

Using Microchip 93 Series Serial EEPROMs with Microcontroller SPI Ports

Author: Keith Pazul
Memory and ASSP Division

INTRODUCTION

Systems requiring embedded control are becoming more and more sophisticated, with microcontrollers required to control these systems increasing in complexity. Many microcontrollers today are being designed with built-in serial communication capability to be able to easily access other features that are not built in to the microcontroller itself. Devices like EEPROMs, A/Ds, D/As, LCDs etc. are all being built with a serial interface to reduce cost, size, pin count, and board area. There are many different serial interfaces on the market used to interface peripherals (I²C™, Microwire®, SPI, for example). One of the serial interfaces that is gaining in popularity is SPI (Serial Peripheral Interface). It is becoming more popular because of its communication speed, simultaneous full-duplex communication, and ease of programming.

Microchip PIC16C64/74 microcontrollers have a built-in serial port that can be configured as an SPI port. Currently, the Microchip Serial EEPROM product line does not support SPI interface Serial EEPROMs, however it is possible to use the 93 series devices on the SPI port. Any version of Microchip's 93 series devices can be communicated with via the SPI port of a PIC16C64/74. The code for this application note is written for a Microchip 93LC56/66, but talking to other 93 series devices can be accommodated with minor code changes. See the *Theory of Operation* section of this application note for more details on how this is accomplished. This code was verified by downloading to Microchip's PICMASTER™ in-circuit emulator (run at full speed with a 10 MHz crystal) and tested to make sure it writes (polling ready/busy pin to verify write cycle completion) and reads properly. A schematic (Figure 1) is included in this application note to describe exactly how the 93LC56/66 was connected to the PIC16C64/74.

The SPI interface was popularized by the Motorola 68HCXX microcontrollers. Microchip receives many requests for Motorola assembly code that uses the SPI port of the 68HC11 or 68HC05 to talk to Microchip serial EEPROMs. Because of this, Microchip has written 68HC11 assembly code to communicate with its 93 series devices via its SPI port. The program was downloaded to a 68HC11 evaluation board and tested

to make sure it writes (polling ready/busy pin to verify write cycle completion) and reads properly. A schematic (Figure 2) is included in this application note to describe exactly how the 93LC56/66 was connected to the microcontroller.

THEORY OF OPERATION

To use an SPI port to communicate with Microchip's 93 series Serial EEPROMs, the bytes to be output to the 93XXXX must be aligned such that the LSB of the address is the 8th bit (LSB) of a byte to be output. From there, the bits should fill the byte from right to left consecutively. If more than 8 bits are required, then two bytes will be required to be output. This same method will work for any 93 series device. A 93LC66 was chosen as the device to write this application note code for, so the following example will be for that particular device. The theory explained below will work for any 93 series device.

Since more than 8 bits are required to control a 93LC66, two consecutive bytes are required.

High Byte (where the start bit, op code bits, and address MSB reside)

| 0 | 0 | 0 | 0 | SB | OP1 | OP0 | A8 |

The High Byte is configured in the following format: SB is the start bit, OP1 is op code MSB, OP0 is op code LSB, and A8 is the 9th address bit that is required to address 512 bytes. The CS can be set before the byte is output because the leading 0's output to the 93xxxx prevent a start bit from being recognized by the 93xxxx until the first high bit is sent.

Low Byte (8 address LSBs)

| A7 | A6 | A5 | A4 | A3 | A2 | A1 | A0 |

The Low Byte contains A7-A0, which are the rest of the address bits required to access 512 bytes.

Data output from master MUST be set up on the falling edge of the clock so that it can be read from the 93XXXX on the next rising edge. Receiving data from the 93XXXX MUST also happen on the falling edge of the clock because the data is output from the 93XXXX on the rising edge of the clock. THIS REQUIRES THE CLOCK PHASE BIT OF THE SPI PORT TO BE OPPOSITE FOR RECEIVING THAN IT IS FOR TRANSMITTING. The clock phase needs to be toggled between 0 for transmitting data and 1 for receiving data. See source code for the exact spot where the clock phase bit needs to be changed.

