAM - variables
*

      ORG  RAM

ee_addr     ds    1                 ; eeprom address stored here.
mem_addr    ds    1                 ; Block index stored here.
block_to_go ds    1                 ; Block length stored here.

data        ds    $ad               ; Rest of data space is data to be stored.

*
* PROGRAM
*
* The main subroutines are READ, EWEN, EWDS,
* WRITE, WRAL, ERASE, and ERAL.  SLACK, WAIT,
* CLOCK, and SHUFFLE support
* these.
*
* Port locations follow.
*
*
CSPORT      EQU   0                 ; Eeprom Chip Select.
CSLINE      EQU   5                 ; porta.5, an output line.

SPCR        EQU   $0a               ; Location of SPI control reg.
SPSR        EQU   $0b               ; Location of SPI status reg.
SPIDAT      EQU   $0c               ; Location of SPI data reg.


      ORG  ROM
```

```
****************************************
* Command set

#if         9346FORM8               ; Command set for 9346 in the byte wide
                                    ; form.
MASK        equ   %01111111         ; Mask of valid address bits
READ1       equ   %00000110         ; READ command padded to 16 bits.
READ2       equ   %00000000
EWEN1       equ   %00000010         ; Write enable command padded to 16 bits.
EWEN2       equ   %01100000
EWDS1       equ   %00000010         ; Write protect command padded to 16 bits.
EWDS2       equ   %00000000
WRITE1      equ   %00000010         ; Write command padded to 16 bits.
WRITE2      equ   %10000000
WRAL1       equ   %00000010         ; Write all command padded to 16 bits.
WRAL2       equ   %00100000
ERASE1      equ   %00000011         ; Erase cell command padded to 16 bits.
ERASE2      equ   %10000000
ERAL1       equ   %00000010         ; Erase all command padded to 16 bits.
ERAL2       equ   %01000000
#endif

#if         9346FORM16              ; Command set for 9346 in the 16 bit wide
                                    ; form.
MASK        equ   %00111111         ; Mask of valid address bits
READ1       equ   %00000011         ; READ command padded to 16 bits.
READ2       equ   %00000000
EWEN1       equ   %00000001         ; Write enable command padded to 16 bits.
EWEN2       equ   %00110000
EWDS1       equ   %00000001         ; Write protect command padded to 16 bits.
EWDS2       equ   %00000000
WRITE1      equ   %00000001         ; Write command padded to 16 bits.
WRITE2      equ   %01000000
WRAL1       equ   %00000001         ; Write all command padded to 16 bits.
WRAL2       equ   %00010000
ERASE1      equ   %00000001         ; Erase cell command padded to 16 bits.
ERASE2      equ   %11000000
ERAL1       equ   %00000001         ; Erase all command padded to 16 bits.
ERAL2       equ   %00100000
#endif

#if         9356FORM16              ; Command set for 9356 in the 16 bit wide
                                    ; form.
MASK        equ   %01111111         ; Mask of valid address bits
READ1       equ   %00001100         ; READ command padded to 16 bits.
READ2       equ   %00000000
EWEN1       equ   %00000100         ; Write enable command padded to 16 bits.
EWEN2       equ   %11000000
EWDS1       equ   %00000100         ; Write protect command padded to 16 bits.
EWDS2       equ   %00000000
WRITE1      equ   %00000101         ; Write command padded to 16 bits.
WRITE2      equ   %00000000
WRAL1       equ   %00000100         ; Write all command padded to 16 bits.
WRAL2       equ   %01000000
ERASE1      equ   %00000111         ; Erase cell command padded to 16 bits.
```

```
ERASE2      equ   %00000000
ERAL1       equ   %00000100         ; Erase all command padded to 16 bits.
ERAL2       equ   %10000000
#endif

#if         9366FORM16              ; Command set for 9366 in the 16 bit wide
                                  ; form.
MASK        equ   %11111111         ; Mask of valid address bits
READ1       equ   %00001100         ; READ command padded to 16 bits.
READ2       equ   %00000000
EWEN1       equ   %00000100         ; Write enable command padded to 16 bits.
EWEN2       equ   %11000000
EWDS1       equ   %00000100         ; Write protect command padded to 16 bits.
EWDS2       equ   %00000000
WRITE1      equ   %00000101         ; Write command padded to 16 bits.
WRITE2      equ   %00000000
WRAL1       equ   %00000100         ; Write all command padded to 16 bits.
WRAL2       equ   %01000000
ERASE1      equ   %00000111         ; Erase cell command padded to 16 bits.
ERASE2      equ   %00000000
ERAL1       equ   %00000100         ; Erase all command padded to 16 bits.
ERAL2       equ   %10000000
#endif


*****************************************;
* WAIT - This routine delays the next command
*        to the eeprom until the most recent
*        write or erase has finished.
*        If in a write or erase
*        cycle the routine loops.  One
*        write or erase takes 4
*        milliseconds.
*
* INPUTS    - none
* OUTPUTS   - none
* DESTROYS  - nothing
*
WAIT:
            bset  CSLINE,CSPORT     ; Select.

            clr   SPIDAT            ; Send 8 more don't care zeros.
            brclr 7,SPSR,$          ; Loop here until eeprom ready.
            tst   SPIDAT            ; Is eeprom ready?  Zero if not.
            beq   WAIT              ; if not ready send more clocks and
                                    ; zeros.
            bclr  CSLINE,CSPORT     ; De-select eeprom is ready.
            rts
```

```
*****************************************;
* IDIO - This routine handles the idiosyncratic
*        requirements of the particular test
*        hardware used.  It may be deleted
*        in most applications
IDIO:
            bset  4,CSPORT+4        ; Output port 0.4
            bset  4,CSPORT          ; Pulls up the "SS" line.
            bset  7,CSPORT+4        ; Output port 0.7
            bset  7,CSPORT          ; Pulls up the "RESET" line.
            rts

*****************************************;
* SUSPI - Sets up the eeprom IO port and the
*         SPI to communicate with the eeprom
*         by polling.
*         Other tasks can share the SPI.
*
* INPUTS   - none
* OUTPUTS  - DDRA,SPI
* DESTROYS - Accumulator.
*
SUSPI:
            bset  CSLINE,CSPORT+4   ; Chip select line is output.
            bclr  CSLINE,CSPORT     ; Chip select is low de-selected.
            lda   #%01010000        ; SPI enabled phase 0.
            sta   SPCR              ; SPI control register, SPI set up.
            rts

*****************************************;
* EESEND - sends a byte to the eeprom through
*           the SPI.
*
* INPUTS    - accumulator, send to SPI
* OUTPUTS   - accumulator, response from SPI
*
* DESTROYS  - Accumulator
*
EESEND:
            sta   SPIDAT            ; Accumulator goes out the SPI.
            brclr 7,SPSR,$          ; Should loop 3 times.
            lda   SPIDAT            ; What comes out of the SPI is
                                    ; placed in the accumulator.
            rts
```

```
*****************************************;
* EWEN - This subroutine enables erase and write
*        operations.  It in effect unlocks the
*        eeprom so that its cells may be
*        changed.
*
* INPUTS    - none
* OUTPUTS   - none
* DESTROYS  - nothing
*
EWEN:
            jsr   SUSPI             ; Ensure that the SPI is set up.
            lda   #EWEN1            ; Load first part of EWEN command.
            bset  CSLINE,CSPORT     ; Select the Eeprom
            jsr   EESEND            ; Send the command out the SPI.
            lda   #EWEN2            ; Load the second part of EWEN
                                    ; command.
            jsr   EESEND            ; Send the command out the SPI.
            bclr  CSLINE,CSPORT     ; deselect the Eeprom
            rts



