Interfacing the X5043 Watchdog Timer to Hitachi H8/3000 Microcontrollers

by Applications Staff

The following code demonstrates how the Xicor X5043 SPI serial EEPROM and watchdog timer could be interfaced to the Hitachi H8/3000 microcontroller family when connected as shown in Figure 1. The interface uses four of the port pins available on H8 family devices to implement the interface.

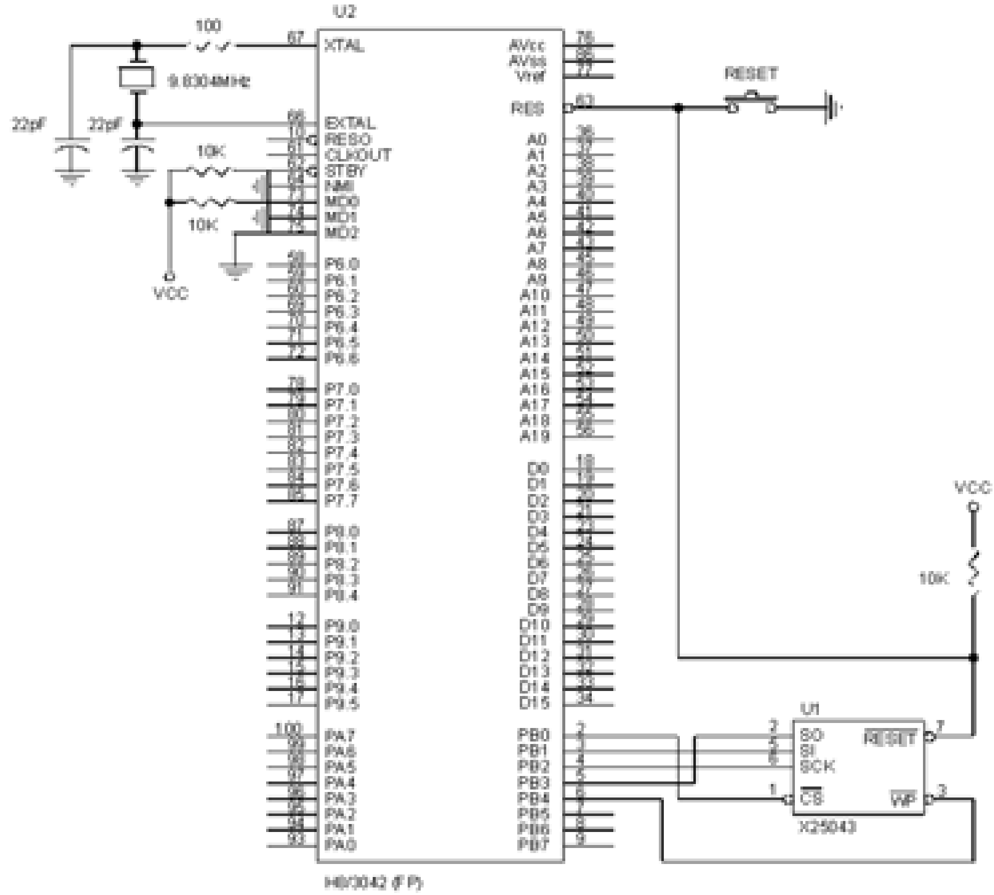

Figure 1. Typical hardware connection for interfacing an X5043 to the H8/3042 microcontroller

```
       .h8300h
       .file "h8x25043.asm"

;  h8x25043.asm

;  Demonstration code for interfacing Xicor X5043 to Hitachi H8/3042

;  Revision: 2.03  02/04/96

;  Expects the H8/3042 hard-wired in Mode 1: 8-bit bus-width, Expanded address
;  space (0x00000-0xFFFFF), Internal ROM disabled, Internal RAM enabled
;  (0xFF710-0xFFF0F), 9.8304 MHx xtal

      .include "h8stddef.inc"       ; H8/3042 standard register definitions

;  H8/3042 specific equates

      .equu H8RAMTOP,   0xFFF0F     ; Highest onboard RAM address (2048 bytes)
      .equ  H8RAMBOT,   0xFF710     ; Lowest onboard RAM address
      .equ  RXD0,       bit_2       ; Receive data pin of port/SCI
      .equ  INTMASK,    0b11000000  ; CCR Interrupt Mask (I and UI bits)

;  Xicor X5043 device-specific equates

      .equ  CS,         bit_0       ; Port bit for Chip Select input (H8 out)
      .equ  SI,         bit_1       ; Port bit for Serial data Input (H8 out)
      .equ  SCK,        bit_2       ; Port bit for Serial ClocK input (H8 out)
      .equ  SO,         bit_3       ; Port bit for Serial data Output (H8 in)
      .equ  WP,         bit_4       ; Port bit for Write Protect input (H8 out)

      .equ  XICOR,      PBDR        ; H8/3042 data port assignment (PB)
      .equ  XICORDDR,   PBDDR       ; H8/3042 data direction port assignment
      .equ  DDRSETUP,   0b11110111  ; Port Data Direction Register setup

      .equ  WREN,       0x06        ; WRite ENable latch instruction
      .equ  WRDI,       0x04        ; WRite DIsable latch instruction
      .equ  WRSR,       0x01        ; WRite Status Register instruction
      .equ  RDSR,       0x05        ; ReaD Status Register instruction
      .equ  WRITE,      0x02        ; WRITE memory instruction
      .equ  READ,       0x03        ; READ memory instruction

      .equ  STATUSREG,  0x00        ; Status Register

      .equ  WIPBIT,     bit_0       ; Write In Progress status bit
      .equ  ADDRBIT,    bit_3       ; Read/Write instruction address MSB
      .equ  PAGESIZE,   4           ; Bytes per page
      .equ  NUMPAGES,   128         ; Number of pages in X25043
      .equ  NUMBYTES,   NUMPAGES*PAGESIZE ; Number of bytes in X25043
      .equ  MAXPOLLS,   100         ; Maximum number of poll attempts for WIP

;  Main equates