AN613

FIGURE 1: PIC16C74 TO 93LC56/66 SCHEMATIC

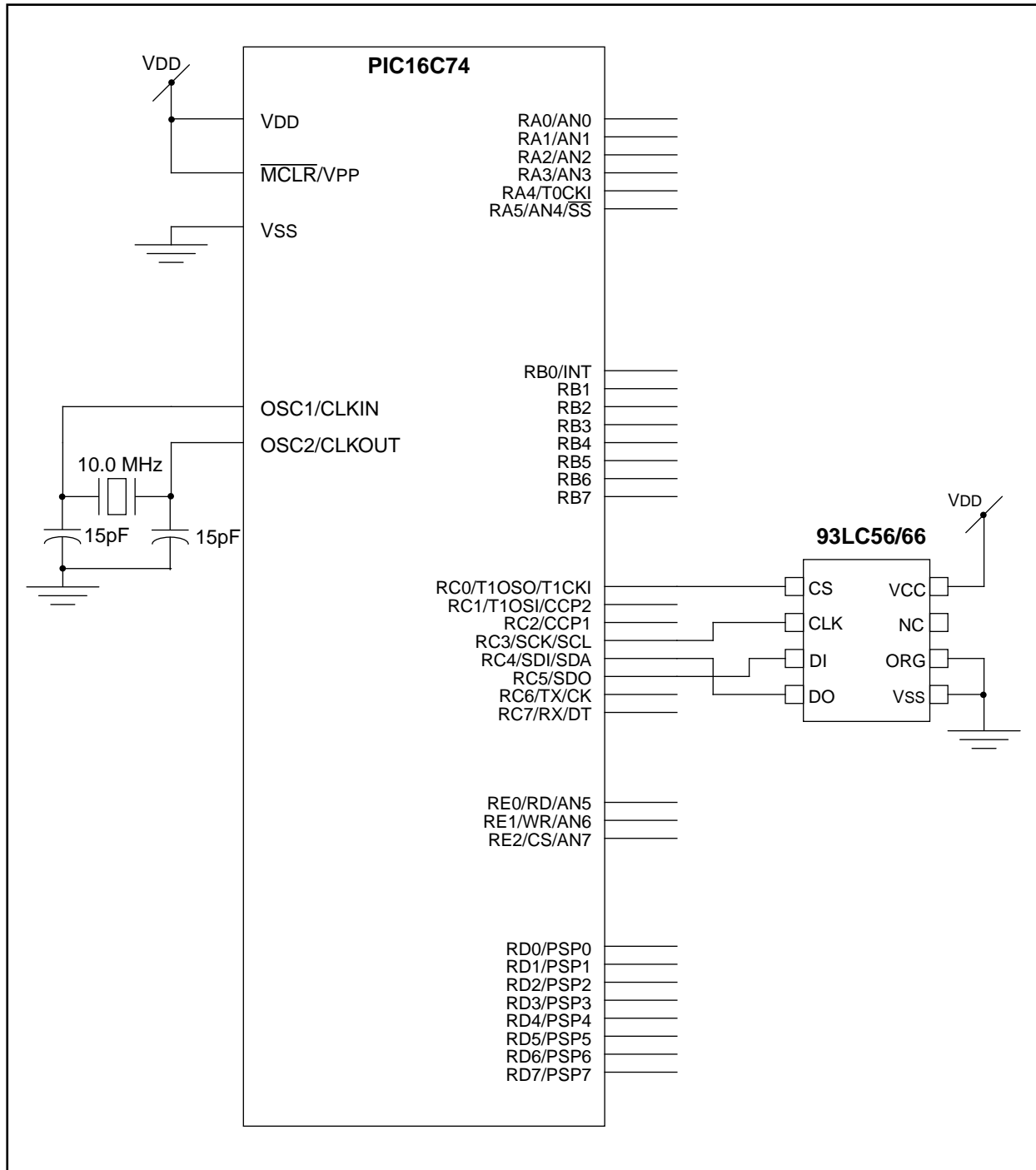
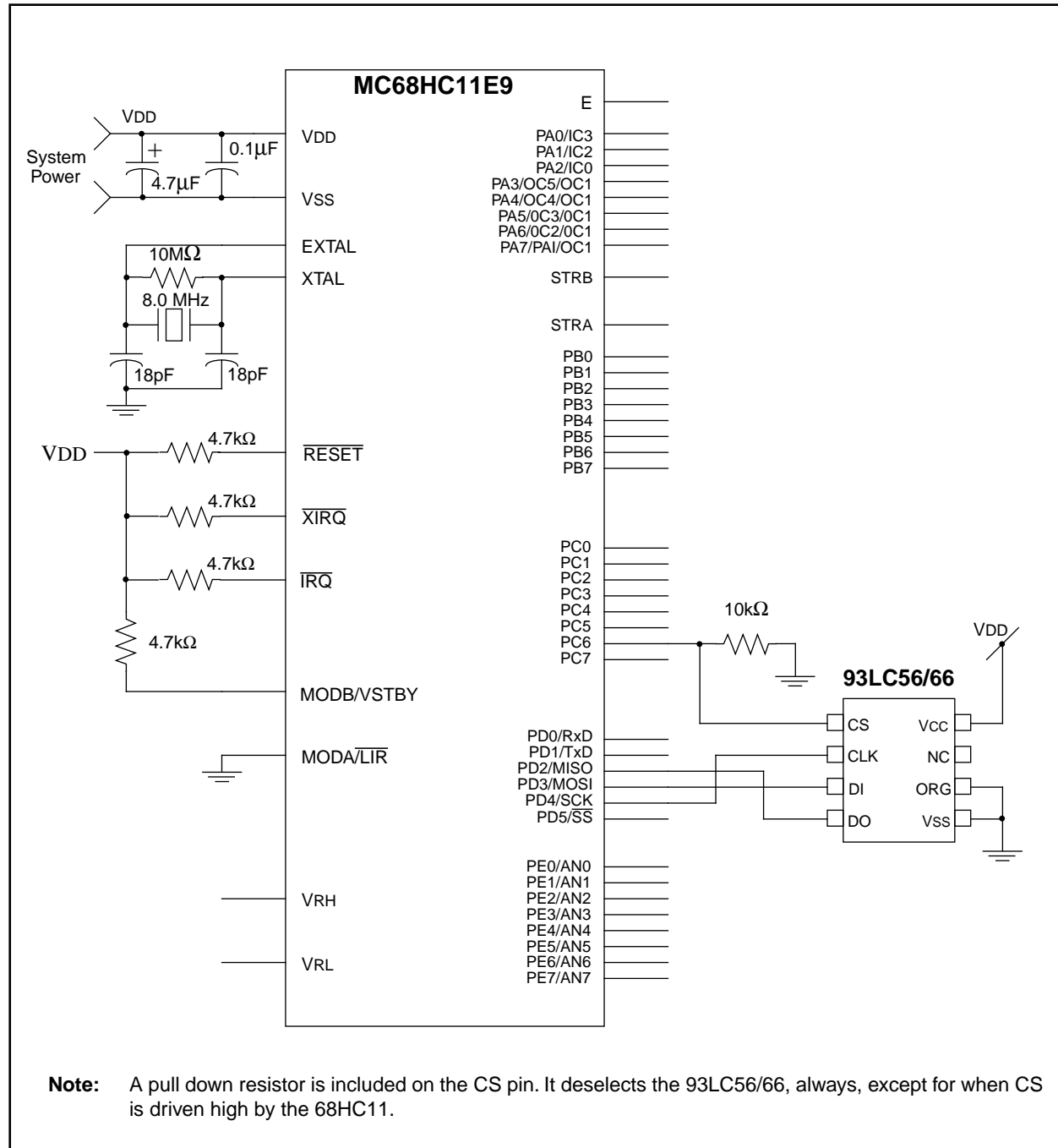


