

Interfacing 2-wire Serial Memories to Hitachi H8/3000 Microcontrollers

by Applications Staff

This code demonstrates how the X24165/645 family of serial EEPROMs could be interfaced to the Hitachi H8/3000 microcontroller family when connected as shown in Figure 1. The interface requires two of the port pins

available on H8 family devices to implement the interface. This interface was tested with the X24645, however by addressing smaller arrays this code can be easily adapted for lower density memories.

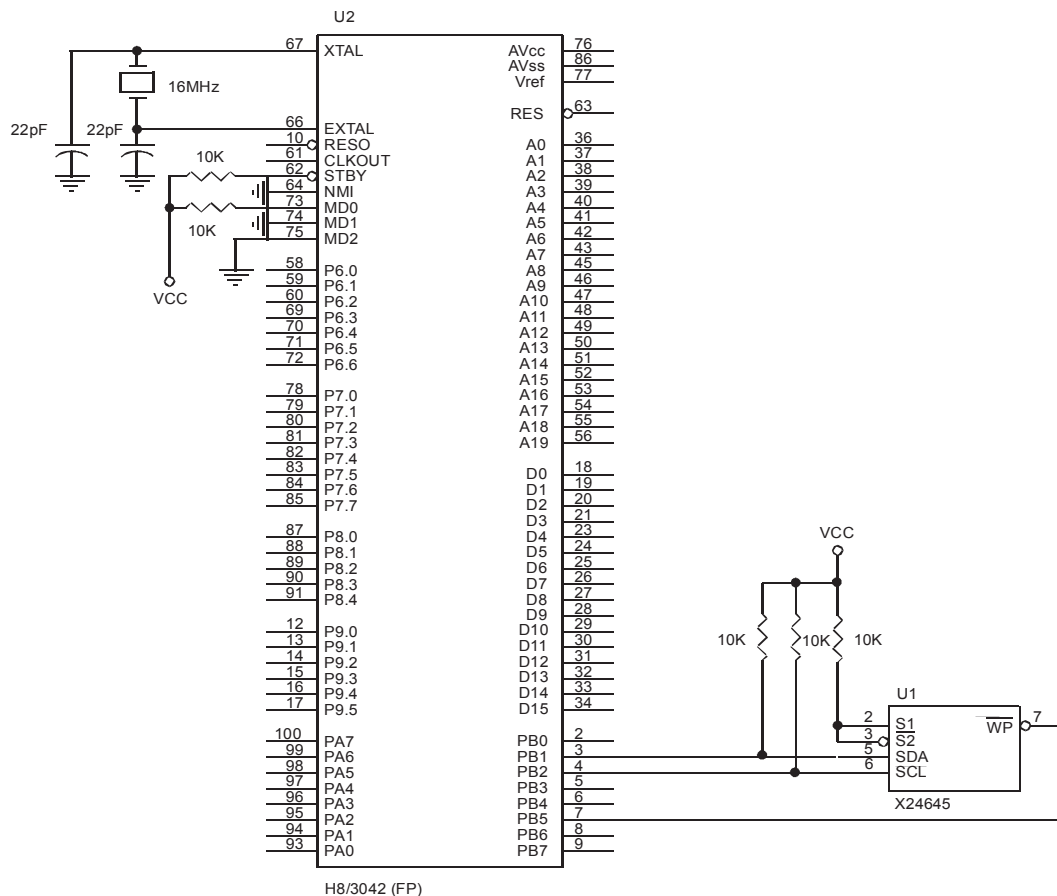


Figure 1. Typical hardware connection for interfacing a X24645 to the H8/3042 microcontroller