      .equ  STACKTOP,   H8RAMTOP-4  ; Stack initializes here and builds down
      .equ  ALLONES,    0b11111111  ; All bits set
      .equ  BIGWRITE,   70          ; Test number of bytes to write
      .equ  RAMBFR0,    H8RAMBOT    ; Start of read/write buffer in on-board RAM
      .equ  RAMBFR1,    H8RAMBOT+PAGESIZE ; Second read/write buffer
      .equ  RAMBFR2,    RAMBFR1+BIGWRITE ; Third read/write buffer
      .equ  DATABYTE0,  0xF0        ; Test byte 0
      .equ  DATABYTE1,  0x0F        ;           1
      .equ  DATABYTE2,  0x55        ;           2/Power-up flag
      .equ  DATABYTE3,  0xAA        ;           3/
      .equ  PWRUPFLGS,  256*DATABYTE3+DATABYTE2 ; 0xAA55
      .equ  XADDRB,     45          ; Test byte write/read address
      .equ  XADDRP,     200         ; Test page write/read address
      .equ  XADDRBP,    0x195       ; Test block protect read/write address
      .equ  XADDRBW,    0x037       ; EEPROM BigWrite destination
      .equ  H8RAMLOC0,  H8RAMBOT+1047 ; Power-up RAM test location 0
      .equ  H8RAMLOC1,  H8RAMBOT+1597 ;                           1

;     Start of code
;     .org  0x00000                 ; H8/3042 Interrupt vector jump table
                                    ; Must reside at address 0x00000

      .long 0x00000100              ;  0 RESET
      .long _bogus_int              ;  1 reserved
      .long _bogus_int              ;  2 reserved
      .long _bogus_int              ;  3 reserved
      .long _bogus_int              ;  4 reserved
      .long _bogus_int              ;  5 reserved
      .long _bogus_int              ;  6 reserved
      .long _bogus_int              ;  7 external interrupt (NMI)
      .long _bogus_int              ;  8 Trap Instruction (4 sources)
      .long _bogus_int              ;  9 Trap Instruction (4 sources)
      .long _bogus_int              ; 10 Trap Instruction (4 sources)
      .long _bogus_int              ; 11 Trap Instruction (4 sources)
      .long _bogus_int              ; 12 External Interrupt IRQ0
      .long _bogus_int              ; 13 External Interrupt IRQ1
      .long _bogus_int              ; 14 External Interrupt IRQ2
      .long _bogus_int              ; 15 External Interrupt IRQ3
      .long _bogus_int              ; 16 External Interrupt IRQ4
      .long _bogus_int              ; 17 External Interrupt IRQ5
      .long _bogus_int              ; 18 reserved
      .long _bogus_int              ; 19 reserved
      .long _bogus_int              ; 20 WOVI Watchdog timer
      .long _bogus_int              ; 21 CMI  Refresh controller
      .long _bogus_int              ; 22 reserved
      .long _bogus_int              ; 23 reserved
      .long _bogus_int              ; 24 IMIA0 GRA0 compare match/input capture
      .long _bogus_int              ; 25 IMIB0 GRB0 compare match/input capture
      .long _bogus_int              ; 26 OVI0  overflow 0