FIGURE 2: MOTOROLA 68HC11 TO MICROCHIP 93LC56/66 SCHEMATIC



Please check the Microchip BBS for the latest version of the source code. For BBS access information, see Section 6, Microchip Bulletin Board Service information, page 6-3.

APPENDIX A: PIC16C64/74 SOURCE CODE

```
*****
;
; To use the SPI port to communicate with 3-wire devices,
; the bytes to be output must be aligned such that the LSB of the
; address is the 8th bit (LSB) of a byte to be output. From there,
; the bits should fill the byte from right to left consecutively.
; This same method will work for any 93xxxx device. A 93LC66 was
; chosen as the device to write this application note code for,
; so the following example will be for that particular device.
; The theory explained below will work for any 93 series device.
;
; Since more than 8 bits are required to control a 93LC66,
; two consecutive bytes are required.
;
; High byte (where start bit, op code bits and address MSB reside)
;
; | 0 | 0 | 0 | 0 | SB | OP1 | OP0 | A8 |
;
; The High Byte is configured in the following format:
; SB is the start bit, OP1 is op code MSB, OP0 is op code LSB, and A8
; is the 9th address bit that is required to address 512 bytes. The CS
; can be set before the byte is output because the leading 0's output to
; the 93xxxx prevent a start bit from being recognized by the 93xxxx until
; the first high bit is sent.
;
; Low byte (8 address LSBs)
;
; | A7 | A6 | A5 | A4 | A3 | A2 | A1 | A0 |
;
; The Low Byte contains A8-A0, which are address bits required
; to access 512 bytes.
;
; The chip select is set high before sending the first byte out because
; the 93LC66 will not recognize a start bit until both CS and DI are
; high on the rising edge of a clock.
;
; This code is written to use the 16C64/74 SPI port to communicate with a
; Microchip 93LC66. The only other I/O line required besides the SPI port
; pins is a chip select. The ORG pin will be grounded to set up the part
; in x8 mode.
;
; PIN DESCRIPTIONS:
;
; SCK      (serial clock)      port C, bit 3
; SDO      (serial data out)   port C, bit 5
; SDI      (serial data in)    port C, bit 4
; CS       (chip select)       port C, bit 0
;
; Transmits from the master MUST happen on the falling edge of the clock
; so they can be read by the 93xxxx on the next rising edge of the clock.
; Receiving from the 93xxx MUST also happen on the falling edge of the
; clock because on the rising edge of the clock, the 93xxxx outputs its bit
; on its DO pin. This requires the CKP bit in the SSPCON register to be set
; to 1 for transmitting data and CKP=0 for receiving data.
;
; This code was written by Keith Pazul on 10/5/94
;
*****

*****
;
; Ram Register Definitions
```

```

;*****
;
; received bytes from EEPROM memory locations 10h to 13h
; will be stored in RAM registers 20h to 23h.
rxdata    equ        24h
txdata    equ        25h
addr      equ        26h
loops     equ        27h
loops2    equ        28h
hibyte    equ        29h
lobyte     equ       2Ah
datbyt    equ       2Bh
;*****
;
;      Other Definitions
;
;*****
;
;*****
;      Bit Definitions
;*****
;
cs         equ        0
sdi        equ        4
;*****
include "c:\mpasm\include\p16cxx.inc"    ; register map for PIC16CXX devices
org        0x000                        ; Reset Vector
goto      Start
;
;*****
;
;   This is the transmit/receive routine.  The received bytes
;   are don't cares until reading back from the array.
;
;*****
output      movwf      SSPBUF              ; place data in buffer so it
                                           ; can be output
loop1      bsf         STATUS, RP0        ; specify bank 1
           btfss       SSPSTAT, BF        ; has data been received (xmit done)?
           goto        loop1              ; not done yet, keep trying
           bcf         STATUS, RP0        ; specify bank 0
           movf        SSPBUF, W          ; empty receive buffer, even if we
                                           ; don't need received data
           movwf       rxdata              ; put received byte into location
           retlw       0                  ; return from subroutine
;*****
;*****
;
;   EWEN routine
;
;*****
EWEN
           bcf         STATUS, RP0        ; need to set bank 0
           bsf         PORTC, cs          ; set chip select line high
           movlw       b'00001001'        ; start bit is bit 3, 00 is
                                           ; op code for EWEN
           call        output              ;
           movlw       b'10000000'        ; 1 req for EWEN, 0000000 are don't cares
                                           ; which is the 8 lsb address bits
           call        output;
           bcf         PORTC, cs          ; bring chip select low to begin
                                           ; EEPROMs internal write cycle
           retlw       0                  ; return from subroutine
; remember CS must be low for at least 250 nsec
;*****