Application Note

AN81

```
.h8300h
.file "h8x24645.asm"

; Test code for interfacing Xicor X24645 Serial EEPROMS to the Hitachi H8/3042
; Outputs all read bytes, in ASCII hex form, serially

; Revision: 1.13 02/21/96

; Expects the H8/3042 hard-wired in Mode 1: 8-bit bus-width, Expanded address
; space (0x00000-0xFFFFF), Internal ROM disabled, Internal RAM enabled
; (0xFF710-0xFFFF0F), 16.0000 MHx xtal

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

; H8/3042 specific equates

    .equ  H8RAMTOP,    0xFFFF0F    ; Highest onboard RAM address (2048 bytes)
    .equ  H8RAMBOT,    0xFF710     ; Lowest onboard RAM address
    .equ  INTMASK,     0b11000000  ; CCR Interrupt Mask (I and UI bits)

; Xicor EEPROM device-specific equates

    .equ  SDA,         bit_1       ; Port bit for Serial Data (H8 in/out)
    .equ  SCL,         bit_2       ; Port bit for Serial CLock (H8 out)
    .equ  S1,          bit_3       ; Port bit for chip Select 1 (H8 out)
    .equ  S2,          bit_4       ; Port bit for chip Select 2 (H8 out)
    .equ  WP,          bit_5       ; Port bit for Write Protect (H8 out)

    .equ  WPR,         0x1FFF      ; Write Protect Register address
    .equ  WEL,         bit_1       ; Write Enable Latch (must be set to write
    ; EEPROM, resets to zero)
    .equ  RWEL,        bit_2       ; Register Write Enable Latch (must be set
    ; to change BP/WPEN bits, resets to zero)
    .equ  BP1,         bit_4       ; Block Protect MSB
    .equ  BP0,         bit_3       ;                               LSB
    .equ  WPEN,        bit_7       ; Write Protect ENable

    .equ  XICOR,       PBDR        ; H8/3042 data port assignment (PB)
    .equ  XICORDDR,    PBDDR       ; H8/3042 data direction port assignment
    .equ  XICORRD,     0b11111101  ; Data Direction Register setup H8<-X24645
    .equ  XICORWR,     0b11111111  ; Data Direction Register setup H8->X24645

    .equ  PAGESIZE,    32          ; Bytes per page
    .equ  NUMBYTES,    8192        ; Number of bytes in EEPROM
    .equ  NUMPAGES,    NUMBYTES/PAGESIZE ; Number of pages in EEPROM
    .equ  PAGEMASK,    PAGESIZE-1  ; Mask out non-page bits
    .equ  PAGEBNDRY,  ~ PAGEMASK   ; Mask out page bits (NOT PAGEMASK)
    .equ  ADDRMASK,    NUMBYTES-1  ; Mask out non-address bits
    .equ  MAXPOLL,     50          ; Maximum number of 'ack' poll attempts
    .equ  RDFLAG,      0b00000001  ; Read flag
    .equ  WRFLAG,      0b00000000  ; Write flag

; Main equates

    .equ  STACKTOP,    H8RAMTOP-4  ; Stack initializes here and builds down
```



Application Note

AN81

```
.equ ALLONES,      0b11111111 ; All bits set
.equ BIGWRITE,    170          ; Test number of bytes to write (>5 pages)
.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 RAMBFR3,     RAMBFR2+BIGWRITE ; Fourth read/write buffer
.equ DATABYTE0,   0xF0         ; Test byte 0
.equ DATABYTE1,   0x0F         ;           1
.equ DATABYTE2,   0x55         ;           2
.equ DATABYTE3,   0xAA         ;           3
.equ XADDRB,      0x1555       ; Test byte write/read address (loc 5461)
.equ TESTPAGE,    165          ; Test page number (0->NUMPAGES-1)
.equ XADDRBP,     NUMBYTES-10 ; Test block protect read/write address
.equ XADDRBW,     14*PAGESIZE+5 ; EEPROM BigWrite destination
.equ H8RAMLOC0,   H8RAMBOT+1047 ; Power up RAM test location 0
.equ H8RAMLOC1,   H8RAMBOT+1597 ;                               1
.equ PWRUPFLGS,   256*DATABYTE3+DATABYTE2 ; 0xAA55
.equ DEVADDR,     0b00000001 ; Device address (S1=1, S2=0)

; Start of code

; .org      0x000100

; h8vectbl.asm (H8/3042 Interrupt vector jump table 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
```



Application Note

AN81

```
.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
    mov.b      #0b00000000, r0l   ; Set Access State Control Register to
    mov.b      r0l, @ASTCR       ; 2-state access for all areas
    bsr        _init_xiport:16   ; Initialize the EEPROM I/O port
    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)
    cmp.w      r0, r3            ; Has RESET brought us here?
    beq        _test_recycle     ; Yes, complement all data this cycle
    xor.b      r6h, r6h         ; No, don't complement data this cycle
    bra        _test_start
```



Application Note

AN81

```
_test_recycle:
    not.b        r6h            ; To complement or not to complement, ...

;    Enable writing to the EEPROM

_test_start:
    bclr        #WP,@XICOR      ; Write Protect line low to enable EEPROM
    mov.b       #DEVADDR,r2l    ; Setup device address select (S1, S2)
    bsr         _write_enable:16 ; Enable EEPROM writes (sets Write Enable Latch)

;    Check device status

    mov.b       #DEVADDR,r2l    ; Setup device address select (S1, S2)
    bsr         _read_status:16 ; Fetch status from Write Protect Register

;    Write a single byte, then another

    mov.b       #DATABYTE3,r0l   ; Setup write test byte
    xor.b       r6h,r0l         ; Complement if alternate test cycles
    mov.w       #XADDRB,r1      ; Setup target test address
    mov.b       #DEVADDR,r2l    ; Setup device address select (S1, S2)
    bsr         _byte_write:16  ; Write byte

    mov.b       #DATABYTE2,r0l   ; Setup to write another test byte
    xor.b       r6h,r0l         ; Complement if alternate test cycles
    mov.w       #XADDRB+1,r1    ; Setup target test address (next location)
    mov.b       #DEVADDR,r2l    ; Setup device address select (S1, S2)
    bsr         _byte_write:16  ; Write byte

;    Read a single random byte

    mov.w       #XADDRB,r1      ; Setup the same address just written
    mov.b       #DEVADDR,r2l    ; Setup device address select (S1, S2)
    bsr         _byte_read_random:16 ; Read byte at this address

;    Read a single byte at current address

    mov.w       #XADDRB,r1      ; Setup the same address just written
    mov.b       #DEVADDR,r2l    ; Setup device address select (S1, S2)
    bsr         _byte_read_current:16 ; Read next byte

;    Generate a page of data in buffer by block moving page write data table
;    to RAM. This provides for complementing the data without placing the code
;    inside the page write subroutine.

    mov.l       #RAMBFR0,er3    ; Setup destination address
    mov.l       #_dataset0,er2  ; Setup source address
    mov.b       #PAGESIZE,r1l   ; Setup byte count to transfer

_xferloop:
    mov.b       @er2+,r0l       ; Get the byte and bump the source pointer
    xor.b       r6h,r0l         ; Complement if necessary
    mov.b       r0l,@er3        ; Stuff it into RAM
    inc.l       #1,er3          ; Bump the destination pointer to next
    dec.b       r1l             ; Click off another byte
    bne         _xferloop       ; Continue until count=0
```



