Hitachi America, Ltd.

Application Note

H8/300

Carol Jacobson

Using Serial EEPROM with the H8/300

INTRODUCTION

Serial EEPROM's provide a low cost, easy to use source of re-programmable, non-volatile memory with minimum impact on system power requirements. Potential applications for serial EEPROMs are infinite but low power, low voltage, commercial and industrial designs consume the greatest volume. SEEPROMs are available from several suppliers, most notably ATMEL and National Semiconductor. Both manufacturers sell 1K (128 bytes) to 4K (512 bytes) versions packaged in 8-pin DIP or SO ICs. ATMEL offers low voltage options operating down to 1.8V with typical operating supplies of 0.2 to 0.4 mA.

SERIAL EEPROM OVERVIEW

Serial EEPROMs support 2, 3 and 4-wire interface protocols that vary primarily in the ease of implementation and maximum bit clock rate. The 2-wire models use only a serial data line (SDA) and clock (SCK). This transfer format follows the IIC bus protocol and therefore has a maximum bit clock rate of 100 KHz.

The 4-wire handshake requires signal lines: data in (DI), data out (DO), ready/busy (RDY/BUSY*) and the serial clock (CLK). A 5th line, chip select (CS), enables the EEPROM. The 4-wire handshake offers the simplest interface by sending the ready/busy signal back to the controller at the end of a memory write or erase cycle. This line can be polled to begin the next cycle. Like the 3-wire format, current 4-wire devices support clock speeds up to 1 MHz.

This application note will discuss the 3-wire (MICROWIRE™) protocol SEEPROM from ATMEL and provide an example interface to the H8/330 Evaluation Board. With minor modifications, this example could also apply to a 4-wire interface. The information furnished herein is intended for example only. This application note should not be used as a substitute for the SEEPROM manufacturer's product information.

3-WIRE FORMAT

The 3-wire format requires 4 signal lines summarized in table I below:

Table I

Pin Name	Abbr.	Function
Chip Select	CS	SEEPROM enable
Data In	DI	serial data input
Data Out	DO	serial data output
Serial Clock	SK	serial bit clock

The controller initiates a data transfer by sending a synchronous instruction frame consisting of a start bit, the appropriate command code (op code), the SEEPROM address and/or data as required. There are seven operations:

READ- Reads data stored at a specified address
WRITE-Writes data to a specified address
ERASE-Erases data at a specified address
WRAL- Writes all locations with a specified data value
ERAL- Erases all locations
EWEN- Enables erase and write operations
EWDS- Disables erase and write operations

The instruction set and frame format for each operation for the AT93C66 are shown below in Table II.

Table II

Instruction	SB	Opcode	Address		Data	
			x 8	x 16	x 8	x 16
READ	1	10	A8 -- A0	A7 -- A0		
EWEN	1	00	11xxxxxxx	11xxxxxxx		
ERASE	1	11	A8 -- A0	A7 -- A0		
WRITE	1	01	A8 -- A0	A7 -- A0	D7 -- D0	D15 -- D0
ERAL	1	00	10xxxxxxx	10xxxxxxx		
WRAL	1	00	01xxxxxxx	01xxxxxxx	D7 -- D0	D15 -- D0
EWDS	1	00	00xxxxxxx	00xxxxxxx		

OPERATION

THE READ CYCLE

To initiate a read cycle a controller transmits a start bit and the READ command, b'10, concatenated with the SEEPROM address. For example, to read data at address h'101 the MCU sends a 12-bit serial stream:

MSB LSB

1	1	0	1	0	0	0	0	0	0	0	1
s	oc	oc									

A 512 byte (4K) serial EEPROM requires a 9-bit address in the command frame immediately following the op code. Sending the byte stream containing the start bit and op code (h'06) immediately followed by a 2-byte, word address (h'0101), results in an erroneous address request. For example, if R0l contains the READ command and R1 contains the 16-bit address h'0101