*****************************************;
* EWDS - This subroutine disables erase and
*        write operations so that data cannot be
*        inadvertently corrupted.  It in effect
*        locks the eeprom so that its cells
*        cannot be changed.
*
* INPUTS    - none
* OUTPUTS   - none
* DESTROYS  - nothing
*
EWDS:
            jsr   SUSPI             ; Ensure that the SPI is set up.
            lda   #EWDS1            ; Load first part of the EWDS
                                    ; command.
            bset  CSLINE,CSPORT     ; Select the Eeprom
            jsr   EESEND            ; Send EWDS1 out the SPI.
            lda   #EWDS2            ; Load second part of the EWDS
                                    ; command.
            jsr   EESEND            ; Send EWDS2 out the SPI.
            bclr  CSLINE,CSPORT     ; deselect the Eeprom
            rts
```

```
*****************************************;
* ERAL - This subroutine erases the entire
*        eeprom.  An erased 93x6 cell will
*        put a high level on the DO line when
*        read, but due to inverting in READ,
*        the result will arrive as 0x00 in
*        6805 memory.  ERAL calls EWEN to
*        allow erasure.
*
* INPUTS    - none
* OUTPUTS   - none
* DESTROYS  - all contents of eeprom
*
ERAL:
            jsr   SUSPI             ; Ensure that the SPI is set up.
            jsr   EWEN              ; "OPEN" the eeprom for writing and
                                    ; erasure.
            lda   #ERAL1            ; Load the first part of the ERAL
                                    ; command.
            bset  CSLINE,CSPORT     ; Select the Eeprom
            jsr   EESEND            ; Send ERAL1 out the SPI.
            lda   #ERAL2            ; Load the second part of the ERAL
                                    ; command.
            jsr   EESEND            ; Send ERAL2 out the SPI.
            bclr  CSLINE,CSPORT     ; deselect the Eeprom
            jsr   WAIT              ; Wait until eeprom is ready.
            rts

*****************************************;
* WRAL - In FORM8 eeproms this subroutine
*        writes the byte in the accumulator to
*        every byte of the Eeprom.  In FORM16
*        eeproms the accumulator is written to
*        the most significant byte the X
*        register is written to the less
*        significant byte.
*
* INPUTS    - Accumulator ( and X for FORM16)
* OUTPUTS   - none
* DESTROYS  - accumulator, ee_addr, and mem_addr .
*

#MACRO WRAL16
            lda   mem_addr          ; Load Second byte of word to be
                                    ; written.
            jsr   EESEND            ; Send that second byte out the SPI
                                    ; to eeprom.
#MACROEND

WRAL:
            sta   ee_addr           ; Store low order byte in ee_addr.
            stx   mem_addr          ; Store high order byte in mem_addr.
            jsr   SUSPI             ; Ensure that the SPI is set up.
```

```
#IFNOT AUTOERASE
            jsr   ERAL              ; if not autoerase, erase the eeprom
                                    ; first.
#ELSEIF
            jsr   EWEN              ; if an autoerase eeprom open it for
                                    ; writing.
#ENDIF
            lda   #WRAL1            ; Load Write all #1 command.
            bset  CSLINE,CSPORT     ; Select the Eeprom
            jsr   EESEND            ; Send Write All command #1 out the
                                    ; SPI.
            lda   #WRAL2            ; Load Write All #2 command.
            jsr   EESEND            ; Send Write all command #2 out the
                                    ; SPI.
            lda   ee_addr           ; Load the low byte to be written.
            jsr   EESEND            ; Send it out the SPI.

#IF         9346FORM16
            WRAL16                  ; Send out the high byte if a form
                                    ; 16 eeprom.
#ENDIF

#IF         9356FORM16
            WRAL16                  ; Send out the high byte if a form
                                    ; 16 eeprom.
#ENDIF

#IF         9366FORM16
            WRAL16                  ; Send out the high byte if a form
                                    ; 16 eeprom.
#ENDIF
            bclr  CSLINE,CSPORT     ; deselect the Eeprom

            jsr   WAIT              ; <- Waits here for erasure to
                                    ; finish.

            jsr   EWDS              ; "Close" or write protect the
                                    ; eeprom.
            rts
```

```
*****************************************;
* ERASE - This subroutine Erases an eight
*         cell byte or 16 cell word in the
*         Eeprom.  The address of the cell is
*         located in the accumulator.  The
*         accumulator is returned unchanged.
*
* INPUTS    - Eeprom address for erasure in Acc.
* OUTPUTS   - none
* DESTROYS  - X
*

ERASE:
            jsr   SUSPI             ; Ensure that the SPI is set up.
            jsr   EWEN              ; Open the eeprom for writing.
            lda   #ERASE1           ; Load with Erase #1 command.
            bset  CSLINE,CSPORT     ; Select the Eeprom
            jsr   EESEND            ; Send Erase #1 command out SPI.
            txa                     ; Copy address to X for storage.
            and   #MASK             ; AND address with mask.
            ora   #ERASE2           ; OR address with ERASE #2 command.
            jsr   EESEND            ; Send out SPI.

            bclr  CSLINE,CSPORT     ; deselect the Eeprom

            jsr   WAIT              ; <- Waits here for erasure to
                                    ; finish.
            txa                     ; Return eeprom address to                    ;
                                    ; accumulator.
            rts

*****************************************
* Write macros -----------------------------

#MACROWRBYTE
            ldx   mem_addr          ; Load pointer reg with address
                                    ; of byte to be sent next.
            lda   ,x                ; Bring that byte into accumulator.
            incx                    ; Increment pointer for next byte.
            stx   mem_addr          ;  Update with address of next byte
                                    ; to be sent.
            jsr   EESEND            ; Send byte out SPI.
            lda   block_to_go       ; Load the length left to be sent.
            deca                    ; Dec length and check if done.
            sta   block_to_go       ; Update  the length  of  block  to  be
                                    ; sent.
#MACROEND

#MACRO INIT16
            lda   block_to_go       ; Load the length left to be sent.
            beq   WRDONE            ; If Zero finish.
            rora                    ; Place least significant bit in ; carry.
            bcc   LEN_OK            ; Ensure that block_to_go
            rola                    ; starts as an even number.
            inca                    ; If not increment to an even ; number.
            sta   block_to_go       ; Update to the new even number.
LEN_OK:
#MACROEND
```

```
*****************************************;
* WRITE - This subroutine Writes a block of
*         eight cell bytes to the Eeprom.
*
* INPUTS    - The following memory locations
*             set up as follows.
*         ee_addr  -> contains the absolute
*                     address of where the
*                     data will start in
*                     the eeprom.
*         mem_addr  -> contains the absolute
*                     starting address of the
*                     block of memory
*                     to be written to eeprom.
*         block_to_go -> The length of the block,
*                     1 writes one byte,
*                     0 writes none.
* OUTPUTS   - none
* DESTROYS  - ee_addr, mem_addr , block_to_go,
*             Acc. and X
*
WRITE:
            jsr   SUSPI             ; Ensure that the SPI is set up.