Application Note

AN81

```
; Write a single page (from buffer 0)

mov.w    #TESTPAGE,r1      ; Setup EEPROM target page
mov.b    #DEVADDR,r21     ; Setup device address select (S1, S2)
mov.l    #RAMBFR0,er3     ; Point to source data
bsr      _page_write:16   ; ... and write the page

; Read a single page (into buffer 1)

mov.w    #TESTPAGE,r1     ; Again, setup EEPROM source page
mov.b    #DEVADDR,r21     ; Setup device address select (S1, S2)
mov.l    #RAMBFR1,er3     ; Point to destination buffer in onboard RAM
bsr      _page_read:16    ; ... and read the page (into different bfr)

; Setup a data buffer for a write sequence spanning many pages

xor.w    r1,r1            ; Initialize byte sequence/count
mov.l    #RAMBFR1,er2     ; ... and point to the destination
_bigwr_setup:
mov.b    r1l,r0l          ; Fetch byte to store
xor.b    r6h,r0l          ; Complement if required
mov.b    r0l,@er2         ; Store a byte in the RAM buffer
inc.l    #1,er2           ; Bump pointer to next RAM location
inc.w    #1,r1            ; Count bytes stored/generate next byte
cmp.w    #BIGWRITE,r1     ; Have we done 'em all?
blt      _bigwr_setup     ; No, continue

; Write a large number of contiguous bytes

mov.w    #BIGWRITE,r4     ; Setup byte count
mov.w    #XADDRBW,r1      ; Setup EEPROM starting address
mov.b    #DEVADDR,r21     ; Setup device address
mov.l    #RAMBFR1,er3     ; Setup source data pointer
bsr      _write_seq:16    ; ... and write 'em

; Read back the large number of bytes just written into another buffer

mov.w    #BIGWRITE,r4     ; Setup to read 'em back ... here's count
mov.w    #XADDRBW,r1      ; EEPROM source address
mov.b    #DEVADDR,r21     ; Setup device address
mov.l    #RAMBFR2,er3     ; Pointer to RAM destination
bsr      _read_seq:16    ; ... and read 'em

; Block Protect

bset     #WP,@XICOR       ; Write Protect line high to enable protect
mov.b    #0b10001000,r0l  ; Setup WPEN, block protect upper 25%
mov.b    #DEVADDR,r21     ; Setup device address
bsr      _write_protect:16

; Verify write protection state

mov.b    #DEVADDR,r21     ; Setup device address select (S1, S2)
bsr      _read_status:16  ; Fetch status from Write Protect Register
```



Application Note

AN81

```
;      End of test terminal loop

bclr      #WP,@XICOR      ; Write Protect line low to enable writes
mov.b     #0b00000000,r01 ; Setup to unprotect all blocks
mov.b     #DEVADDR,r21    ; Setup device address
bsr       _write_protect:16
bset      #WP,@XICOR      ; Write Protect line high to enable protect

mov.b     #DEVADDR,r21    ; Setup device address select (S1, S2)
bsr       _read_status:16 ; Fetch status from Write Protect Register

_endloop:
bra       _endloop
; EEPROM Interface Subroutines for XICOR X24645

; Name:      _init_xiport
; Function:   Initializes the H8 I/O port bit directions
; Calls:     None
; Expects:   Nothing
; Returns:   Nothing
; Registers: Perturbs r01
; Remarks    Must be called once initially to setup the I/O port

_init_xiport:
mov.b     #XICORWR,r01    ; Setup Port data directions
mov.b     r01,@XICORDDR   ; ... and load 'em
bset      #S1,@XICOR      ; Chip Select 1 high
bset      #S2,@XICOR      ;                2
bset      #SCL,@XICOR     ; Clock high
bset      #WP,@XICOR      ; Write Protect line high, disable writes
bset      #SDA,@XICOR     ; Data line high
rts

; Name:      _write_enable
; Function:   Sets WEL in WPR to enable EEPROM writes
; Calls:     _byte_write
; Expects:   Device select in r21
; Returns:   Carry clear if okay, set otherwise
; Registers: Alters r01, r1, r21
; Remarks    Must be called after reset (power up), before writing EEPROM

_write_enable:
mov.w     #WPR,r1         ; Setup Write Protect Register address
mov.b     #0b00000010,r01 ; Setup to write the WEL bit
bsr       _byte_write:16 ; ... and do so
rts

; Name:      _read_status
; Function:   Reads the content of the Write Protect Register (WPR)
; Calls:     _byte_read_random
; Expects:   Device select in r21
; Returns:   Carry reset if okay, set otherwise
; Registers: Alters r01, r1, r21
; Remarks
```