```

AN613

```
;*****
;
;   This routine outputs the two bytes required to send
;   the start bit, op code bits, and address bits
;
;*****
WRITE
    bcf      STATUS, RP0      ; need to set bank 0
    bsf      PORTC, cs        ; set chip select line high
    movf     hibyte, 0        ; put hibyte in w reg
    call     output           ;
    movf     FSR, 0           ; put addr pointed to by FSR into
                                ; w reg
    call     output           ;
    movf     datbyt, 0        ; get ready to output data in datbyt
    call     output           ;
    bcf      PORTC, cs        ; bring chip select low to begin
                                ; EEPROMs internal write cycle
    incf     FSR, 1           ; point to next location to
                                ; write to
    retlw    0
;*****
;*****
;
;   This is the module that reads one byte of data
;
;*****
READ
    bcf      STATUS, RP0      ; need to set bank 0
    bsf      PORTC, cs        ; set chip select line high
    bsf      SSPCON, CKP      ; make sure CKP is 1 to output
                                ; next instruction and addr
    movf     hibyte, 0        ; move data from hibyte to w
    call     output           ;
    movf     lobyte, 0        ; get ready to send next byte
                                ; which is the 8 lsb address bits
    call     output           ;
;*****
;   This is where CKP bit is reset for receiving data.
;*****
    bcf      SSPCON, CKP      ; change clock polarity to 0
    movlw    0x00            ; The byte xmitted here is a
                                ; don't care
    call     output           ;
    bcf      PORTC, cs        ; bring chip select low to
                                ; terminate read command
    clrf     INDF             ; clr location pointed to by FSR
    movf     rxdata, 0        ; move received data to w reg
    movwf    INDF            ; put received data in location
                                ; pointed to by FSR
    incf     FSR, 1           ; point to next location to
                                ; write to
    incf     lobyte, 1        ; next addr to read from
    retlw    0 ;
;*****
;*****
;*****
;*****
Start
    bcf      STATUS, RP0      ; need to set bank 0
    clrf     PORTC            ; initialize port c
    bsf      STATUS, RP0      ; need to set bank 1
    movlw    0x10            ; all bits are outputs except SDI
    movwf    TRISC           ; for SPI input
    clrf     PIE1             ; disables all peripheral ints
    clrf     INTCON           ; disables all interrupts
```

```

        bcf      STATUS, RP0      ; need to set bank 0
        clrf     SSPCON           ; clear SSP control register
        movlw    0x31             ; SPI master, clk/16, ckp=1
        movwf    SSPCON           ; SSPEN enabled
        call     EWEN             ; output EWEN for enabling writes
;*****
;   The next thing we will do is to write 0x5A to locations
;   10h through 13h.
;*****
        movlw    b'00001010'      ; start bit is bit 3, 01 is
                                   ; op code for write
        movwf    hibyte           ; load into hibyte

        movlw    0x10             ; put beginning address in FSR
        movwf    FSR             ; for later use
        movlw    b'01011010'      ; load 0x5A as data to be sent out
        movwf    datbyt          ;
wrnext   call     WRITE           ; call write subroutine
;*****
;   Ready/Busy poll to decide when write is complete
;   and the 93LC66 is available for writing the next
;   byte.
;*****
        nop                     ; cs must be low for > 250 ns
        bsf      PORTC, cs        ; and then be brought high
rbusy    btfss    PORTC, sdi      ; test ready/busy status
                                   ; if 1, internal write is done
        goto     rbusy           ; part still writing, stay in
                                   ; loop
        bcf      PORTC, cs        ; bring cs back low
        btfss    FSR, 2          ; have we written all 4 locations?
        goto     wrnext          ; no, then write next byte
;*****
;   Now, lets read back 10h through 13h (non-sequentially) and
;   store it in ram locations 20h through 23h in the 16C74.
;   With the Picmaster, I can read those memory locations
;   and see that it was read in properly. This is how
;   I can quickly verify that the read function is working
;   properly.
;*****
        movlw    0x20             ; where in RAM to begin storing
        movwf    FSR             ; data read back from EEPROM
        movlw    0x10             ; addr of where to begin reading
        movwf    lobyte          ; EEPROM from
        movlw    b'00001100'      ; start bit is bit 3, 10 is
        movwf    hibyte          ; op code for read
rdnext   call     READ           ;
        btfss    FSR, 2          ; have we 20h thru 23?
        goto     rdnext          ; no, then read next location

;*****
;   While program is in limbo routine, it is possible
;   to halt the processor with Picmaster and look at
;   the data contained 16C74 RAM locations 20h through 23h.
;*****
limbo    nop
        goto     limbo
;
;
        end

```

Please check the Microchip BBS for the latest version of the source code. For BBS access information, see Section 6, Microchip Bulletin Board Service information, page 6-3.