#IF         AUTOERASE
            jsr  EWEN               ; Open the eeprom for writing.
#ENDIF

#IF         9346FORM16
            INIT16                  ; Adjust the length of the block to
                                    ; be sent.
#ENDIF

#IF         9356FORM16
            INIT16                  ; Adjust the length of the block to
                                    ; be sent.
#ENDIF

#IF         9366FORM16
            INIT16                  ; Adjust the length of the block to
                                    ; be sent.
#ENDIF

WRLP:                               ; <- This is where the loop starts
                                    ; for repetitive writes to the                 ; eeprom
; eeprom until the block to be
                                    ; written is zero.
#IFNOT AUTOERASE
            lda   ee_addr           ; eeprom address to be written,
            jsr   ERASE             ; Erase the cell if not autoerase.
#ENDIF

            lda   #WRITE1           ; Load the first part of the write
                                ; command.
            bset  CSLINE,CSPORT     ; Select the Eeprom
```

```
            jsr   EESEND            ; Send the first part of the write
                                    ; command out SPI.
            lda   ee_addr           ; eeprom address to be written.
            inca                    ; Increment it for next byte.
            sta   ee_addr           ; Update eeprom address to be
                                    ; written.
            deca                    ; Decrement eeprom address to be
                                    ; written for this byte.
            and   #MASK             ; AND with address mask for this
                                    ; type and form of eeprom.
            ora   #WRITE2           ; OR with Write command #2.
            jsr   EESEND            ; Send address and WRITE2 out SPI.

            WRBYTE                  ; Send a byte to be written out SPI.

#IF         9346FORM16
            WRBYTE                  ; If 16 bit form Send the second
                                    ; byte to be written out SPI.
#ENDIF

#IF         9356FORM16
            WRBYTE                  ; If 16 bit form Send the second
                                    ; byte to be written out SPI.
#ENDIF

#IF         9366FORM16
            WRBYTE                  ; If 16 bit form Send the second
                                    ; byte to be written out SPI.
#ENDIF
            bclr  CSLINE,CSPORT     ; deselect the Eeprom

            jsr   WAIT              ; <- Waits here until the byte
                                    ; is written.
            tsta                    ; Acc still has block_to_go.
            bne   WRLP              ; Loop until the block left is zero.
WRDONE:                             ; All done writing.
            jsr   EWDS              ; "Close" or Write protect the
                                    ; eeprom
            rts

*******************************************
*  reading - The following is used to read
*            form 16 configured eeproms.
*
#MACRO RD_BYTE
            jsr   EESEND            ; Read a byte from the eeprom
                                    ; through the SPI.
            ldx   block_to_go       ; Load the length left to read.
            beq   NOSAVE            ; Only store if length is left.
            decx                    ; Decrement the block counter.
            stx   block_to_go       ; Update the block to go length.
            ldx   mem_addr          ; Load location where the byte from
                                    ; eeprom is to go.
            sta   ,x                ; Store the byte from eeprom to
                                    ; memory.
```

```
            incx                    ; Increment the location for the
                                    ; next byte.
            stx   mem_addr          ; Update the memory address for the
                                    ; next read.
NOSAVE:                             ; jump to here if end of block
                                    ; occurs.
#MACROEND

*****************************************;
* READ - This subroutine reads a block of
*        data out of the eeprom and places it
*        in a block of 6805 memory.  It has the
*        autosequence feature as an option.
*
* INPUTS    - The following memory locations
*             set up as follows.
*         ee_addr  -> contains the eeprom
*                     address where the data
*                     block starts.
*
*         mem_addr  -> contains the absolute
*                     starting
*                     address of the 6805
*                     memory block
*                     destination.
*
*         block_to_go -> The length of the block,
*                     1 reads one byte,
*                     0 reads none.
* OUTPUTS   - a block of updated memory
* DESTROYS  - ee_addr, mem_addr, block_to_go,
*             Acc. and X
*
*
READ:
            jsr   SUSPI             ; Ensure that the SPI is set up.

            jsr   EWDS              ; Ensure that the eeprom is write
                                    ; protected.
RDNLP:
            ldx   block_to_go       ; Load in the length of block to
                                    ; read.
            tstx                    ; Test for a zero length block.
            beq   RDNDONE           ; Test length to see if done.
            decx                    ; Decrement the length of the block.
            stx   block_to_go       ; Update block length for next loop.

#if         9366FORM16
            lda   ee_addr           ; Bring in eeprom address
            lsla                    ; Place MS Bit in carry.
            clra                    ; Zero out the accumulator.
            rola                    ; MS Bit of ee_address is LS Bit of
                                    ; accumulator.
            ora   #READ1            ; Overlay first part of read
                                    ; command.
```

```
#ELSEIF
            lda   #READ1            ; Load the first part of read
                                    ; command.
#ENDIF
            tst   SPSR              ; clean out the SPI receiver.
            tst   SPIDAT            ; Ensure SPI has no old data in it.
            bset  CSLINE,CSPORT     ; Select the Eeprom
            jsr   EESEND            ; Send READ1 command.
            lda   ee_addr           ; Load in eeprom address.
            and   #MASK             ; Mask in only valid address bits.
            lsla                    ; Shift left to create dummy clock
                                    ; idiosyncratic to READ.
            ora   #READ2            ; OR address with READ#2.
            jsr   EESEND            ; Send READ2 out the SPI.
            lda   ee_addr           ; Load accumulator with eeprom
                                    ; address.
            inca                    ; Increment eeprom address for next
                                    ; pass.
            sta   ee_addr           ; Update eeprom address.
            clra                    ; Clear the accumulator to read
                                    ; eeprom.
            jsr   EESEND            ; Read first byte.
            ldx   mem_addr          ; Load X with the location to store
                                    ; read byte.
            sta   ,x                ; Store the read byte.
            incx                    ; Increment X in preparation for
                                    ; next read.
            stx   mem_addr          ; Update the memory address.

#IF         AUTOSEQ
WRLOOP:                             ; Tighter loop for autosequence
                                    ; eeproms.
            RD_BYTE                 ; Read a byte from the eeprom +
                                    ; Store it.
            bne   WRLOOP            ; If block length = 0, all done,
                                    ; else loop.
            bclr  CSLINE,CSPORT     ; deselect the Eeprom
            bra   RDNDONE           ; Branch to out.
#ENDIF

#IF         9346FORM16
            RD_BYTE                 ; Read a byte from the eeprom +
                                    ; Store it.
#ENDIF

#IF         9356FORM16
            RD_BYTE                 ; Read a byte from the eeprom +
                                    ;  Store it.
#ENDIF

#IF         9366FORM16
            RD_BYTE                 ; Read a byte from the eeprom +
                                    ; Store it.
```

```
#ENDIF
            bclr  CSLINE,CSPORT     ; deselect the Eeprom
            bra   RDNLP             ; Branch to set up command and
                                    ; address necessary
                                    ; for non autosequenced eeproms.
RDNDONE:                            ; Branch to here when all done.
            rts


*****************************************;
* START - Sample calling of routines.
*
BSTART  EQU  0                      ; Start eeprom addresses for these
                                    ; examples.
BL_LEN  EQU  $80                    ; Length of block for these
                                    ; examples.