      .long _bogus_int              ; 27 reserved
      .long _bogus_int              ; 28 IMIA1 GRA1 compare match/input capture
      .long _bogus_int              ; 29 IMIB1 GRB1 compare match/input capture
      .long _bogus_int              ; 30 OVI1  overflow 1
      .long _bogus_int              ; 31 reserved
      .long _bogus_int              ; 32 IMIA2 GRA2 compare match/input capture
      .long _bogus_int              ; 33 IMIB2 GRB2 compare match/input capture
      .long _bogus_int              ; 34 OVI2  overflow 2
      .long _bogus_int              ; 35 reserved
      .long _bogus_int              ; 36 IMIA3 GRA3 compare match/input capture
      .long _bogus_int              ; 37 IMIB3 GRB3 compare match/input capture
      .long _bogus_int              ; 38 OVI3  overflow 3
      .long _bogus_int              ; 39 reserved
      .long _bogus_int              ; 40 IMIA4 GRA4 compare match/input capture
      .long _bogus_int              ; 41 IMIB4 GRB4 compare match/input capture
      .long _bogus_int              ; 42 OVI4 overflow 4
      .long _bogus_int              ; 43 reserved
      .long _bogus_int              ; 44 DEND0A  DMAC group 0
      .long _bogus_int              ; 45 DEND0B  DMAC group 0
      .long _bogus_int              ; 46 DEND1A  DMAC group 0
      .long _bogus_int              ; 47 DEND1B  DMAC group 0
      .long _bogus_int              ; 48 reserved
      .long _bogus_int              ; 49 reserved
      .long _bogus_int              ; 50 reserved
      .long _bogus_int              ; 51 reserved
      .long _bogus_int              ; 52 ERI0 receive error       SCI chan 0
      .long _bogus_int              ; 53 RXI0 receive data full   SCI chan 0
      .long _bogus_int              ; 54 TXI0 transmit data empty SCI chan 0
      .long _bogus_int              ; 55 TEI0 transmit end        SCI chan 0
      .long _bogus_int              ; 56 ERI1 receive error       SCI chan 1
      .long _bogus_int              ; 57 RXI1 receive data full   SCI chan 1
      .long _bogus_int              ; 58 TXI1 transmit data empty SCI chan 1
      .long _bogus_int              ; 59 TEI1 transmit end        SCI chan 1
      .long _bogus_int              ; 60 ADI  A/D end

_bogus_int:
      orc         #0b11000000,ccr   ; Disable interrupts (I and UI)
      rte

;     .org        0x000100

_powerup:                           ; Initialization
      orc         #INTMASK,ccr      ; Disable interrupts (I and UI)
      mov.l       #STACKTOP,er7     ; Initialize the Stack pointer
      bsr         _init_xiport:16   ; Initialize the X25043 I/O port
      bset        #WP,@XICOR        ; Write Protect line high to enable X25043
                                    ; writes
      bsr         _wren_cmnd:16     ; Enable writing to the X25043 to ...