Application Note

AN81

```
_read_status:
    mov.w    #WPR,r1          ; Setup Write Protect Register address
    bsr     _byte_read_random:16 ; Read the status of WPR
    rts

; Name:      _write_protect
; Function:  Enables or disables software write protect (if WP high)
; Calls:    _byte_write
; Expects:  Device select in r2l, write protect state in r0l:
;           bit 7  WPEN set to enable write protect, reset to disable
;           bits 4:3 BP1:0 Block protect identifier:
;           none      00
;           top quarter 01 0x1800-0x1FFF
;           top half   10 0x1000-0x1FFF
;           all       11 0x0000-0x1FFF
; Returns:  Carry reset if okay, set otherwise
; Registers: Alters er0, r1, e2, r2h
; Remarks   On entry, r0l should contain the WPEN and BP bits in their correct
;           locations ... they are not verified.

;           Note that the final write to WPR requires 10 ms, however the normal
;           timeout for EEPROM write cycles is 5 ms. To compensate we repeat
;           the polling for another 5 ms before we decide that the write was
;           unsuccessful (carry set to one).

_write_protect:
    mov.w    r2,e2          ; Preserve the device select
    mov.w    r0,e0          ; ... and the protect code
    bsr     _write_enable:16
    bcs     _write_prnack   ; If acknowledge failed
    mov.b    #0b00000110,r0l ; Setup second byte in the sequence w/RWEL
    mov.w    #WPR,r1        ; ... and the WPR address
    mov.w    e2,r2         ; Recover device select
    bsr     _byte_write:16  ; ... and write it!
    bcs     _write_prnack   ; If acknowledge failed
    mov.w    e0,r0         ; Recover protect code
    or.b     #0b00000110,r0l ; Ensure that WEL and RWEL are set
    mov.w    #WPR,r1        ; Again, the Write Protect Register address
    mov.w    e2,r2         ; Recover device select
    bsr     _byte_write:16  ; Write it (5ms polling)!
    ; Enter the following extended polling after
    ; the stop condition from _byte_write
    mov.w    #WPR,r1        ; Again, the Write Protect Register address
    mov.b    #MAXPOLLs,r0h  ; Setup maximum poll attempts again (5ms)
    bsr     _delay:16

_long_poll:
    mov.w    e2,r2          ; ... and device select
    mov.b    #WRFLAG,r2h    ; Setup write command
    bsr     _send_slave_addr:16
    bcc     _longpoll_okay  ; EEPROM acknowledge (low) is write done
    bsr     _stop:16        ; Send stop sequence
    bsr     _delay:16
    dec.b    r0h            ; Click off the poll count ... reached zero?
    bne     _long_poll      ; No, continue polling
```



Application Note

AN81

```
    orc          #0b00000001, ccr    ; Yes, timed out ... set carry (nack)
    rts                                     ; SCL high SDA high
_longpoll_okay:                                     ; Successful, carry=0
    bsr          _stop:16
_write_prnack:
    rts                                     ; Exits with SCL high and SDA high

; Name:          _byte_write
; Function:      Writes a single byte to the Xicor EEPROM memory array
; Calls:         _send_byte, _send_slave_addr, _send_byte_addr, _write_poll, _stop
; Expects:      Byte to be sent in r0l, address in r1, device select in r2l
; Returns:      Acknowledge in the Carry bit
; Registers:    r0, r1, r2, e1, e2
; Remarks:

_byte_write:
    mov.b       r0l, r0h                ; Preserve the byte to be written
    mov.w       r1, e1                  ; ... and its address
    mov.w       r2, e2                  ; ... and the select
    mov.b       #WRFLAG, r2h           ; Setup Read/Write flag (bit 0=0, write)
    bsr        _send_slave_addr:16
    bcs        _byte_wrnack            ; EEPROM did not acknowledge
    bsr        _send_byte_addr:16
    bcs        _byte_wrnack            ; Failed acknowledge
    mov.b       r0h, r0l                ; Recover byte to write to EEPROM
    bsr        _send_byte:16           ; And transmit it
    bcs        _byte_wrnack            ; Negative acknowledge
    bsr        _write_poll:16          ; Poll for write completion
    rts                                     ; Exits with SCL high and SDA high
_byte_wrnack:
    bsr        _stop:16                ; Output the stop sequence
    rts                                     ; Exits with SCL high and SDA high

; Name:          _byte_read_random
; Function:      Reads a single byte from the Xicor EEPROM memory array
; Calls:         _send_byte, _recv_byte _send_word
; Expects:      Serial EEPROM byte address in r1, device select in r2l
; Returns:      Byte read in r0l, carry set if any problem, reset means okay
; Registers:    Alters e1, e2, r2h, r5l
; Remarks:

_byte_read_random:
    mov.w       r1, e1                  ; Preserve the EEPROM address
    mov.w       r2, e2                  ; ... and the select bits
    mov.b       #WRFLAG, r2h           ; Setup Read/Write flag (bit 0=0, write)
    bsr        _send_slave_addr:16     ; Dummy write
    bcs        _byte_rdnack            ; If the acknowledge failed
    bsr        _send_byte_addr:16
    bcs        _byte_rdnack            ; EEPROM failed acknowledge
    mov.w       e1, r1                  ; Recover EEPROM address
    mov.w       e2, r2                  ; ... and select code
    mov.b       #RDFLAG, r2h           ; Setup Read/Write flag (bit 0=1, read)
    bsr        _send_slave_addr:16     ; The actual read ...
    bcs        _byte_rdnack            ; Acknowledge fail
```