STARTRD:
            jsr   IDIO              ; ensure the ports are set up
                                    ; for this particular test set up.
            lda   #BSTART           ; Start reading eeprom at
                                    ; address BSTART.
            sta   ee_addr           ; Place first eeprom address in ; ee_addr.
            lda   #data             ; Load in start address of receiving
                                    ; memory.
            sta   mem_addr          ; Place start address in mem_addr.
            lda   #BL_LEN           ; Length of block to read in.
            sta   block_to_go       ; Store block length.
            jsr   READ              ; Read the eeprom.
            bra   $                 ; jump to this location
                                    ; (do nothing else).

STARTWR:
            jsr   IDIO              ; ensure the ports are set up
                                    ; for this particular test set up.
            lda   #data             ; Load in start address of receiving
                                    ; memory.
            sta   mem_addr          ; Place start address in mem_addr.
            lda   #BSTART           ; Start writing eeprom with bytes at
                                    ; address BSTART and up.
            sta   ee_addr           ; Place first eeprom address in
                                    ; ee_addr.
            lda   #BL_LEN           ; Length of block to write to
                                    ; eeprom.
            sta   block_to_go       ; Store block length.
            jsr   WRITE             ; Write the block to the eeprom.
            bra   $                 ; jump to this location
                                    ; (do nothing else).

STARTERAL:
            jsr   IDIO              ; ensure the ports are set up
                                    ; for this particular test set up.
            jsr   ERAL              ; Erases the entire serial eeprom
            bra   $                 ; jump to this location
                                    ; (do nothing else).
```

```
STARTWRL:
            jsr   IDIO              ; ensure the ports are set up
                                    ; for this particular test set up.
            lda   #$a5              ; (write $a5 to form 8 eeprom.)

#IF         9346FORM16
            ldx   #$c3              ; write $a5c3 to form 16 eeprom.
#ENDIF

#IF         9356FORM16
            ldx   #$c3              ; write $a5c3 to form 16 eeprom.
#ENDIF

#IF         9366FORM16
            ldx   #$c3              ; write $a5c3 to form 16 eeprom.
#ENDIF
            jsr   WRAL              ; 0xa5 to all memory locations in
                                    ; the eeprom.
*           bra   $                 ; jump to this location
                                    ; (do nothing else).


STARTERSE:
            lda   #$05              ; Load A with the eeprom address to
                                    ; be erased.
            jsr   ERASE             ; Erases memory location 5 of the
                                    ; eeprom.
            bra   $                 ; jump to this location
                                    ; (do nothing else).


      ORG VECTORS

VECSPI:  fdb   STARTRD              ; SPI VECTOR
VECSCI:  fdb   STARTRD              ; SCI VECTOR
VECTMR:  fdb   STARTRD              ; TIMER VECTOR
VECIRQ:  fdb   STARTRD              ; IRQ VECTOR
VECSWI:  fdb   STARTRD              ; SWI VECTOR
VECRST:  fdb   STARTRD              ; START VECTOR
```

Appendix J — SPI to EEPROM Using Interrupt Application Source

```
*~~~~~~~~~~~~~~~~MACRO~~~FOR~~WRITING~~~~~~~~~~~~~~~~~~~~~~~~~
* This writes a block of memory starting at absolute address
* "RAM_start" of length "length" to the eeprom starting at
* its absolute address "ee_start."

#MACRO WRBLOCK          ee_start,RAM_start,length
            jsr   CK_CLR            ; Ensure the eeprom is free.
            lda   #%2               ; Get Start of block in memory.
            sta   mem_addr          ; Place memory start in proper
                                    ; place.
            lda   #%1               ; Get Start of block in destination
                                    ; eeprom.
            sta   ee_addr           ; Place eeprom destination start in
                                    ; proper place.
            lda   #%3               ; Get the full block length.
            sta   block_to_go       ; Place Block length in proper
                                    ; place.
            jsr   WRITE             ; Write block from memory to eeprom.
#MACROEND

RAM         EQU   $50               ; RAM starts at $50
ROM         EQU   $100              ; ROM starts at $100
VECTORS     EQU   $1ff4             ; RESET and interrupt vectors start
                                    ; at $1ff4.

****************************************
* Eeprom type and configuration switches
*
#SETNOT     9346FORM8               ; 9346 eeprom, 1 byte format.
#SETNOT     9346FORM16              ; 9346, 2 byte word format.
#SETNOT     9356FORM16              ; 9356, 2 byte word format.
#SET        9366FORM16              ; 9366, 2 byte word format.
*
* Use with AUTOERASE eeproms only.
*
#SETNOT     AUTOSEQ                 ; For eeproms that automatically
                                    ; sequence to the next cell when
                                    ; being read.


****************************************
*
* RAM - variables
*
*
*
      ORG   RAM
ee_addr     ds    1                 ; eeprom address stored here.
mem_addr    ds    1                 ; Block index stored here.
block_to_go ds    1                 ; Block length stored here.
flag        ds    1                 ; Flags for eeprom status.
```

```
* Two flags are usually used
WR          equ   0               ; Set for Write, reset for erase.
m_to_pr     equ   1               ; More to program flag.
data        ds    $ad             ; Rest of data space is data to be
                                  ; stored.

      ORG   ROM

****************************************
* Command set
#if         9346FORM8             ; Command set for 9346 in the byte
                                  ; wide form.
MASK        equ   %01111111       ; Mask of valid address bits
READ1       equ   %00000110       ; READ command padded to 16 bits.
READ2       equ   %00000000
EWEN1       equ   %00000010       ; Write enable command padded to 16
                                  ; bits.
EWEN2       equ   %01100000
EWDS1       equ   %00000010       ; Write protect command padded to 16
                                  ; bits.
EWDS2       equ   %00000000
WRITE1      equ   %00000010       ; Write command padded to 16 bits.
WRITE2      equ   %10000000
WRAL1       equ   %00000010       ; Write all command padded to 16
                                  ; bits.
WRAL2       equ   %00100000
ERASE1      equ   %00000011       ; Erase cell command padded to 16
                                  ; bits.
ERASE2      equ   %10000000
ERAL1       equ   %00000010       ; Erase all command padded to 16
                                  ; bits.
ERAL2       equ   %01000000
#endif

#if         9346FORM16            ; Command set for 9346 in the 16 bit
                                  ; wide form.
MASK        equ   %00111111       ; Mask of valid address bits
READ1       equ   %00000011       ; READ command padded to 16 bits.
READ2       equ   %00000000
EWEN1       equ   %00000001       ; Write enable command padded to 16
                                  ; bits.
EWEN2       equ   %00110000
EWDS1       equ   %00000001       ; Write protect command padded to 16
                                  ; bits.
EWDS2       equ   %00000000
WRITE1      equ   %00000001       ; Write command padded to 16 bits.
WRITE2      equ   %01000000
WRAL1       equ   %00000001       ; Write all command padded to 16
                                  ; bits.
WRAL2       equ   %00010000
ERASE1      equ   %00000001       ; Erase cell command padded to 16
                                  ; bits.
ERASE2      equ   %11000000
ERAL1       equ   %00000001       ; Erase all command padded to 16
                                  ; bits.
```