MSB R0L LSB

0	0	0	0	0	1	1	0
					s	oc	oc

MSB R1H LSB

0	0	0	0	0	0	0	1
							A8

MSB R1L LSB

0	0	0	0	0	0	0	1
							A0

Sequentially loading the Transmit Data Register with R0L, R1H, R1L results in the following 24-bit stream:

MSB

0	0	0	0	0	1	1	0	0	0	0	0
					s	oc	oc	A8			

LSB

0	0	0	1	0	0	0	0	0	0	0	1
			A1	A0							

The SEEPROM reads the address as h'002, ignoring the last 7 bits of the stream. To avoid this, we concatenate the 9th address bit, A8, with the start bit and op-code held in R0L and send two bytes, R0L and R1L.

MSB R0L LSB

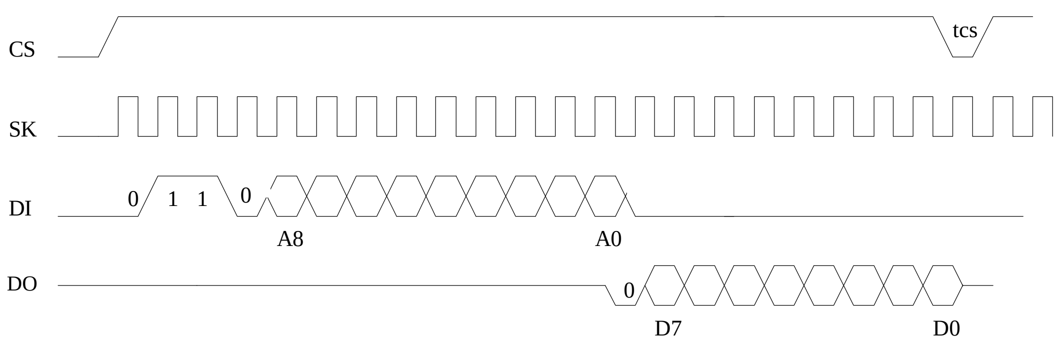
0	0	0	0	1	1	0	1
				s		oc	A
					oc		8

MSB R1L LSB

0	0	0	0	0	0	0	1
							A0

SEEPROMs less than 4K have 8-bit addresses, A7 - A0, that fit into a single register. The extra step of concatenating A8 with the command byte isn't required when working with < 4K.

Following the command frame the MCU should begin to read data from the DO line. The SEEPROM changes data on the rising edge of the serial clock. On the rising edge of the clock at which it detects A0, the SEEPROM responds to the read request by sending a low 'dummy bit' followed by the 8-bit data stored at address h'101. The timing diagram for a READ cycle is shown below in figure 1.



AT93Cx6 Read Cycle

Figure 1

THE WRITE CYCLE

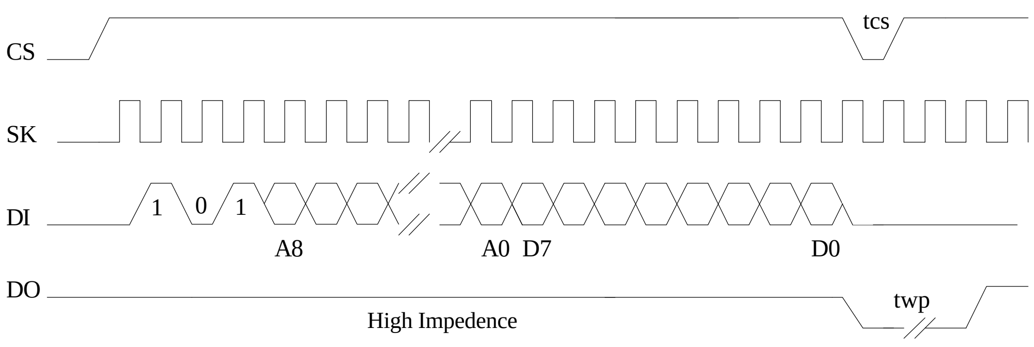
A WRITE cycle begins in the same manner as a READ cycle but includes data as a third byte in the command frame. The first data bit, D7, must immediately follow the last address bit, A0. A typical 3 byte WRITE command frame to a 4K SEEPROM should look like this:

MSB			1st byte				LSB
0	0	0	0	1	0	1	1
				s	oc	oc	A8

MSB			2nd byte				LSB
0	0	0	0	0	0	0	1
							A0

MSB			3rd byte				LSB
1	0	1	0	1	0	1	0
D7							D0

This command frame sends a start bit, the WRITE op code, address h'101 and 8-bit data, h'AA. The result writes the value h'AA at SEEPROM address h'101. According to manufacturers' specifications, a write cycle can take up to 10 ms to complete after the SEEPROM receives D0. If the MCU toggles CS after transmitting D0, the SEEPROM responds by pulling the DO line low for the duration of the write cycle. DO thereby provides an end of cycle signal. Timing of CS low following A0 is not critical as long as the line toggles after the last data bit transfers out of the TSR. SEEPROM WRITE cycle timing is shown in figure 2.



AT93Cx6 Write Cycle

Figure 2

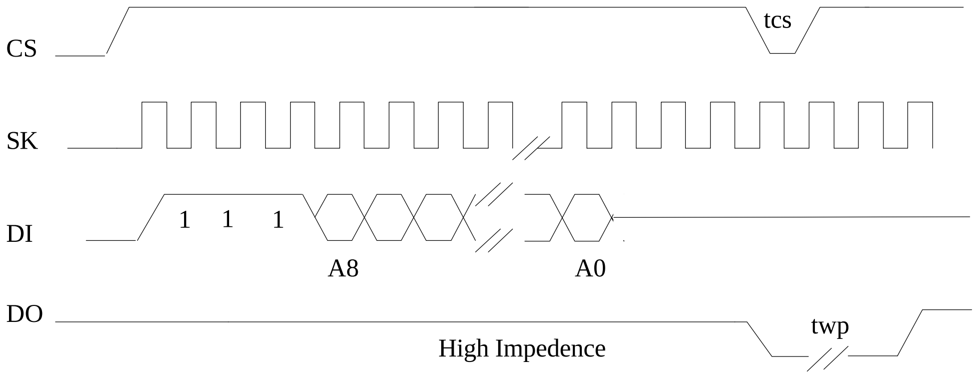
THE ERASE CYCLE

The single address ERASE cycle operates much like a write cycle but does not include the third data byte. The controller sends a 2-byte command frame consisting of a start bit, op code and the address to be erased. A typical frame is shown below:

MSB			1st byte				LSB
0	0	0	0	1	1	1	1
				s	oc	oc	A8

MSB			2nd byte				LSB
0	0	0	0	0	0	0	1
							A0

This command sequence erases data stored at SEEPROM address h'101 by programming the address to all 1's. The ERASE cycle can take up to 10 ms to complete. Therefore, as with the WRITE cycle, if the controller toggles the CS line after the A0 clock, the EEPROM responds by pulling DO low for the duration of the ERASE cycle. ERASE timing is shown in figure 3.



AT93Cx6 Erase Cycle

Figure 3

H8/330 - SERIAL EEPROM DESIGN

The remainder of this document describes the hardware and software required to interface a serial EEPROM add-on board to the H8/330/338 evaluation board. The add-on board consists of a 3-wire, 4K serial EEPROM from ATMEL, indicator LED's and a CMOS inverting Schmidt trigger. The schematic is given at the end.

A serial communication link was established between the H8/330 SCI port and the SEEPROM by tying RxD, TxD and SCK lines to DO, DI and SK respectively. The serial port was set for synchronous operation at 100K using an internal clock source. I/O port pin 8.0 provided a CS line.

The SEEPROM transmits data on the rising edge of the clock and guarantees valid data 250 ns after the clock edge. The H8/33x, however, expects to receive data on the same rising clock edge. Therefore, to guarantee operation, the serial clock output to the SEEPROM was inverted via the Schmidt trigger to reverse the polarity.

Three LEDs on the connector board signalled READ, WRITE and ERASE cycles.