APPENDIX B: MOTOROLA 68HC11 SOURCE CODE

```
*****
; 3-Wire byte write and byte read using SPI port (220 bytes)
;
; This program (hcllsp66.*) writes 4 bytes of $5A to a Microchip 93LC66
; using the SPI port of the Motorola 68HC11 microcontroller. Ready/busy
; polling is used to determine when the current byte is done writing and
; the next byte is ready to be written to the 93LC66. After the 4 bytes
; are written, they are read back and stored in RAM locations $100-$103.
; The evaluation board can be stopped after receiving the 4 bytes to
; view RAM locations and verify that what was written to the 93LC66 is
; what is read back.
;
; This code was written by Keith Pazul on 3/7/95.
;
; This is the way the bytes will be aligned to be sent to the 93LC56/66:
;
; First byte (where start bit, op codes reside)
;
; | 0 | 0 | 0 | 0 | SB | OP1 | OP0 | A8 |
;
; where SB is start bit, OP1 is op code msb, OP0 is op code lsb,
; and A8 is address msb.
;
; Next byte (address)
;
; | A7 | A6 | A5 | A4 | A3 | A2 | A1 | A0 |
;
; where A8-A0 are address bits required to address 4k bits of memory.
;
; This code will work for any 93 series device. The only difference
; is that the number of address bits is adjusted and the start bit
; and op code bits are adjusted to follow directly after the address
; MSB.
;
; THE 93LC56 or 66 IS ASSUMED TO BE IN x8 MODE. IT REQUIRES
; THE USER TO TIE THE ORG PIN TO Vss.
;
; This program was written using a 68CH11EVBU evaluation board.
; This board has a monitor program in firmware of the 68HC11 which
; allows single stepping, register viewing, modifying, etc. Since this
; is the case, the program code will be loaded into the on-chip EEPROM.
; This EEPROM begins at $B600. The control registers are left to thier
; default location of $1000 and the RAM is left to its default location.
; RAM locations $48-$ff are used by the monitor program and are not
; available for program use. Therefore, the stack pointer is set a $47
; and will be able to use all of the RAM to $00, and the RAM variables
; begin at location $100 and go up from there. I cannot program the
; reset vector since it is ROM space (at $FFFE), so the way I run this
; program is to use the monitor program that comes with the evaluation
; board to set the program counter to the starting address of my user
; program ($B600) and begin from there. For users who do have access
; to the reset vector, the label of the beginning program (in my case, it
; is called START) should be loaded at location $FFFE. This program was
; not assembled using a Motorola assembler, but was assembled using
; Universal Cross-Assemblers Cross-32 Meta-Assembler. It has the
; ability to assemble just about any microcontroller code. There are
; certain commands that are unique to the cross-assembler. These
; commands will be commented differently than other comments to
; be recognizable. They will look like this:
;
*****
```



```

; Special cross-assembler command(s)
;+++++

; I do not know the exact assembler commands required to accomplish
; assembly using a Motorola assembler.
;
; The crystal that comes with the evaluation board is 8 Mhz.
;
; The SPI port in on port D. The bits are defined as follows:
;
;      MISO      PORT D, PIN 2 (pin 22 on package)
;      MOSI      PORT D, PIN 3 (pin 23 on package)
;      SCK       PORT D, PIN 4 (pin 24 on package)
;      SS\       PORT D, PIN 5 (pin 25 on package)
;      CS        PORT C, PIN 6
;
; Note that only the CS pin resides on port C.
;*****

;+++++
CPU      "C:\WINC32\68HC11.TBL"    ; LOAD TABLE
HOF      "MOT8"; Hex output is Motorola S-records
;+++++
;
;*****
; 68HC11 control register locations
;*****
REGBS    EQU      1000H            ; BEGINNING OF REGISTERS
RAMBS    EQU      100H            ; BEGINNING OF RAM VARIABLES
DDRC     EQU      REGBS+07H       ; DATA DIRECTION REG FOR PORT C
PORTC    EQU      REGBS+03H       ; PORT C DATA REGISTER
PCOFF    EQU      03H            ; OFFSET FROM CONTROL REG BEG.
PORTD    EQU      1008H          ; PORT D DATA REGISTER
DDRD     EQU      1009H          ; PORT D DATA DIRECTION REGISTER
SPCR     EQU      1028H          ; SPI CONTROL REGISTER
SPSR     EQU      1029H          ; SPI STATUS REGISTER
SPDR     EQU      102AH          ; SPE DATA REGISTER
;*****
;*****
; User defined constants
;*****
CSMASK   EQU      01000000B       ; BIT MASK FOR CHIP SELECT
SDIMASK  EQU      00000100B       ; BIT MASK FOR MISO PIN
;*****

LDS      #0047H                  ; STACK POINTER BEGINS AT $47

ORG      100H                   ; RAM variables begin at 100h

;+++++
; DFS is a Universal Cross-Assembler directive that stands for define
; storage. 1 is for byte, 2 is for word, 4 is for long word. These are
; the user defined RAM variables.

RXARRAY  DFS      1 * {4}         ; 100H-103H
RXDATA   DFS      1               ; 104H
DATBYT   DFS      1               ; 105H
HIBYTE   DFS      1               ; 106H
LOBYTE   DFS      1               ; 107H
RBTEST   DFS      1               ; 108H
ADDROFF  DFS      1               ; 109H
;+++++

;*****
; Program code cannot be placed in ROM for eval board because