      mov.b       #0b00110000,r0l   ; Setup to disable the WDT
      bsr         _wrsr_cmnd:16     ; ... and do so!

_main:
      mov.l       #H8RAMLOC0,er1    ; Point to power-up flag location 0
      mov.b       @er1,r0l          ; ... and fetch the flag
      mov.l       #H8RAMLOC1,er2    ; ... the other flag, location 1
      mov.b       @er2,r0h          ; ... fetch it
      mov.w       #PWRUPFLGS,r3     ; Setup flag compare (and re-write)
      mov.b       r3l,@er1          ; Store first flag
      mov.b       r3h,@er2          ; ... and the second
                                    ;   (we've powered-up or reset)
      cmp.w       r0,r3             ; Has the WDT brought us here?
      beq         _skip_wdt         ;  Yes, skip the Watch Dog Timer exercise
```

```
;      Exercise the X5043 Watch Dog Timer one time

       bsr          _wren_cmnd:16      ;  No, enable writing to the X5043
       mov.b        #0b00100000,r0l    ; Setup the WDT for 200ms
       bsr          _wrsr_cmnd:16      ; ... and enable it
_wdt_delay:
       mov.w        #30788,r3          ; Setup loop count (100ms @9.8304MHz)
_wdtdly_loop:
       dec.w        #1,r3              ; Crank the counter
       bne          _wdtdly_loop       ; Done?
       bsr          _wdog_reset:16     ; Kick the dog (verifies WDT reset)
_wdt_wait:
       bra          _wdt_wait:16       ; Spin here until WDT kicks back to reset

;      Skip the Watch Dog Timer ... done once, now byte Read/Write exercise

_skip_wdt:
       xor.b        r0l,r0l            ; Setup null byte to clear power-up flags
       mov.l        #H8RAMLOC0,er1     ; Point to power-up flag location 0
       mov.b        r0l,@er1           ; ... and clear the flag
       mov.l        #H8RAMLOC1,er1     ; ... the other flag, location 1
       mov.b        r0l,@er1           ; ... and clear it (now reset recycles WDT)
       bsr          _rdsr_cmnd:16      ; Read the X5043 status register
       bsr          _wren_cmnd:16      ; Enable writing to the X5043
       mov.b        #DATABYTE0,r0l     ; Setup data for single byte write
       mov.w        #XADDRB,r1         ; Setup X5043 target address
       bsr          _byte_write:16     ; ... and write it
       mov.w        #XADDRB,r1         ; Setup X5043 source address
       bsr          _byte_read:16      ; ... and fetch it!
       bsr          _wren_cmnd:16      ; Enable writing to the X5043
       mov.b        #DATABYTE1,r0l     ; Setup data for single byte write
       mov.w        #XADDRB,r1         ; Setup X5043 target address
       bsr          _byte_write:16     ; ... and write it
       mov.w        #XADDRB,r1         ; Setup X5043 source address
       bsr          _byte_read:16      ; ... and fetch it!