PREPARING COMMAND FRAMES

The SCI port sends and receives 8-bit data LSB first. Unfortunately the SEEPROM sends and expects to receives 8-bit information MSB first. Therefore, software must transpose the first byte of the command frame, MSB to LSB, for the SEEPROM to correctly interpret the start bit and op-code. Addresses are also scrambled by transmission but the actual storage location is invisible to the user and all locations are accessible. Data bytes, inverted during transmission, will be re-inverted by the H8/330 during reception. If other devices will access the SEEPROM and those devices follow the MSB first protocol, address and data bytes should also be transposed for correct operation. The routine "ROTATE" performs the MSB to LSB inversion operation with minimal CPU overhead. In this software example command, address and data bytes are all transposed MSB to LSB before sending.

The routine "FORMAT" concatenates address bit A8 with the LSB of the command byte and transposes the data as described above. The final command frame resides at the address held in R5.

THE WRITE OPERATION

The write operation routine provided, ("WRITE"), executes a write cycle in three phases:

1. The H8/330 initiates a write cycle by sending the 3 byte WRITE command frame containing: the command byte plus A8, the write address byte and the data byte. After sending the data byte, the CPU polls the TDRE bit (bit 7, SCI_SSR register). When TDRE goes high, the data byte has transferred to the Transmit Shift Register (TSR). The CPU then writes a dummy byte to the TDR, clears TDRE and polls TDRE again. A second high at TDRE indicates the dummy byte has moved into the TSR and the last bit of the data byte (D0) has transferred out of the TSR. At this point, software can disable the transmitter without losing valid data.

2. The CPU toggles the CS line (P8.0) by executing two sequential BNOT instructions, turns off the transmitter and starts Timer 1. Timer 1 counts down a 940 us delay (see note) to allow the SEEPROM time to complete the write cycle. During this period the CPU can either enter a sleep state or return to a main routine until timer 1 interrupts the CPU.

Note: Although ATMEL and National Semiconductor specify the write and erase cycle time (twp) as 10 ms maximum, the delay experienced in this example was actually < 950 us.

3. At the end of the delay period the timer interrupts the CPU to start the receiver. The receiver continues to read data on the DO (RxD) line until it detects a value > 0. This indicates the SEEPROM has completed the write cycle and released DO high. The CPU repeats the write sequence until the byte count held in R4 is exhausted.

Timing for this sequence is shown below in figure 4:

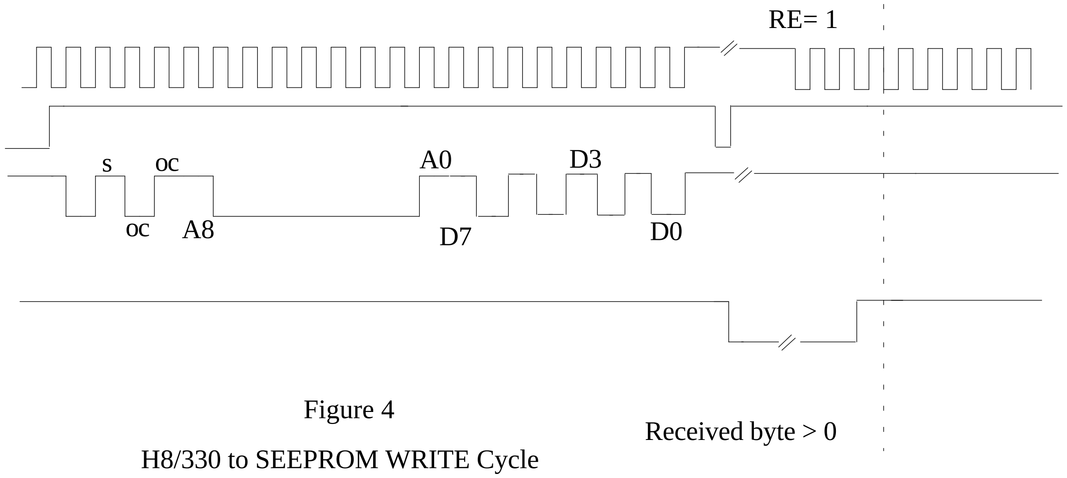


Figure 4

H8/330 to SEEPROM WRITE Cycle

The H8/330 writes the value H'AA to address H'101

THE READ OPERATION

The read cycle requires a two byte command frame containing the command byte and lower address byte. Immediately after the last valid byte moves into the TSR, the CPU starts a timer to count a 7.5 bit delay. The CPU then starts the receiver to capture the data sent by the SEEPROM.