```
ERAL2       equ   %00100000
#endif

#if         9356FORM16              ; Command set for 9356 in the 16 bit
                                    ; wide form.
MASK        equ   %01111111         ; Mask of valid address bits
READ1       equ   %00001100         ; READ command padded to 16 bits.
READ2       equ   %00000000
EWEN1       equ   %00000100         ; Write enable command padded to 16
                                    ; bits.
EWEN2       equ   %11000000
EWDS1       equ   %00000100         ; Write protect command padded to 16
                                    ; bits.
EWDS2       equ   %00000000
WRITE1      equ   %00000101         ; Write command padded to 16 bits.
WRITE2      equ   %00000000
WRAL1       equ   %00000100         ; Write all command padded to 16
                                    ; bits.
WRAL2       equ   %01000000
ERASE1      equ   %00000111         ; Erase cell command padded to 16
                                    ; bits.
ERASE2      equ   %00000000
ERAL1       equ   %00000100         ; Erase all command padded to 16
                                    ; bits.
ERAL2       equ   %10000000
#endif

#if         9366FORM16              ; Command set for 9366 in the 16 bit
                                    ; wide form.
MASK        equ   %11111111         ; Mask of valid address bits
READ1       equ   %00001100         ; READ command padded to 16 bits.
READ2       equ   %00000000
EWEN1       equ   %00000100         ; Write enable command padded to 16
                                    ; bits.
EWEN2       equ   %11000000
EWDS1       equ   %00000100         ; Write protect command padded to 16
                                    ; bits.
EWDS2       equ   %00000000
WRITE1      equ   %00000101         ; Write command padded to 16 bits.
WRITE2      equ   %00000000
WRAL1       equ   %00000100         ; Write all command padded to 16
                                    ; bits.
WRAL2       equ   %01000000
ERASE1      equ   %00000111         ; Erase cell command padded to 16
                                    ; bits.
ERASE2      equ   %00000000
ERAL1       equ   %00000100         ; Erase all command padded to 16
                                    ; bits.
ERAL2       equ   %10000000
#endif
```

```
CMDLST                                  ; Command list.
            DB    WRAL1                 ; WRAL Write All is #0 in the
                                        ; command list.
            DB    WRAL2
            DB    ERAL1                 ; ERAL Erase All is #1 in the
                                        ; command list.
            DB    ERAL2
            DB    ERASE1                ; ERASE one cell is #2 in the
                                        ; command list.
            DB    ERASE2
            DB    WRITE1                ; WRITE a block of cells is #3 in
            DB    WRITE2                ; the command list.
            DB    EWDS1                 ; EWDS Write protect or close is #4
            DB    EWDS2                 ; in the command list.
            DB    EWEN1                 ; EWEN Enable write or open is #5 in
            DB    EWEN2                 ; the command list.

*
* PROGRAM
*
* The main subroutines are READ, EWEN, EWDS,
* WRITE, WRAL, ERASE, and ERAL.
*

CSPORT      EQU   0                     ; Eeprom Chip Select.
CSLINE      EQU   5                     ; porta.5, an output line.

SPCR        EQU   $0a                   ; Location of SPI control reg.
SPSR        EQU   $0b                   ; Location of SPI status reg.
SPIDAT      EQU   $0c                   ; Location of SPI data reg.

SPIIRON     EQU   %11010011             ; SPI and interrupt on
                                        ; with lowest possible
                                        ; baud rate.

*****************************************;
* SETUP - This routine initializes the flags
*        to the preset inactive condition.
*
SETUP:
            bclr  m_to_pr,flag          ; Initialize to no more to
                                        ; program into the eeprom.
            bclr  WR,flag               ; Not writing at initialization.
            rts
```

```
*****************************************;
* IDIO - This routine handles the idiosyncratic
*        requirements of the particular test
*        hardware used.  It may be deleted
*        in most applications
IDIO:
           bset  4,CSPORT+4        ; Output in this application.
           bset  4,CSPORT          ; Pulls up the "SS" line.
           bset  7,CSPORT+4        ; Output in this application.
           bset  7,CSPORT          ; Pulls up the "RESET" line.
           rts

*****************************************;
* SUSPI - Sets up the eeprom IO port and the
*         SPI to communicate with the eeprom
*         by polling.
*         Other tasks can share the SPI.
*
* INPUTS   - none
* OUTPUTS  - DDRA,SPI
* DESTROYS - Accumulator.
*
SUSPI:
           bset  CSLINE,CSPORT+4   ; Output for Chip Select.
           bclr  CSLINE,CSPORT     ; Initialize to not selected.
           lda   #%01010000        ; SPI enabled phase 0.
           sta   SPCR              ; Set up the SPI to phase 0.
           rts

*****************************************;
* SUSPIR - Sets up the SPI to communicate
*          with the eeprom with interrupts.
*          This is used to determine when the
*          eeprom is ready.
*
* INPUTS   - none
* OUTPUTS  - DDRA,SPI
* DESTROYS - Accumulator.
*
SUSPIR:
           lda   #SPIIRON          ; SPI enabled phase 0.
           sta   SPCR              ; Set up SPI with interrupt.
           cli                     ; Enable the interrupt.
           rts

*****************************************
* SUALT - This is an example alternate
*         set up of the SPI.  It runs at a
*         higher baud rate than the eeprom
*         SPI, and uses the interrupt.
*         However, use of the interrupt or
*         the higher baud rate is not necessary.
*
```

```
SUALT:
            lda   #%11010000        ; Interrupt with high baud rate.
            sta   SPCR              ; Set up the alternate SPI.
            bclr  CSLINE,CSPORT     ; ensure de-selection.
            cli                     ; Allow SPI interrupt.
            rts

*******************************************
* CLRSPI - This sets the SPI to the reset
*          condition.
*
CLRSPI:
            clr   SPCR              ; Shut off SPI.
            clr   SPSR              ; Zero status register.
            rts

*****************************************;
* EESEND - sends a byte through the SPI to
*          the serial eeprom and receives
*          a byte from the serial eeprom
*
* INPUTS    - accumulator, send to SPI
* OUTPUTS   - accumulator, response from SPI
*
* DESTROYS  - Accumulator
*
EESEND:
            sta   SPIDAT            ; Byte to send is in accumulator.
            brclr 7,SPSR,$          ; Should loop 3 times.
            lda   SPIDAT            ; Bring in what SPI has received.
            rts

*****************************************;
* SENDADR - Sends two bytes to the eeprom through
*           the SPI.
*           A code is read in X to determine the
*           command sent to the eeprom.  They cross
*           as follows:
*             0 = WRITE ALL
*             1 = ERASE ALL
*             2 = ERASE
*             3 = WRITE
*             4 = EWDS
*             5 = EWEN
*           If the command is ERASE or WRITE the
*           Eeprom address is included
*           else only the command is included.
*
* INPUTS    - Number for the command in "X"
* OUTPUTS   - none
* DESTROYS  - Accumulator and "X"
*
```

```
SENDADR:
            cpx   #6                ; Ensure the value is in bounds.
            bcc   ABSEN             ; If invalid, exit.
            txa                     ; Copy command code to A.
            lslx                    ; Scale X input as a word pointer.
            lsra                    ; If command code is 0, 1, 4, or 5
            lsra                    ; an address is not to be sent.
            bset  CSLINE,CSPORT     ; Select the eeprom here.
            bcc   SEN2              ; Jump to code which will not send
                                    ; address.
            lda   cmdlst,x          ; Bring in first byte of proper
                                    ; command.
            jsr   EESEND            ; Send that first byte of command
                                    ; out SPI.
            lda   ee_addr           ; Bring in eeprom address.
            inca                    ; Increment it for the next pass.
            sta   ee_addr           ; Update eeprom address for next
                                    ; pass.
            deca                    ; Decrement eeprom address for this
                                    ; pass.
            and   #MASK             ; Mask off non-address bits.
            ora   cmdlst+1,x        ; OR second part of command with address.
            bra   ADRDONE           ; Re-join the paths of this routine.