;      Page Read/Write exercise

       bsr          _wren_cmnd:16      ; Enable writing to the X5043
       mov.w        #XADDRP,r1         ; Setup X5043 target page address
       mov.l        #_dataset0,er2     ; Point to source data
       bsr          _page_write:16     ; ... and write the page
       mov.w        #XADDRP,r1         ; Again, setup X5043 source page address
       mov.l        #RAMBFR0,er2       ; Point to destination buffer in onboard RAM
       mov.w        #PAGESIZE,e1       ; Setup byte read count
       bsr          _read_seq:16       ; ... and read the page

;      Sequential Read/Write exercise

       xor.w        r0,r0              ; Initialize byte sequence (0->BIGWRITE-1)
       mov.l        #RAMBFR1,er2       ; ... and point to the destination
_bigwr_setup:
       mov.b        r0l,@er2           ; Store a byte in the RAM buffer
       inc.l        #1,er1             ; Bump pointer to next RAM location
       inc.w        #1,r0              ; Count bytes stored/generate next byte
```

```
      cmp.w       #BIGWRITE,r0       ; Have we done 'em all?
      blt         _bigwr_setup       ;  No, continue

      mov.w       #BIGWRITE,e1
      mov.w       #XADDRBW,r1
      mov.l       #RAMBFR1,er2       ; Setup X5043 'big write' (count is in e1)
      bsr         _wren_cmnd:16      ; Enable writing to the X5043
      bsr         _write_seq:16      ; ... and write 'em

      mov.w       #BIGWRITE,e1       ; Setup to read 'em back!
      mov.w       #XADDRBW,r1
      mov.l       #RAMBFR2,er2
      bsr         _read_seq:16

;     Block Protect exercise

      bsr         _wren_cmnd:16      ; Enable writing to the X5043
      mov.b       #0b00110100,r0l    ; Setup Block Protect upper quarter (0x180+)
      bsr         _wrsr_cmnd:16      ; ... and protect it
      mov.w       #XADDRBP,r1        ; Setup X25043 source address (protected)
      bsr         _byte_read:16      ; ... and fetch it!
      not.b       r0l                ; One's complement, for re-write verify
      mov.b       r0l,r1l            ; Temporarily preserve it
      bsr         _wren_cmnd:16      ; Enable writing to the X5043
      mov.b       r1l,r0l            ; Recover data for single byte write
      mov.w       #XADDRBP,r1        ; Setup X5043 target address (protected)
      bsr         _byte_write:16     ; ... and write it
      mov.w       #XADDRBP,r1        ; Setup X5043 source address
      bsr         _byte_read:16      ; ... and fetch it!

_end_loop:
      bra         _end_loop          ; Terminal infinite loop

; X25043 Interface Subroutines

; Name:      _init_xiport
; Function:  Initializes the H8 I/O port bit directions
; Calls:     None
; Expects:   Nothing
; Returns:   Nothing
; Registers: r0l
; Remarks    Must be called once initially to setup the I/O port

_init_xiport:
      mov.b       #DDRSETUP,r0l
      mov.b       r0l,@XICORDDR
      bset        #CS,@XICOR         ; Chip Select disable X5043
      bclr        #SI,@XICOR         ; Drop Serial Input to X5043
      bclr        #SCK,@XICOR        ; Drop the clock SCK
      bclr        #WP,@XICOR         ; Write Protect line low, disable writes
      rts

; Name:      _wren_cmnd
; Function:  Sends the command to enable writing to the Xicor X5043 EEPROM
; Calls:     _send_byte
; Expects:   Nothing
; Returns:   Nothing
; Registers: r0l
; Remarks:

_wren_cmnd:
      bclr        #SCK,@XICOR       ; SCK low
      bclr        #CS,@XICOR        ; CS low
      mov.b       #WREN,r0l         ; Setup Write Enable instruction
      bsr         _send_byte:16
      bclr        #SCK,@XICOR       ; SCK low
      bset        #CS,@XICOR        ; CS high
      rts
; Name:     _wrdi_cmnd
; Function:  Sends the command to disable writing to the Xicor X5043 EEPROM