It is important to note: When enabling the receiver while the transmitter is active, the TDRE bit must be = 1 when RE is set. If TDRE= 0 when RE is set to 1, the receiver will not start.

Timing for this last step is fairly critical. The receiver must start in time to capture the first data bit, D0, but still avoid erroneous data on the front end.

The SEEPROM precedes each data byte with a half clock low pulse. In this example timer 1 counts a 7.5 bit clock delay before starting the receiver. This allows the last valid address byte to shift out of the TSR and the SEEPROM to respond with the first erroneous low pulse before the receiver begins to read data.

The actual amount of delay time depends upon the bit transfer rate as well as the CPU instruction execution time. A 100K bit clock produces a 10 us bit period. A 75 us clock should provide the necessary delay, however, the time required for the CPU to recognize the timer 1 time-out condition and respond could cause the receiver to miss D0. In this example a logic analyzer was used to pinpoint the correct delay period. Figure 5 gives the timing diagram for this cycle.

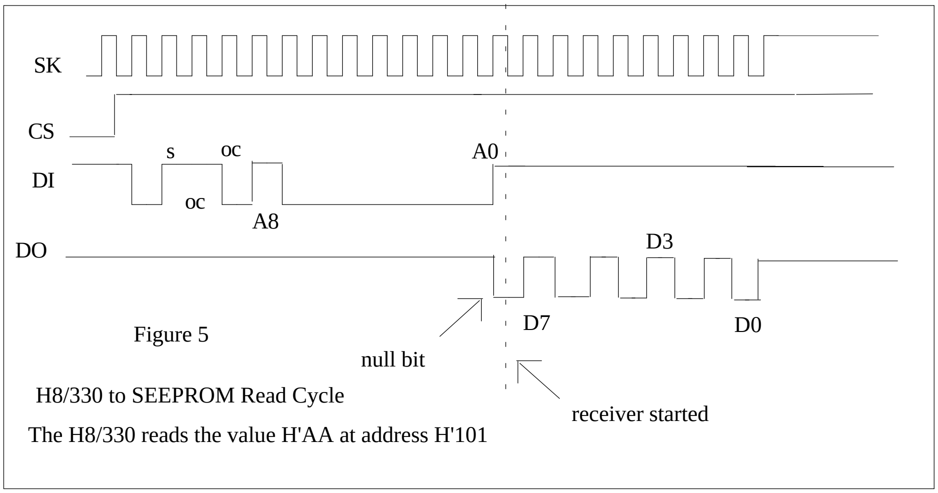


Figure 5

H8/330 to SEEPROM Read Cycle

The H8/330 reads the value H'AA at address H'101

THE ERASE OPERATION

The erase cycle proceeds in much the same manner as the write cycle but without the data byte The erase command frame contains two bytes, the command byte and the address to erase. As with the write cycle, after the last valid data byte the CPU toggles CS to initiate a BUSY response from the SEEPROM and starts timer 1 for an interrupt controlled delay.

The ERASE cycle requires the serial clock to remain active for the duration of the cycle. In this example, software holds the serial clock active by continuing to load h'00 into the TDR. At the end of the delay period the interrupt routine starts the receiver to begin looking for a high state on DO. When DO is detected high both the transmitter and receiver are disabled. The SEEPROM ERASE cycle is shown in figure 6.