SEN2:                               ; No address send starts here.
            lda   cmdlst,x          ; Bring in first byte of proper
                                    ; command.
            jsr   EESEND            ; Send that first byte of command
                                    ; out SPI.
            lda   cmdlst+1,x        ; Bring in second byte of proper
                                    ; command.

ADRDONE:
            jsr   EESEND            ; Send out the second part of
                                    ; command with or without address.
ABSEN:                              ; Branch around invalid commands
                                    ; comes here.
            rts

*
*****************************************;
* EWEN - This subroutine enables erase and write
*        operations.  It in effect unlocks the
*        eeprom so that its cells may be
*        changed.
*
* INPUTS    - none
* OUTPUTS   - none
* DESTROYS  - Accumulator and "X"
*
EWEN:
            ldx   #$05              ; Bring in 5 for EWEN.
            jsr   SENDADR           ; Interpret command 5 as EWEN.
            bclr  CSLINE,CSPORT     ; Release the eeprom.
            rts
```

```
*****************************************;
* EWDS - This subroutine disables erase and
*        write operations so that data cannot be
*        inadvertently corrupted.  It in effect
*        locks the eeprom so that its cells
*        cannot be changed.
*
* INPUTS    - none
* OUTPUTS   - none
* DESTROYS  - Accumulator and "X"
*
EWDS:
            ldx   #$04              ; Bring in 4 for EWDS.
            jsr   SENDADR   PF      ; Interpret command 4 as EWDS.
            bclr  CSLINE,CSPORT     ; deselect the Eeprom
            rts

*****************************************;
* CK_CLR - This subroutine checks the status
*          of eeprom operation.  If the eeprom
*          is busy the routine loops until the
*          previous eeprom operation becomes
*          ready.  It also checks the SPI to
*          ensure that the SPI is interrupt
*          operating.
*
* INPUTS    - none
* OUTPUTS   - none
* DESTROYS  - Accumulator.
*
CK_CLR:
            brset CSLINE,CSPORT,NCLR; Selected?
            brset m_to_pr,flag,NCLR ; Programming?
            bra   READY             ; All Clear.

NCLR:                               ; Not Clear, something going on.
            cli                     ; Ensure interrupts are on.
            lda   SPCR              ; Bring in SPI control register
            cmp   #SPIIRON          ; Check SPI control reg.
            beq   CK_LOOP           ; If OK do not re- set up.
            jsr   SUSPIR            ; Re-set up the SPI.

CK_LOOP:
            brset CSLINE,CSPORT,$   ; Loop until eeprom de-selected.
            brset m_to_pr,flag,$    ; Loop until eeprom is free.

READY:
            rts
```

```
*****************************************;
* ERAL - This subroutine erases the entire
*        eeprom.  An erased cell will put a high
*        level on the DO line when read, but
*        due to inverting in READ, the result
*        will arrive as 0x00 in 6805 memory.
*        ERAL calls EWEN to allow erasure.
*
* INPUTS    - none
* OUTPUTS   - none
* DESTROYS  - all contents of eeprom
*
ERAL:
            jsr   SUSPI             ; Set up the SPI for polling.
            jsr   EWEN              ; Open the eeprom.
            ldx   #$01              ; 1 is the command list location for
                                    ; erase all.
            jsr   SENDADR           ; Interpret command 1 as ERAL.
            bra   SIRXIT            ; Set interrupt and exit.

*****************************************
*  writing - The following is used to
*            write-all form 16 configured
*            eeproms.
*
#MACRO      WRAL16
            lda   block_to_go       ; Bring back in the high byte to
                                    ; write.
            coma                    ; Compliment it.
            jsr   eesend            ; Send it to the eeprom for
                                    ; programming.
#MACROEND

*****************************************;
* WRAL - In FORM8 eeproms this subroutine
*        writes the byte in the accumulator to
*        every byte of the Eeprom.  In FORM16
*        eeproms the accumulator is written to
*        the most significant byte the X
*        register is written to the less
*        significant byte.
*
* INPUTS    - Accumulator ( and X for FORM16)
* OUTPUTS   - none
* DESTROYS  - Accumulator, X, ee_addr .
*
*
WRAL:
            sta   mem_addr          ; Store low byte to be written.
            stx   block_to_go       ; Store high byte to be written.
            jsr   SUSPI             ; Set up the SPI for polling.

            jsr   EWEN              ; Open the eeprom for writing.
            clrx                    ; 0 is code for Write all.
            jsr   SENDADR           ; Interpret Command 0 as Write All.
```

```
            lda   mem_addr          ; Bring back data for sending.
            coma                    ; Complement for writing.
            jsr   eesend            ; Send byte to eeprom for writing.

#IF         9346FORM16
            WRAL16                  ; Write upper byte of a 16 bit wide
                                    ; eeprom.
#ENDIF

#IF         9356FORM16
            WRAL16                  ; Write upper byte of a 16 bit wide
                                    ; eeprom.
#ENDIF

#IF         9366FORM16
            WRAL16                  ; Write upper byte of a 16 bit wide
                                    ; eeprom.
#ENDIF
            bra   SIRXIT            ; Set interrupt and exit.

*****************************************;
* ERASE - This subroutine Erases an eight
*         cell byte or 16 cell word in the
*         Eeprom.  The address of the cell is
*         located in the accumulator.  The
*         accumulator is returned unchanged.
*
* INPUTS    - The following memory locations
*             set up as follows.
*         ee_addr   -> contains the absolute
*                      address of where the
*                      erasure will start in
*                      the eeprom.
*         mem_addr  -> (not used)
*
*         block_to_go  -> The length of the block,
*                      1 writes one byte,
*                      0 writes none.
* OUTPUTS   - none
* DESTROYS  - ee_addr , mem_addr , block_to_go,
*             Accumulator and X
*
*
ERASE:
            jsr   SUSPI             ; Set up the SPI for polling.
            lda   block_to_go       ; Bring in the length of block to be
                                    ; erased.
            beq   WRDONE            ; When the block is 0 use the same
                                    ; finish as write.
            deca                    ; one less in the block to go.
            sta   block_to_go       ; Update the block to go.
            brset m_to_pr,flag,NOTER1 ; Check for first pass.
            jsr   EWEN              ; Open the eeprom for erasure.
            bset  m_to_pr,flag      ; Set programming function flags.
            bclr  WR,flag           ; Clear writing flag.
```

```
NOTER1:
            ldx   #$02              ; Erase is selection 2.
            jsr   SENDADR           ; Send address to erase.
            bra   SIRXIT            ; Set interrupt and exit.