; Calls:    _send_byte
; Expects:   Nothing
; Returns:   Nothing
; Registers: r0l
; Remarks:

_wrdi_cmnd:
      bclr        #SCK,@XICOR       ; SCK low
      bclr        #CS,@XICOR        ; CS low
      mov.b       #WRDI,r0l         ; Setup Write Disable instruction
      bsr         _send_byte:16
      bclr        #SCK,@XICOR       ; SCK low
      bset        #CS,@XICOR        ; CS high
      rts

; Name:      _wrsr_cmnd
; Function:  Sends the command which enables writing to the WD0, WD1, BP0 and
;            BP1 bits of the Xicor X5043 EEPROM status register
; Calls:     _send_byte, _poll_write
; Expects:   Block Protect/Watchdog Timer bits in r0l
; Returns:   Nothing
; Registers: r0l, e0
; Remarks:

_wrsr_cmnd:
      mov.w       r0,e0             ; Temporarily preserve WDT/BP bits
      bclr        #SCK,@XICOR       ; SCK low
      bclr        #CS,@XICOR        ; CS low
      mov.b       #WRSR,r0l         ; Setup Write Status Register instruction
      bsr         _send_byte:16
      mov.w       e0,r0             ; Recover WDT/BP bits
      bsr         _send_byte:16
      bclr        #SCK,@XICOR       ; SCK low
      bset        #CS,@XICOR        ; CS high
      bsr         _poll_write:16
      rts

; Name:      _rdsr_cmnd
; Function:  Sends the command which reads the contents of the Xicor X5043
;            EEPROM status register
; Calls:     _send _byte, _recv_byte
; Expects:   Nothing
; Returns:   Status in r0l
; Registers: r0l
; Remarks:

_rdsr_cmnd:
      bclr        #SCK,@XICOR       ; SCK low
      bclr        #CS,@XICOR        ; CS low
      mov.b       #RDSR,r0l         ; Setup Read Status Register instruction
      bsr         _send_byte:16
      bsr         _recv_byte:16
      bclr        #SCK,@XICOR       ; SCK low
      bset        #CS,@XICOR        ; CS high
      rts


; Name:      _byte_write
; Function:  Writes a single byte to the Xicor X5043 EEPROM memory array
; Calls:     _send_byte, _poll_write
; Expects:   Byte to be sent in r0l, address in r1 (0x0000-0x01FF)
; Returns:   Nothing
; Registers: r0l, r1, e0
; Remarks:

_byte_write:
      bclr        #SCK,@XICOR       ; SCK low
      bclr        #CS,@XICOR        ; CS low
      mov.w       r0,e0             ; Temporarily preserve the byte
      mov.b       #WRITE,r0l        ; Setup Write instruction
      rotr.b      r1h               ; Rotate the address MSB into Carry
      bst         #ADDRBIT,r0l      ; Stuff it
      bsr         _send_byte:16
      mov.b       r1l,r0l           ; Fetch address LS byte
      bsr         _send_byte:16
      mov.w       e0,r0             ; Recover the byte
      bsr         _send_byte:16
      bclr        #SCK,@XICOR       ; SCK low
      bset        #CS,@XICOR        ; CS high
      bsr         _poll_write:16
      rts

; Name:      _byte_read
; Function:  Reads a single byte from the Xicor X5043 EEPROM memory array
; Calls:     _send_byte, _recv_byte
; Expects:   Serial EEPROM byte address in r1 (0x0000-0x01FF)
; Returns:   Byte read in r0l
; Registers: r0l, r1, e0
; Remarks:

_byte_read:
      bclr        #SCK,@XICOR       ; SCK low
      bclr        #CS,@XICOR        ; CS low
      mov.b       #READ,r0l         ; Setup Read instruction
      rotr.b      r1h               ; Rotate the address MSB into Carry
      bst         #ADDRBIT,r0l      ; Stuff it
      bsr         _send_byte:16
      mov.b       r1l,r0l           ; Fetch address LS byte
      bsr         _send_byte:16     ; ... and send it
      bsr         _recv_byte:16     ; Receive byte from EEPROM
      bclr        #SCK,@XICOR       ; SCK low
      bset        #CS,@XICOR        ; CS high
      rts

; Name:      _page_write
; Function:  Sends a full page (4 bytes) to the Xicor X5043 EEPROM