```

AN613

```
; eval board firmware is loaded in ROM. Program code will be
; loaded in EEPROM (which is 512 bytes and begins at B600h).
;

                ORG      0B600H
;*****
;
; This is the main portion of the code. It is where the reset
; vector should set program counter to.
;
;*****

START
    LDX          #REGBS          ; LOADS BEG OF REGISTERS INTO X

    BCLR         PCOFF,X,CSMASK  ; MASK SURE CS IS CLEARED

    LDAA         #10110000B      ; SETS UP BITS 0-3 AND 6 AS INPUTS,
    STAA         DDRC            ; BITS 4,5, AND 7 AS OUTPUTS

    CLR          PORTC           ; CLEAR ALL PORT C BITS

    LDAA         #11111011B      ; ALL BITS ARE OUTPUTS EXCEPT MISO
    STAA         DDRD            ;

    LDAA         #01010010B      ; SPIE=0,SPE=1,DWOM=0,MSTR=1,
    STAA         SPCR            ; CPOL=0,CPHA=0, CLK/16

    LDAA         #SDIMASK        ; STORE READY/BUSY MASK IN
    STAA         RBTEST          ; LOCATION RBTEST

    LDAA         #10H            ;
    STAA         LOBYTE          ; LOAD 8 LSB'S OF ADDRESS

    CLR          ADDROFF         ; SET ADDRESS OFFSET TO 0 FOR
                                ; READ COMMANDS

    JSR          EWEN            ; SEND EWEN COMMAND

;*****
; Now lets write 5Ah out to address 10h
;*****

    LDAA         #00001010B      ; STRT BIT IS BIT 3, 01 IS
    STAA         HIBYTE          ; OP CODE FOR WRITE, LSB IS
                                ; ADDRESS

    LDAA         #01011010B      ; LOAD 0x5A IN DATA LOCATION
    STAA         DATBYT          ;

WRNEXT    JSR          WRITE      ; CALL WRITE ROUTINE

;*****
; ready/busy poll input pin to see when internal write
; cycle is complete.
;*****

    BSET         PCOFF,X,CSMASK  ; RAISE CS FOR READY/BUSY POLLING

RBUSY    LDAA         PORTD       ; INPUT PORT D
    ANDA         RBTEST          ; MASK OFF ALL BITS EXCEPT MISO
    BEQ          RBUSY          ; IF MISO IS LO, PART STILL BUSY,
                                ; ELSE WRITE IS COMPLETE

    BCLR         PCOFF,X,CSMASK  ; POLLING COMPLETE, END CHIP SELECT
```

```

        LDAA      LOBYTE      ; GET ADDRESS OFFSET
        BITA      #00000100B  ; ARE ALL BYTES WRITTEN YET?
        BEQ       WRNEXT      ; ALL BYTES HAVE NOT BEEN WRITTEN,
                                ; WRITE NEXT BYTE

;*****
;   Now lets read back the location of rxdata to see if we
;   read back what we wrote to that location.
;*****

        LDAA      #00001100B   ; STRT BIT IS BIT 3, 10 IS
        STAA      HIBYTE       ; OP CODE FOR READ, LSB IS
                                ; ADDRESS
        LDAA      #10H         ; RESET BEGINNING ADDRESS
        STAA      LOBYTE       ; FOR READ OPERATIONS

        LDY       #RAMBS       ; RX BUFF IS AT BEG OF RAM

RDNEXT   JSR       READ         ; CALL READ SUBROUTINE

        LDAA      LOBYTE       ; LOAD ACC WITH MASK THAT
                                ; CHECKS TO SEE IF 4 BYTES
                                ; HAVE BEEN WRITTEN
        BITA      #00000100B   ; CHECK TO SEE IF 4 BYTES HAVE
        BEQ       RDNEXT       ; BEEN WRITTEN. IF NOT, READ
                                ; NEXT BYTE.

        LDAA      #10110000B   ; DISABLES CS TO PROHIBIT
        STAA      DDRC         ; POSSIBILITY OF INADVERTANT
                                ; WRITES

LIMBO    NOP
        JMP       LIMBO

;*****
;
;   This subroutine outputs the data stored in TXBUFF
;
;*****

OUTPUT
LOOP1    STAA      SPDR         ; MOVE DATA FROM ACC A TO SPDR
        LDAB      SPSR         ; SEE IF XFER COMPLETE FLAG SET
        BPL       LOOP1        ; IF NOT, LOOP UNTIL SET

        LDAA      SPDR         ; MOVES RECEIVED BYTE FROM DATA REG
        STAA      RXDATA       ; TO RECEIVE DATA LOCATION

        RTS

;*****

;*****
;
;   EWEN routine
;
;*****
EWEN
        LDAA      #11110000B   ; SET CS FROM IN TO OUT. ALL
        STAA      DDRC         ; OTHER BITS ARE THE SAME

        BSET      PCOFF,X,CSMASK ; DROP CS TO BEGIN COMMAND

        LDAA      #00001001B   ; STRT BIT IS BIT 3, 00 IS
                                ; OP CODE FOR EWEN
        JSR       OUTPUT       ;