*******************************************
*  writing - The following is used to write
*             form 16 configured eeproms.
*
#MACRO      WR_BYTE
            lda   ,x                ; X still points to the next byte,
            coma                    ; bring it into the acc. and
                                    ; complement.
            jsr   EESEND            ; Send the byte.
            incx                    ; Increment to point to the next
                                    ; byte.
            stx   mem_addr          ; Store updated pointer for the next
                                    ; pass.
            lda   block_to_go       ; Bring in the remaining block
                                    ; length.
            beq   done16            ; IF zero length we are done.
            deca                    ; IF not decrement the block length.
            sta   block_to_go       ; Update the block length.
done16:
#MACROEND

*****************************************;
* WRITE - This subroutine Writes a block of
*         eight cell bytes to the Eeprom.
*
* INPUTS    - The following memory locations
*             set up as follows.
*         ee_addr   -> contains the absolute
*                      address of where the
*                      data will start in
*                      the eeprom.
*         mem_addr  -> contains the absolute
*                      starting address of the
*                      block of memory
*                      to be written to eeprom.
*         block_to_go  -> The length of the block,
*                      1 writes one byte,
*                      0 writes none.
* OUTPUTS   - none
* DESTROYS  - ee_addr , mem_addr , block_to_go,
*             Accumulator and X
*
*
WRITE:
            jsr   SUSPI             ; Set up the SPI for sending data.
            lda   block_to_go       ; Bring in the block length left.
            beq   WRDONE            ; If zero block length we are finished.
            deca                    ; Decrement the block length.
            sta   block_to_go       ; Update block length.
            brset m_to_pr,flag,NOTWR1; Check for first pass.
```

```
            jsr   EWEN              ; IF first pass open the eeprom.
            bset  m_to_pr,flag      ; Set programming in process flags.
            bset  WR,flag           ; Set the writing flag.

NOTWR1:
            ldx   #$03              ; code for write is 3.
            jsr   SENDADR           ; Send the code for writing and ee
                                    ; address.
            ldx   mem_addr          ; Bring in the memory address as
                                    ; pointer.
            lda   ,x                ; Bring in the byte to be written.
            incx                    ; Increment the pointer for next
                                    ; pass.
            stx   mem_addr          ; Update memory address pointer.
            coma                    ; Complement the byte to be written.
            jsr   eesend            ; Send it to the eeprom to be
                                    ; programmed into it.

#IF         9346FORM16
            WR_BYTE                 ; Send the second byte to be
                                    ; programmed if form 16.
#ENDIF

#IF         9356FORM16
            WR_BYTE                 ; Send the second byte to be
                                    ; programmed if form 16.
#ENDIF

#IF         9366FORM16
            WR_BYTE                 ; Send the second byte to be
                                    ; programmed if form 16.
#ENDIF

SIRXIT:                             ; Put the eeprom into the busy ready
                                    ; mode.
            bclr  CSLINE,CSPORT     ; De-select the Eeprom
            bset  CSLINE,CSPORT     ; Re-Select the Eeprom
            jsr   SUSPIR            ; Set up SPI int on and lowest baud
                                    ; rate.
            clra                    ; Send Zeros out SPI.  Every 8
                                    ; clocks the
            sta   SPIDAT            ; interrupt will fire and the eeprom
                                    ; will be checked for ready.
            cli                     ; Enable SPI interrupt.
            rts

WRDONE:
        bclrm_to_pr,flag            ; no more to write.
        jsr   EWDS                  ; Write protect the eeprom.
        rts
```

```
*******************************************
*  reading - The following is used to read
*            form 16 configured eeproms.
*
#MACRO RD_BYTE
            jsr   EESEND            ; read a byte from an addressed
                                    ; eeprom.
            ldx   block_to_go       ; Read in the remaining block
                                    ; length.
            beq   RD16END           ; Only store if remaining length is
                                    ; non Zero.
            decx                    ; Decrement the remaining block
                                    ; length.
            stx   block_to_go       ; Update the remaining block length.
            coma                    ; Complement the byte read from the
                                    ; eeprom.
            ldx   mem_addr          ; Load the pointer with the address
                                    ; to place
                                    ; the byte read from memory.
            sta   ,x                ; Store the read byte to memory.
            incx                    ; Increment the memory address
                                    ; pointer.
            stx   mem_addr          ; Update the memory address pointer.
RD16END:
#MACROEND

*
*******************************************;
* READ - This subroutine reads a block of
*        data out of the eeprom and places it
*        in a block of 6805 memory.  It has the
*        autosequence feature as an option.
*
* INPUTS    - The following memory locations
*             set up as follows.
*         ee_addr   -> contains the eeprom
*                      address where the data
*                      block starts.
*
*         mem_addr  -> contains the absolute
*                      starting
*                      address of the 6805
*                      memory block
*                      destination.
*
*         block_to_go  -> The length of the block,
*                         1 reads one byte,
*                         0 reads none.
* OUTPUTS   - a block of updated memory
* DESTROYS  - ee_addr , mem_addr , block_to_go,
*             Accumulator and X
*
*
```

```
READ:
            jsr         SUSPI       ; Set up the SPI for polling.

            jsr         EWDS        ; Write protect the eeprom as a
                                    ; precaution. Anything worth reading
                                    ; is worth protecting.

RDNLP:                              ; Loop for non-autosequenced read.
            ldx   block_to_go       ; Read in the remaining bloc
                                    ; length.
            tstx                    ; Test length to see if done.
            beq   RDNDONE           ; IF done jump out of routine.
            decx                    ; decrement the block for this pass.
            stx   block_to_go       ; Store updated block length for
                                    ; checking on the next pass.

#if         9366FORM16
            lda   ee_addr           ; Bring in eeprom address
            lsla                    ; Place MS Bit in carry.
            clra                    ; Zero out the accumulator.
            rola                    ; MS Bit of ee_address is LS Bit of
                                    ; accumulator.
            ora   #READ1            ; Overlay read command #1.
#ELSEIF
            lda   #READ1            ; Bring in read command #1.
#ENDIF
            lda   #READ1            ; Bring in Read command #1.
            tst   SPSR              ; clean out the SPI status register.
            tst   SPIDAT            ; clean out the SPI receiver.
            bset  CSLINE,CSPORT     ; Select the Eeprom
            jsr   EESEND            ; Send READ command 1.
            lda   ee_addr           ; Bring in the eeprom address to be
                                    ; read.
            and   #MASK             ; Mask off non-valid bits of
                                    ; address.
            lsla                    ; Shift left to accommodate read
                                    ; transition clock.
            ora   #READ2            ; OR address with second part of
                                    ; READ command.
            jsr   EESEND            ; Send READ2 | ee_address.
            lda   ee_addr           ; Bring in ee_address.
            inca                    ; Increment ee_address.
            sta   ee_addr           ; Update ee_address for next pass.
            clra                    ; Clear Acc. so a logic low is sent
                                    ; to eeprom.
            jsr   EESEND            ; Read first byte.
            coma                    ; Complement byte, all data is
                                    ; complemented.
            ldx   mem_addr          ; Bring in pointer for memory.
            sta   ,x                ; Store the read byte in the memory
                                    ; location pointed to.
            incx                    ; Increment the memory pointer for
                                    ; next pass.
            stx   mem_addr          ; Update memory pointer.
```

```
#IF AUTOSEQ                         ; If an autosequence eeprom a
                                    ; smaller more efficient loop may be
                                    ; used.
WRLOOP:
            RD_BYTE                 ; Read byte and store.
            bne   WRLOOP            ; IF block to read is not zero, read
                                    ; more.
            bclr  CSLINE,CSPORT     ; deselect the Eeprom
            bra   RDNDONE           ; Branch to out of routine.  None of
                                    ; the rest of the code is used if
                                    ; the eeprom is autosequence.
#ENDIF

#IF         9346FORM16
            RD_BYTE                 ; Read byte and store.
#ENDIF

#IF         9356FORM16
            RD_BYTE                 ; Read byte and store.
#ENDIF

#IF         9366FORM16
            RD_BYTE                 ; Read byte and store.
#ENDIF
            bclr  CSLINE,CSPORT     ; deselect the Eeprom
            bra   RDNLP             ; Branch always, block check is done
                                    ; above.