Application Note

AN81

```
    mov.b    #0b10000000,r5l    ; Setup negative acknowledge
    bsr      _recv_byte:16      ; Read byte
    andc     #0b11111110,ccr    ; Clear the carry
_byte_rdnack:
    bsr      _stop:16           ; Output stop sequence
    rts                                     ; Exits with SCL high and SDA high

; Name:      _byte_read_current
; Function:  Reads 'the next' byte from the Xicor EEPROM memory array
; Calls:     _send_slave_addr, _recv_byte, _stop
; Expects:   Serial EEPROM byte address in r1, device select in r2l
; Returns:   Byte read in r0l, carry=0 if byte was read, =1 if not
; Registers: r2h, r5l
; Remarks:

_byte_read_current:
    mov.b    #RDFLAG,r2h       ; Setup Read/Write flag (bit 0=1, read)
    bsr      _send_slave_addr:16 ; Byte address not necessary
    bcs      _byte_rdcnack     ; Acknowledge failed
    mov.b    #0b10000000,r5l    ; Setup nack, only reading one byte
    bsr      _recv_byte:16      ; Read byte
    andc     #0b11111110,ccr    ; Indicate we read the byte successfully
_byte_rdcnack:
    bsr      _stop:16           ; Output the stop sequence
    rts                                     ; Exits with SCL high and SDA high

; Name:      _page_write
; Function:  Sends a full page to the Xicor EEPROM
; Calls:     _send_byte, _write_poll, _send_slave_addr, _send_byte_addr, _stop
; Expects:   Serial EEPROM destination page number in r1, pointer to byte
;            source in er3, device select in r2l
; Returns:   Carry set if unsuccessful, cleared otherwise
; Registers: Alters r0l, er1, r2h, er3, r4
; Remarks:

_page_write:
    mov.w    #PAGESIZE,r4      ; Setup the number of bytes per page
    mulx.w   r4,er1            ; Calculate EEPROM byte address
    mov.w    r1,e1             ; Preserve the address
    mov.w    r2,e2             ; ... and the select
    mov.b    #WRFLAG,r2h      ; Setup Read/Write flag (bit 0=0, write)
    bsr      _send_slave_addr:16
    bcs      _page_wrnack     ; Acknowledge fail
    bsr      _send_byte_addr:16
    bcs      _page_wrnack     ; Acknowledge fail
_page_wrloop:
    mov.b    @er3+,r0l        ; Fetch next byte, point to next one
    bsr      _send_byte:16     ; Output it
    bcs      _page_wrnack     ; If acknowledge fails
    dec.b    r4l              ; Click off another byte, done?
    bne     _page_wrloop      ; No, keep on
    bsr      _stop:16         ; Yes, output the stop sequence
    bsr      _write_poll:16    ; Wait until writing is complete ...
_page_wrnack:
```