; Calls:     _send_byte, _poll_write
; Expects:   Serial EEPROM destination starting address in r1, pointer to
;            source bytes (4) in er2
; Returns:   Nothing
; Registers: r0l, r0h, r1, er2
; Remarks:

_page_write:
      bclr        #SCK,@XICOR       ; SCK low
      bclr        #CS,@XICOR        ; CS low
      mov.b       #WRITE,r0l        ; Setup Write instruction
      rotr.b      r1h               ; Rotate the address MSB into Carry
      bst         #ADDRBIT,r0l      ; Stuff it
      bsr         _send_byte:16
      mov.b       r1l,r0l           ; Fetch address LS byte
      and.b       #0b11111100,r0l   ; Mask drivel to page boundary
      bsr         _send_byte:16
      mov.b       #PAGESIZE,r3l
_page_wrloop:
      mov.b       @er2+,r0l         ; Fetch byte to send, point to next one
      bsr         _send_byte:16
      dec.b       r3l               ; Click off another sent byte
      bne         _page_wrloop
      bclr        #SCK,@XICOR       ; SCK low
      bset        #CS,@XICOR        ; CS high
      bsr         _poll_write:16
      rts

; Name:      _read_seq
; Function:  Reads a sequence of bytes from the Xicor X5043 Serial EEPROM
; Calls:     _send_byte, _recv_byte
; Expects:   EEPROM source starting address in r1 (0x0000-0x01FF), count of
;            bytes to read in e1, pointer to start of destination storage in er2
; Returns:   Byte array read in memory
; Registers: r0l, r0h, r1, e1, er2
; Remarks:

_read_seq:
      bclr        #SCK,@XICOR       ; SCK low
      bclr        #CS,@XICOR        ; CS low
      mov.b       #READ,r0l         ; Setup Read instruction
      rotr.b      r1h               ; Rotate the address MSB into Carry
      bst         #ADDRBIT,r0l      ; Stuff it
      bsr         _send_byte:16
      mov.b       r1l,r0l           ; Fetch address LS byte
      bsr         _send_byte:16
_read_seqloop:
      bsr         _recv_byte:16
      mov.b       r0l,@er2          ; Store EEPROM byte into memory
      inc.l       #1,er2            ; Point to next storage location
      dec.w       #1,e1             ; Click off another byte read
      bne         _read_seqloop
      bclr        #SCK,@XICOR       ; SCK low
      bset        #CS,@XICOR        ; CS high
      rts

; Name:      _write_seq
; Function:  Writes a sequence of bytes to the Xicor X5043 Serial EEPROM
; Calls:     _send_byte, _poll_write, _wren_cmnd
; Expects:   EEPROM destination starting address in r1 (0x0000-0x01FF), count of
;            bytes to write in e1, pointer to start of source storage in er2
; Returns:   Nothing
; Registers: r0l, r0h, r1, e1, er2, r3, e3
; Remarks:   Takes advantage of page write mode to minimize write times

_write_seq:
      bclr        #SCK,@XICOR       ; SCK low
      bclr        #CS,@XICOR        ; CS low
      mov.w       r1,e3             ; preserve EEPROM destination address
      mov.b       #WRITE,r0l        ; Setup Write instruction
      rotr.b      r1h               ; Rotate the address MSB into Carry
      bst         #ADDRBIT,r0l      ; Stuff it
      bsr         _send_byte:16
      mov.b       r1l,r0l           ; Fetch address LS byte
      bsr         _send_byte:16
_write_seqloop:
      mov.b       @er2+,r0l         ; Fetch byte to write & point to next one
      bsr         _send_byte:16
      dec.w       #1,e1             ; Click off another byte written
      inc.w       #1,e3             ; Point to next destination address
      mov.w       e3,r3             ; ... and scratchpad it
      and.b       #0b00000011,r3l   ; Keep destination address page bits
      beq         _wrseq_pagend     ; '00' means prior address was page end
      or.w        e1,e1             ; Is the byte count zero?