RDNDONE:
            clr   flag              ; Clear the eeprom flag.
            jsr   SUALT             ; Set up alternate SPI.
            rts


*################################################
* SPI Interrupt handler
*
* SPI interrupt handler is only used to assess the
* eeprom's ready condition during erasure, or writing.
SPI:
            brclr CSLINE,CSPORT,SPIALT ; IF eeprom is not selected,
                                    ; then some other interrupt
                                    ; driven SPI service must be
                                    ; active.  Jump there.
            tst   SPSR              ; Test status register to reset
                                    ; interrupt.
            lda   SPIDAT            ; Read data, to reset interrupt,
                                    ; too.
            beq   NOTREADY          ; If Zero, the eeprom is not ready.
            bclr  CSLINE,CSPORT     ; IF ready, deselect for next
                                    ; command.
            brclr m_to_pr,flag,FINISH ; Single program cycle, go to
                                    ; end.
            lda   block_to_go       ; A WRITE is in process, check for
                                    ; done.
```

```
            beq   FINISH            ; No more to write, go to end.
            brset WR,flag,WRF       ; erase or write?
            jsr   ERASE             ; If erase, and not finished, erase
                                    ; another.
            bra   SPIDONE           ; Done for now.

WRF:
            jsr   WRITE             ; IF WRITE, and not finished, write
                                    ; another.
            bra   SPIDONE           ; Done for now.
NOTREADY:                           ; IF eeprom not ready send more
                                    ; clocks.
            clra                    ; Zero the accumulator to send
                                    ; eeprom logic low.
            sta   SPIDAT            ; Send clocks to the eeprom using
                                    ; SPI.

SPIDONE:
            rti

FINISH:
            bclr  m_to_pr,flag      ; Clear the more to program flag.
            jsr   CLRSPI            ; Reset SPI.
            jsr   SUALT             ; Set up the SPI for an alternate
                                    ; handler.
SPIALT:
            tst   SPSR              ; This is filler, put the
            lda   #$69              ; other SPI handler here
            sta   SPIDAT            ; if it will be interrupt
            rti                     ; driven.

*################################################

*****************************************;
* START - Sample calling of routines.
*
BSTART      EQU   0
BL_LEN      EQU   $80               ; Length of block for examples.

STARTRD:
            jsr   SETUP             ; Flags must be cleared on system
                                    ; start up.
            jsr   IDIO              ; Set up lines idiosyncratic of this
                                    ; hardware.
            jsr   CK_CLR            ; Ensure the eeprom is free.
            lda   #BSTART           ; Start reading eeprom at
                                    ; address BSTART.
            sta   ee_addr           ; Place that address in memory so
                                    ; program can get it.
            lda   #data             ; Get the location of the lowest
                                    ; vector block.
            sta   mem_addr          ; Place it in memory so the program
                                    ; can get it.
            lda   #BL_LEN           ; Length of block to read.
            sta   block_to_go       ; Place it in memory so the program
```

```
                                ; can get it.
            jsr   READ          ; Read whatever is in the eeprom.
            bra   $             ; jump to this location
                                ; (do nothing else).

STARTWR:
            jsr   SETUP         ; Flags must be cleared on system
                                ; start up.
            jsr   IDIO          ; Set up lines idiosyncratic of this
                                ; hardware.
            WRBLOCK   BSTART,data,BL_LEN ; See macro at the beginning.
            jsr   CK_CLR        ; Protect memory during write.

* Zero out all memory as a test.
            lda   #$ff          ; Place #ff in highest
            sta   $ff           ; place in lower RAM
            ldx   #$50          ; Start of lower RAM.
            clra                ; Acc = 0 to be written to lower
                                ; RAM.

LOOP2:
            sta   ,x            ; Place the Zero in Acc in memory
                                ; pointed to.
            incx                ; Increment memory pointed to.
            brset 1,$ff,LOOP2   ; When the last byte goes to 0,
                                ; done.
            bra   STARTRD       ; Read back from eeprom, should be
                                ; the same.

STARTERAL:
            jsr   SETUP         ; Flags must be cleared on system
                                ; start up.
            jsr   IDIO          ; Set up lines idiosyncratic of this
                                ; hardware.
            jsr   CK_CLR        ; Ensure the eeprom is free.
            jsr   ERAL          ; Erases the entire serial eeprom
            bra   $             ; (do nothing else).

STARTWRL:
            jsr   SETUP         ; Flags must be cleared on system
                                ; start up.
            jsr   IDIO          ; Set up lines idiosyncratic of this
                                ; hardware.
            lda   #$a5          ; (write $a5 to form 8 eeprom.)

#IF         9346FORM16
            ldx   #$c3          ; write $a5c3 to form 16 eeprom.
#ENDIF

#IF         9356FORM16
            ldx   #$c3          ; write $a5c3 to form 16 eeprom.
#ENDIF

#IF         9366FORM16
            ldx   #$c3          ; write $a5c3 to form 16 eeprom.
```

```
#ENDIF
            jsr   CK_CLR            ; Ensure the eeprom is free.
            jsr   WRAL              ; 0xa5 to all memory locations in
                                    ; the eeprom. ($a5c3 to 16 bit form)
*           bra   $                 ; (do nothing else).

STARTERSE:
            jsr   SETUP             ; Flags must be cleared on system
                                    ; start up.
            jsr   IDIO              ; Set up lines idiosyncratic of this
                                    ; hardware.
            lda   #$05              ; ee_address to start block erasure.
            sta   ee_addr           ; Place it in memory so the program
                                    ; can get it.
            lda   #3                ; Length of block to erase
            sta   block_to_go       ; Place it in memory so the program
                                    ; can get it.
            jsr   CK_CLR            ; Ensure the eeprom is free.
            jsr   ERASE             ; Erases memory location 5+ of the
                                    ; eeprom.
            bra   $                 ; (do nothing else).


IRQ:                                ; External interrupt.
            jsr   IDIO              ; Should never get here.
            rti

      ORG   VECTORS

VECSPI:     fdb   SPI               ; SPI VECTOR
VECSCI:     fdb   STARTRD           ; SCI VECTOR
VECTMR:     fdb   STARTRD           ; TIMER VECTOR
VECIRQ:     fdb   IRQ               ; IRQ VECTOR
VECSWI:     fdb   STARTWR           ; SWI VECTOR
VECRST:     fdb   STARTRD           ; START VECTOR
```


How to reach us:

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

Mfax™: RMFAX0@email.sps.mot.com – TOUCHTONE 602-244-6609, US & Canada ONLY 1-800-774-1848

INTERNET: http://www.mot.com/SPS/

JAPAN: Nippon Motorola Ltd.; Tatsumi-SPD-JLDC, 6F Seibu-Butsuryu-Center, 3-14-2 Tatsumi Koto-Ku, Tokyo 135, Japan. 81-3-3521-8315

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