Application Note

AN81

```
    bsr        _stop:16          ; Output the stop sequence
    rts                               ; Exits with SCL high and SDA high

; Name:      _page_read
; Function:  Reads a full page from the Xicor EEPROM
; Calls:     _send_byte, _poll_write _send_word
; Expects:   EEPROM page number in r1, pointer to destination bytes in er3,
;            select code in r2l
; Returns:   Bytes read into destination and carry=0 if successful, carry
;            set otherwise
; Registers: Alters r0l, er1, e2, r2h, er3, r4, r5l
; Remarks:
_page_read:
    mov.w     #PAGESIZE,r4      ; Setup the number of bytes per page
    mulx.w   r4,er1             ; Calculate EEPROM byte address
    mov.w    r1,e1              ; Preserve the address
    mov.w    r2,e2              ; ... and the select
    mov.b    #WRFLAG,r2h       ; Setup Read/Write flag (bit 0=0, write)
    bsr     _send_slave_addr:16 ; Dummy write to setup address in EEPROM
    bcs     _page_rdnack       ; If acknowledge fails ...
    bsr     _send_byte_addr:16
    bcs     _page_rdnack
    mov.w   e1,r1              ; Recover EEPROM address
    mov.w   e2,r2              ; ... and device select
    mov.b   #RDFLAG,r2h       ; Setup Read/Write flag (bit 0=1, read)
    bsr     _send_slave_addr:16 ; The actual read ...
    bcs     _page_rdnack       ; Acknowledge fail
    dec.b   r4l                ; So that the last byte read is count=0
_page_rdnack:
    mov.b   #0b00000000,r5l    ; Setup acknowledge
    bsr     _recv_byte:16      ; Read byte
    mov.b   r0l,@er3           ; Store it
    inc.l   #1,er3             ; Point to next storage location
    dec.b   r4l                ; Another one read, last one next?
    bne     _page_rdnack      ; No, continue reading
    mov.b   #0b10000000,r5l    ; Yes, setup negative acknowledge
    bsr     _recv_byte:16      ; Read byte
    mov.b   r0l,@er3           ; Store it
    andc    #0b11111110,ccr    ; Indicate success
_page_rdnack:
    bsr     _stop:16          ; We're done ... stop
    rts                               ; Exits with SCL high and SDA high

; Name:      _read_seq
; Function:  Reads a sequence of bytes from the Xicor EEPROM
; Calls:     _send_byte, _recv_byte _send_slave_addr, _send_byte_addr
; Expects:   EEPROM source starting address in r1, count of bytes to read in r4,
;            pointer to start of destination storage in er3, select in r2l
; Returns:   Byte array in memory and carry reset if okay, carry set if not
; Registers: Alters r0l, e1, e2, r2h, er3, r4, r5l
; Remarks:   Byte count (r4) is not limit verified (i.e. up to 65536 bytes may
;            be read, cycling through the EEPROM addresses)
_read_seq:
    mov.w   r1,e1              ; Preserve the destination address
```



Application Note

AN81

```
mov.w    r2,e2          ; ... and the select
mov.b    #WRFLAG,r2h    ; Setup Read/Write flag (bit 0=0, write)
bsr      _send_slave_addr:16 ; Dummy write
bcs      _read_sqnack    ; Acknowledge fail
bsr      _send_byte_addr:16
bcs      _read_sqnack    ; Failed acknowledge
mov.w    e1,r1          ; Recover destination address
mov.w    e2,r2          ; ... and device select code
mov.b    #RDFLAG,r2h    ; Setup Read/Write flag (bit 0=1, read)
bsr      _send_slave_addr:16 ; The actual read ...
bcs      _read_sqnack    ; Acknowledge fail
dec.w    #1,r4          ; So that last byte read is count=0
_read_sqloop:
mov.b    #0b00000000,r5l ; Setup acknowledge
bsr      _recv_byte:16   ; Read byte
mov.b    r0l,@er3        ; ... and store it
inc.l    #1,er3          ; Point to the next location
dec.w    #1,r4          ; ... and count it down, is this one last?
bne      _read_sqloop    ; No, continue reading
mov.b    #0b10000000,r5l ; Yes, setup negative acknowledge
bsr      _recv_byte:16   ; Read byte
mov.b    r0l,@er3        ; ... and store it
andc     #0b11111110,ccr ; Indicate success ... carry=0
_read_sqnack:
bsr      _stop:16        ; Stop!
rts      ; Exits with SCL high and SDA high

; Name:      _write_seq
; Function:  Writes a sequence of bytes to the Xicor EEPROM
; Calls:     _send_byte, _send_slave_addr, _write_poll, _send_byte_addr
; Expects:   EEPROM destination starting address in r1, count of bytes to write
;           in r4, pointer to start of source storage in er3, select in r2l
; Returns:   Carry set if successful, cleared otherwise
; Registers: Alters r0l, er1, e2, r2h, er3, er4
; Remarks:   Takes advantage of page write mode to minimize write times

_write_seq:
mov.w    r1,e1          ; Preserve the destination address
mov.w    r2,e2          ; ... and the select
_wr_nextpg:
mov.b    #WRFLAG,r2h    ; Setup Read/Write flag (bit 0=0, write)
bsr      _send_slave_addr:16 ; Start of the write sequence
bcs      _write_sqdone    ; Acknowledge fail
bsr      _send_byte_addr:16 ; Next part of the address
bcs      _write_sqdone    ; Acknowledge fail
_wr_sqloop:
mov.b    @er3+,r0l      ; Fetch byte to write and point to next one
bsr      _send_byte:16   ; Send it
bcs      _write_sqdone    ; Acknowledge fail
inc.w    #1,e1          ; Advance EEPROM address
dec.w    #1,r4          ; Click off another byte written
mov.w    e1,e4          ; Get the current EEPROM address
and.w    #PAGEMASK,e4   ; Mask out non-page bits ... page end?
beq      _page_end      ; Yes, do the polling thing ...
or.w    r4,r4          ; No, has the count reached zero?
```