```

AN613

```
LDAA      #10000000B      ; 1 IS REQ FOR EWEN.  REST
                        ; ARE DON'T CARE BITS FOR
                        ; LOWER 8 ADDR BITS

JSR        OUTPUT        ;

BCLR       PCOFF,X,CSMASK  ; SET CS TO COMPLETE EWEN

RTS

;*****

;*****
;
;      WRITE routine
;
;*****
WRITE
      BSET      PCOFF,X,CSMASK  ; BEGIN WRITE COMMAND

      LDAA      HIBYTE          ; GET DATA FROM HIBYTE TO OUTPUT
      JSR        OUTPUT        ;

      LDAA      LOBYTE          ; 8 LSBs OF ADDRESS POINTED TO
      JSR        OUTPUT        ; BY ADDROFF OFFSET TO Y REG

      LDAA      DATBYT          ; OUTPUT DATA BYTE
      JSR        OUTPUT        ;

      BCLR       PCOFF,X,CSMASK  ; DROP CS TO BEGIN COMMAND

      INC        LOBYTE          ; INC ADDRESS FOR NEXT LOCATION
                        ; TO WRITE TO

      RTS

;*****

;*****
;
;      READ routine
;
;*****
READ
      LDAA      SPCR            ; READ IN SPI CONTROL REG
      ANDA      #11111011B      ; SET CPHA TO 0 FOR CONTROL
      STAA      SPCR            ; PORTION OF READ SEQUENCE

      BSET       PCOFF,X,CSMASK  ; DROP CS TO BEGIN COMMAND

      LDAA      HIBYTE          ; OUTPUT HI BYTE FOR READ CMD
      JSR        OUTPUT        ;

      LDAA      LOBYTE          ; 8 LSBs OF ADDRESS
      JSR        OUTPUT        ;

;*****
;*****
;*****
; To read a byte, a byte must be transmitted.  Here we xmit a byte
; of don't care bits to receive a byte.  WE MUST CHANGE THE CPHA
; FROM 0 TO 1 TO CHANGE BEFORE DATA IS RECEIVED.  IF NOT, THEN THE
; MICRO WILL TRY TO RECEIVE AT THE SAME TIME THE 93XX66 WILL BE
; TRANSMITTING.  THE SYMPTOM OF THIS NOT SET UP PROPERLY IS THAT
; THE DATA READ WILL BE SHIFTED RIGHT ONE BIT WITH MSB BEING 0.
;*****
;*****
;*****

      LDAA      SPCR            ; READ IN SPI CONTROL REG
```

```
ORAA      #00000100B      ; SET CPHA TO 1
STAA      SPCR

JSR       OUTPUT          ; XMIT A BYTE OF DON'T CARE BITS
                        ; TO RECEIVE A BYTE

BCLR      PCOFF,X,CSMASK   ; CLR CS TO END COMMAND

LDAA      RXDATA          ; GET DATA RECIEVED IN RXDATA
STAA      0, Y            ; AND STORE IN LOCATION POINTED
                        ; TO BY Y INDEX REG + 0 OFFSET

INY
INC       LOBYTE          ; INC ADDR OF NEXT BYTE TO READ
                        ; FROM

RTS
;*****

;+++++++
; This command tells the cross-assembler that code starts
; at the location of START

END       START
;+++++++
```

AN613

NOTES:

NOTES:

Note the following details of the code protection feature on PICmicro® MCUs.

- The PICmicro family meets the specifications contained in the Microchip Data Sheet.
- Microchip believes that its family of PICmicro microcontrollers is one of the most secure products of its kind on the market today, when used in the intended manner and under normal conditions.
- There are dishonest and possibly illegal methods used to breach the code protection feature. All of these methods, to our knowledge, require using the PICmicro microcontroller in a manner outside the operating specifications contained in the data sheet. The person doing so may be engaged in theft of intellectual property.
- Microchip is willing to work with the customer who is concerned about the integrity of their code.
- Neither Microchip nor any other semiconductor manufacturer can guarantee the security of their code. Code protection does not mean that we are guaranteeing the product as “unbreakable”.
- Code protection is constantly evolving. We at Microchip are committed to continuously improving the code protection features of our product.

If you have any further questions about this matter, please contact the local sales office nearest to you.

Information contained in this publication regarding device applications and the like is intended through suggestion only and may be superseded by updates. It is your responsibility to ensure that your application meets with your specifications. No representation or warranty is given and no liability is assumed by Microchip Technology Incorporated with respect to the accuracy or use of such information, or infringement of patents or other intellectual property rights arising from such use or otherwise. Use of Microchip's products as critical components in life support systems is not authorized except with express written approval by Microchip. No licenses are conveyed, implicitly or otherwise, under any intellectual property rights.

Trademarks

The Microchip name and logo, the Microchip logo, FilterLab, KEELOQ, microID, MPLAB, PIC, PICmicro, PICMASTER, PICSTART, PRO MATE, SEEVAL and The Embedded Control Solutions Company are registered trademarks of Microchip Technology Incorporated in the U.S.A. and other countries.

dsPIC, ECONOMONITOR, FanSense, FlexROM, fuzzyLAB, In-Circuit Serial Programming, ICSP, ICEPIC, microPort, Migratable Memory, MPASM, MPLIB, MPLINK, MPSIM, MXDEV, PICC, PICDEM, PICDEM.net, rPIC, Select Mode and Total Endurance are trademarks of Microchip Technology Incorporated in the U.S.A.

Serialized Quick Turn Programming (SQTP) is a service mark of Microchip Technology Incorporated in the U.S.A.

All other trademarks mentioned herein are property of their respective companies.

© 2002, Microchip Technology Incorporated, Printed in the U.S.A., All Rights Reserved.

 Printed on recycled paper.



Microchip received QS-9000 quality system certification for its worldwide headquarters, design and wafer fabrication facilities in Chandler and Tempe, Arizona in July 1999. The Company's quality system processes and procedures are QS-9000 compliant for its PICmicro® 8-bit MCUs, KEELOQ® code hopping devices, Serial EEPROMs and microperipheral products. In addition, Microchip's quality system for the design and manufacture of development systems is ISO 9001 certified.