      bne         _write_seqloop    ;  No, continue with this page
      bclr        #SCK,@XICOR       ; SCK low
      bset        #CS,@XICOR        ; CS high
      bsr         _poll_write:16
      rts
_wrseq_pagend:
      bclr        #SCK,@XICOR       ; SCK low
      bset        #CS,@XICOR        ; CS high
      bsr         _poll_write:16    ; Is the write still in progress in EEPROM?
      or.w        e1,e1             ; Is the byte count zero?
      beq         _wrseq_done       ;   Yes, we're done
      bsr         _wren_cmnd:16     ;   No, more to do so re-enable writing
      mov.w       e3,r1             ; Recover next EEPROM address as expected
      bra         _write_seq        ; Back to the EEPROM
_wrseq_done:
      rts

; Name:      _send_byte
; Function:  Sends a byte to the Xicor X5043 EEPROM, serially shifting MSB
;            first
; Calls:     None
; Expects:   Byte to be sent in r0l
; Returns:   Nothing
; Registers: r0l, r0h
; Remarks:

_send_byte:
      mov.b       #8,r0h            ; Setup bit count
_send_loop:
      bclr        #SCK,@XICOR       ; SCK low
      rotxl.b     r0l               ; Slip next MSB into Carry
      bst         #SI,@XICOR        ; Copy Carry to I/O port bit
      bset        #SCK,@XICOR       ; SCK high
      dec.b       r0h               ; Click off a bit
      bne         _send_loop        ; Continue if not done
      bclr        #SI,@XICOR        ; SI low
      rts

; Name:      _recv_byte
; Function:  Receives a byte from the Xicor X5043 EEPROM, serially shifting MSB
;            first
; Calls:     None
; Expects:   Nothing
; Returns:   Received byte in r0l
; Registers: r0l, r0h
; Remarks:

_recv_byte:
      mov.b       #8,r0h            ; Setup bit count
_recv_loop:
      bset        #SCK,@XICOR       ; SCK high
      nop
      nop
      nop                           ; Guarantee limited clock rate
      bclr        #SCK,@XICOR       ; SCK low
      bld         #SO,@XICOR        ; Copy input port bit to Carry
      rotxl.b     r0l               ; Slip Carry into LSB
      dec.b       r0h               ; Click off a bit
      bne         _recv_loop        ; Continue if not done
      rts

; Name:      _poll_write
; Function:  Polls for the completion of the non-volatile write cycle by
;            examining the Write-In-Progress bit of the status register
; Calls:     _rdsr_cmnd
; Expects:   Nothing
; Returns:   Nothing
; Registers: r1l
; Remarks:
```

```
_poll_write:
      mov.b       #MAXPOLLS,r1l     ; Setup maximum number of poll attempts
_poll_loop:
      bsr         _rdsr_cmnd:16     ; Fetch the X5043 Status Register content
      btst        #WIPBIT,r0l       ; Is the Write-In-Progress bit zero?
      beq         _poll_loop1       ;  Yes, write  is complete
      dec.b       r1l               ;  No, click off another poll attempt
      bne         _poll_loop        ; If we haven't exceeded maximum polls ...
_poll_loop1:
      rts

; Name:       _wdog_reset
; Function:  Resets the watchdog timer
; Calls:      None
; Expects:   Nothing
; Returns:   Nothing
; Registers: None
; Remarks:

_wdog_reset:
      bclr        #CS,@XICOR
      nop
      nop
      nop
      bset        #CS,@XICOR
      rts

;  Data tables

      .align      0
_dataset0:
      .byte       DATABYTE0
      .byte       DATABYTE1
      .byte       DATABYTE2
      .byte       DATABYTE3
```