Application Note

AN81

```
    bne        _wr_sqloop        ;    No, more bytes to write
_page_end:    ;    Yes, poll for write completion ...
    bsr        _write_poll:16
    bcs        _write_sqdone     ;    Polling timed out without an acknowledge
    or.w       r4,r4             ;    Are there more bytes to write?
    beq        _write_sqdone     ;    No, we're done ...
    mov.w      e1,r1             ;    Yes, recover EEPROM address ...
    mov.w      e2,r2             ;    ... and the device select
    bra        _wr_nextpg       ;    Continue writing the next page
_write_sqdone:
    bsr        _stop:16         ;    Done, send the stop sequence
    rts                          ;    Exits with SCL high and SDA high
```

```
Name:        _send_slave_addr
; Function:   Writes the first byte 'slave' address sequence to the Xicor EEPROM
; Calls:     _send_byte, -delay
; Expects:   EEPROM destination starting address in r1, device select address
;           in r2l, Read/Write flag in r2h (bit 0=0 for write, 1 for read)
; Returns:   Acknowledge in carry (1=fail, 0=okay)
; Registers: r0l, r0h, r1, r2, r3l
; Remarks:
```

```
_send_slave_addr:
    mov.b      #XICORWR,r0l      ;    Setup SDA for output (H8->X24645)
    mov.b      r0l,@XICORDDR     ;    ... and load it
    bset       #SDA,@XICOR      ;    Guarantee that data (SDA) is high
    bsr        _delay:16        ;    Guarantee required time
    bset       #SCL,@XICOR      ;    Guarantee that clock (SCL) is high
    and.b      #0b00000011,r2l   ;    Eliminate that which is not device address
    rotr.b     r2l              ;    Slip device address to upper two bits
    rotr.b     r2l
    and.b      #0b00011111,r1h   ;    Mask away non-upper byte address bits
    shll.b     r1h              ;    Nudge byte address, make room for r/w bit
    mov.b      r2h,r0l          ;    0x00000000=write, 0x00000001=read
    and.b      #0b00000001,r0l   ;    Mask out any driv1 from caller
    or.b       r2l,r0l          ;    Stuff device address ...
    or.b       r1h,r0l          ;    ... and byte upper address, r/w bit
    bclr       #SDA,@XICOR      ;    Drop SDA while SCL high (start condition)
    bsr        _send_byte       ;    Go send the byte ...
    rts                          ;    Exits with SCL low and SDA low
```

```
; Name:      _send_byte_addr
; Function:   Writes the second 'byte' address to the Xicor EEPROM
; Calls:     _send_byte
; Expects:   EEPROM destination starting address in r1
; Returns:   Acknowledge in carry (1=fail, 0=okay)
; Registers: r0l
; Remarks:   If required, must be called immediately after '_send_slave_addr'
```

```
_send_byte_addr:
    mov.b      r1l,r0l          ;    Fetch destination address LS byte
    bsr        _send_byte       ;    And send it ...
    rts                          ;    Exits with SCL low and SDA low
```

```
; Name:      _send_byte
```



Application Note

AN81

```
; Function: Sends a byte to the Xicor EEPROM, serially shifting MSB first
; Calls:    _delay
; Expects: Byte to be sent in r0l
; Returns: Acknowledge bit in carry (1=fail, 0=okay)
; Registers: r0l, r0h, r2h
; Remarks:
```

```
_send_byte:
    mov.b    #8,r2h            ; Setup bit count
_send_loop:
    bsr     _delay:16         ; Delay set for 16MHz, 2-state (no waits)
    bclr    #SCL,@XICOR      ; Clock (SCL) low
    shll.b  r0l              ; Slip MS bit into carry
    bst     #SDA,@XICOR      ; Store carry state to port pin SDA
    bsr     _delay:16
    bset    #SCL,@XICOR      ; Clock (SCL) high
    dec.b   r2h              ; Count another bit, are we done?
    bne     _send_loop       ; No, continue ...
    bsr     _delay:16
    bclr    #SCL,@XICOR      ; Yes, clock (SCL) low
    mov.b   #XICORRD,r0l     ; Setup SDA for input (H8<-X24645)
    mov.b   r0l,@XICORDDR    ; SDA input
    bsr     _delay:16
    bset    #SCL,@XICOR      ; Clock (SCL) high
    bsr     _delay:16
    bld     #SDA,@XICOR      ; Copy input pin (SDA) into carry ... ack
    bclr    #SCL,@XICOR      ; Clock (SCL) low
    mov.b   #XICORWR,r0l     ; Setup SDA for output (H8->X24645)
    bclr    #SDA,@XICOR      ; SDA low
    mov.b   r0l,@XICORDDR    ; SDA output
    rts                                     ; Exits with SCL low and SDA low
```

```
; Name:    _recv_byte
; Function: Receives a byte from the Xicor EEPROM, serially shifting MSB first
; Calls:    _delay
; Expects: Acknowledge in r5l bit 7 (1=acknowledge, 0=negative acknowledge)
; Returns: Received byte in r0l
; Registers: Alters r0l, r0h, r2h, r5l
; Remarks: Clock rate limited for H8/3042 16MHz xtal and 2-state access
```

```
_recv_byte:
    mov.b    #8,r2h            ; Setup bit count
    mov.b    #XICORRD,r0l     ; Setup SDA for input (H8<-X24645)
    mov.b    r0l,@XICORDDR    ; SDA input
_recv_loop:
    bsr     _delay:16         ; Delay set for 16MHz, 2-state (no waits)
    bclr    #SCL,@XICOR      ; Clock (SCL) low
    bsr     _delay:16
    bld     #SDA,@XICOR      ; Load SDA port pin into carry
    rotxl.b r0l              ; Carry into LSB, MSB into carry
    bset    #SCL,@XICOR      ; Clock (SCL) high
    dec.b   r2h              ; Count another bit, are we done?
    bne     _recv_loop       ; No, continue ...
    bsr     _delay:16
    bclr    #SCL,@XICOR      ; Yes, clock (SCL) low
```