WORLDWIDE SALES AND SERVICE

AMERICAS

Corporate Office

2355 West Chandler Blvd.
Chandler, AZ 85224-6199
Tel: 480-792-7200 Fax: 480-792-7277
Technical Support: 480-792-7627
Web Address: <http://www.microchip.com>

Rocky Mountain

2355 West Chandler Blvd.
Chandler, AZ 85224-6199
Tel: 480-792-7966 Fax: 480-792-7456

Atlanta

500 Sugar Mill Road, Suite 200B
Atlanta, GA 30350
Tel: 770-640-0034 Fax: 770-640-0307

Boston

2 Lan Drive, Suite 120
Westford, MA 01886
Tel: 978-692-3848 Fax: 978-692-3821

Chicago

333 Pierce Road, Suite 180
Itasca, IL 60143
Tel: 630-285-0071 Fax: 630-285-0075

Dallas

4570 Westgrove Drive, Suite 160
Addison, TX 75001
Tel: 972-818-7423 Fax: 972-818-2924

Detroit

Tri-Atria Office Building
32255 Northwestern Highway, Suite 190
Farmington Hills, MI 48334
Tel: 248-538-2250 Fax: 248-538-2260

Kokomo

2767 S. Albright Road
Kokomo, Indiana 46902
Tel: 765-864-8360 Fax: 765-864-8387

Los Angeles

18201 Von Karman, Suite 1090
Irvine, CA 92612
Tel: 949-263-1888 Fax: 949-263-1338

New York

150 Motor Parkway, Suite 202
Hauppauge, NY 11788
Tel: 631-273-5305 Fax: 631-273-5335

San Jose

Microchip Technology Inc.
2107 North First Street, Suite 590
San Jose, CA 95131
Tel: 408-436-7950 Fax: 408-436-7955

Toronto

6285 Northam Drive, Suite 108
Mississauga, Ontario L4V 1X5, Canada
Tel: 905-673-0699 Fax: 905-673-6509

ASIA/PACIFIC

Australia

Microchip Technology Australia Pty Ltd
Suite 22, 41 Rawson Street
Epping 2121, NSW
Australia
Tel: 61-2-9868-6733 Fax: 61-2-9868-6755

China - Beijing

Microchip Technology Consulting (Shanghai)
Co., Ltd., Beijing Liaison Office
Unit 915
Bei Hai Wan Tai Bldg.
No. 6 Chaoyangmen Beidajie
Beijing, 100027, No. China
Tel: 86-10-85282100 Fax: 86-10-85282104

China - Chengdu

Microchip Technology Consulting (Shanghai)
Co., Ltd., Chengdu Liaison Office
Rm. 2401, 24th Floor,
Ming Xing Financial Tower
No. 88 TIDU Street
Chengdu 610016, China
Tel: 86-28-6766200 Fax: 86-28-6766599

China - Fuzhou

Microchip Technology Consulting (Shanghai)
Co., Ltd., Fuzhou Liaison Office
Unit 28F, World Trade Plaza
No. 71 Wusi Road
Fuzhou 350001, China
Tel: 86-591-7503506 Fax: 86-591-7503521

China - Shanghai

Microchip Technology Consulting (Shanghai)
Co., Ltd.
Room 701, Bldg. B
Far East International Plaza
No. 317 Xian Xia Road
Shanghai, 200051
Tel: 86-21-6275-5700 Fax: 86-21-6275-5060

China - Shenzhen

Microchip Technology Consulting (Shanghai)
Co., Ltd., Shenzhen Liaison Office
Rm. 1315, 13/F, Shenzhen Kerry Centre,
Renminnan Lu
Shenzhen 518001, China
Tel: 86-755-2350361 Fax: 86-755-2366086

Hong Kong

Microchip Technology Hongkong Ltd.
Unit 901-6, Tower 2, Metroplaza
223 Hing Fong Road
Kwai Fong, N.T., Hong Kong
Tel: 852-2401-1200 Fax: 852-2401-3431

India

Microchip Technology Inc.
India Liaison Office
Divyasree Chambers
1 Floor, Wing A (A3/A4)
No. 11, O'Shaugnessey Road
Bangalore, 560 025, India
Tel: 91-80-2290061 Fax: 91-80-2290062

Japan

Microchip Technology Japan K.K.
Benex S-1 6F
3-18-20, Shinyokohama
Kohoku-Ku, Yokohama-shi
Kanagawa, 222-0033, Japan
Tel: 81-45-471- 6166 Fax: 81-45-471-6122

Korea

Microchip Technology Korea
168-1, Youngbo Bldg. 3 Floor
Samsung-Dong, Kangnam-Ku
Seoul, Korea 135-882
Tel: 82-2-554-7200 Fax: 82-2-558-5934

Singapore

Microchip Technology Singapore Pte Ltd.
200 Middle Road
#07-02 Prime Centre
Singapore, 188980
Tel: 65-334-8870 Fax: 65-334-8850

Taiwan

Microchip Technology Taiwan
11F-3, No. 207
Tung Hua North Road
Taipei, 105, Taiwan
Tel: 886-2-2717-7175 Fax: 886-2-2545-0139

EUROPE

Denmark

Microchip Technology Nordic ApS
Regus Business Centre
Lautrup høj 1-3
Ballerup DK-2750 Denmark
Tel: 45 4420 9895 Fax: 45 4420 9910

France

Microchip Technology SARL
Parc d'Activite du Moulin de Massy
43 Rue du Saule Trapu
Batiment A - 1er Etage
91300 Massy, France
Tel: 33-1-69-53-63-20 Fax: 33-1-69-30-90-79

Germany

Microchip Technology GmbH
Gustav-Heinemann Ring 125
D-81739 Munich, Germany
Tel: 49-89-627-144 0 Fax: 49-89-627-144-44

Italy

Microchip Technology SRL
Centro Direzionale Colleoni
Palazzo Taurus 1 V. Le Colleoni 1
20041 Agrate Brianza
Milan, Italy
Tel: 39-039-65791-1 Fax: 39-039-6899883

United Kingdom

Arizona Microchip Technology Ltd.
505 Eskdale Road
Winnersh Triangle
Wokingham
Berkshire, England RG41 5TU
Tel: 44 118 921 5869 Fax: 44-118 921-5820

01/18/02