Application Note

AN81

```
mov.b    #XICORWR,r0h    ; Setup SDA for output (H8->X24645)
mov.b    r0h,@XICORDDR   ; SDA output
shll.b   r5l             ; Move acknowledge bit into carry
bst      #SDA,@XICOR     ; Carry into SDA ... H8 acknowledge
bsr      _delay:16
bset     #SCL,@XICOR     ; Clock (SCL) high
bsr      _delay:16
bclr     #SCL,@XICOR     ; Clock (SCL) low
bclr     #SDA,@XICOR     ; Remove H8 acknowledge (SDA low)
rts
```

```
; Name:      _write_poll
; Function:   Monitors EEPROM internal write completion status
; Calls:      _stop, _delay, _send_slave_addr
; Expects:    EEPROM address in e1, select in e2
; Returns:    Carry set if polling times out, cleared if normal completion
; Registers:  Alters r1, r2, r0h
; Remarks:
```

```
_write_poll:
    bsr      _stop:16      ; Polling starts from a stop condition
    mov.b    #MAXPOLLS,r0h ; Setup maximum number of poll attempts
    bsr      _delay:16
_poll_loop:
    mov.w    e1,r1        ; Recover EEPROM address
    mov.w    e2,r2        ; ... and device select
    mov.b    #WRFLAG,r2h  ; Setup write command
    bsr      _send_slave_addr:16
    bcc      _write_okay   ; EEPROM acknowledge (low) is write done
    bsr      _stop:16      ; Send stop sequence
    bsr      _delay:16
    dec.b    r0h          ; Click off the poll count ... reached zero?
    bne      _poll_loop    ; No, continue polling
    orc      #0b00000001,ccr ; Yes, timed out ... set carry
    rts
_write_okay:
    bsr      _stop:16
    rts                  ; SCL high SDA high
```

```
; Name:      _stop
; Function:   Generates the EEPROM stop sequence
; Calls:      _delay
; Expects:    Nothing
; Returns:    Nothing
; Registers:  None
; Remarks:
```

```
_stop:
    bclr     #SDA,@XICOR   ; Stop condition requires SDA low
    bsr      _delay:16     ; Wait ...
    bset     #SCL,@XICOR   ; Clock (SCL) high
    push.l   er6           ; Non-destructive 16-state delay (1.0 usec)
    pop.l    er6
    bsr      _delay:16     ; Wait ...
    bset     #SDA,@XICOR   ; Raise SDA while SCL high (stop condition)
```



Application Note

AN81

```
    rts                                ; Exits with SCL high and SDA high

; Name:                                _delay
; Function:                             Delays execution so that a 16MHz 2-state H8/3042 won't exceed
;                                         EEPROM operational timing
;
; Calls:                                 Nothing
; Expects:                               Nothing
; Returns:                               In a little while (actually 28 states or 1.75 usec at 16MHz)
; Registers:                             None
; Remarks:                               This delay should be reduced to optimize performance at
;                                         other clock rates/access state settings
;

_delay:                                ; 8 states (0.5 usec @ 16MHz)
    nop                                ; 2 states (0.125 usec)
    nop                                ; 2 states (0.125 usec)
    nop                                ; 2 states (0.125 usec)
    nop                                ; 2 states (0.125 usec)
    nop                                ; 2 states (0.125 usec)
    nop                                ; 2 states (0.125 usec)
    rts                                ; 8 states (0.5 usec)

; Data tables

_dataset0:
    .byte    0xAA,0x55,0xAA,0x55,0xAA,0x55,0xAA,0x55
    .byte    0x01,0x02,0x04,0x08,0x10,0x20,0x40,0x80
    .byte    0xAA,0x55,0xAA,0x55,0xAA,0x55,0xAA,0x55
    .byte    0xFE,0xFD,0xFB,0xF7,0xEF,0xDF,0xBF,0x7F
```



LittleDiode supplies new, hard to find or obsolete electronic components and semiconductors all over the world.

With over two million different components listed you are sure to find the part you need.

Feel free to visit us today at our online store:

LittleDiode.com

Looking forward to providing you with the best possible service.