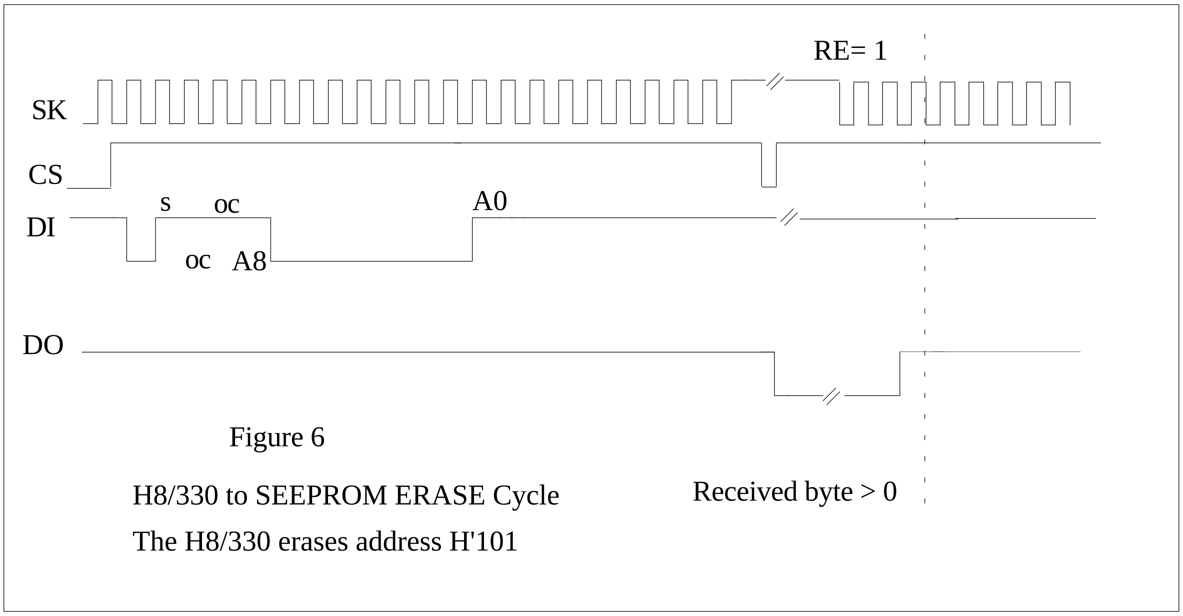


Figure 6

H8/330 to SEEPROM ERASE Cycle

The H8/330 erases address H'101

SEEPROM ENABLE AND DISABLE COMMANDS

Software must execute the global command, EWEN, to enable write or erase cycles after power-up. In the same manner as a standard command, the transmitter sends a serial bit stream containing the EWEN sequence (there is no address or data byte). The routine "EWEN" executes this sequence.

Likewise, the SEEPROM can be disabled by executing the EWDS sequence. The routine "EWDS" will disable write and erase operations to protect memory contents from accidental corruption.

ADDITIONAL COMMANDS

Most SEEPROMs support global commands to access all address locations for ERASE or WRITE operations. These commands are generally used either during start-up procedures or during a test sequence. Routines "ERAL" and "WRAL" erase and write all locations respectively. The command procedure is basically the same as for a single access cycle. Please refer to the manufacturer's literature for more details.

SELECTION OF 2, 3 OR 4-WIRE INTERFACE

Serial EEPROM's supporting 2, 3 or 4-wire handshakes are available from ATMEL and 2 or 3-wire devices are available from National Semiconductor. Protocol selection depends upon the user's system requirements, required transfer speed and the level of CPU overhead that can be tolerated.

The interface between a 2-wire SEEPROM and the H8/33x must be handled by software control of two I/O port lines. Therefore the maximum data transfer rate is limited by the rate at which the CPU can interpret the required port state and write to the port data register. This time will vary by the user's program and code location (on-chip versus off-chip memory), however, it has been shown that the maximum rate of 100K is achievable with certain restrictions. The 2-wire protocol requires the greatest CPU overhead and only operates to 100K, however, the interface only needs two I/O ports and access to a timer.

The 3-wire interface offers a significant improvement over the 2-wire by allowing use of the on-chip serial port. When the serial port is idle the CPU can return to a main routine or move into the low power sleep mode.

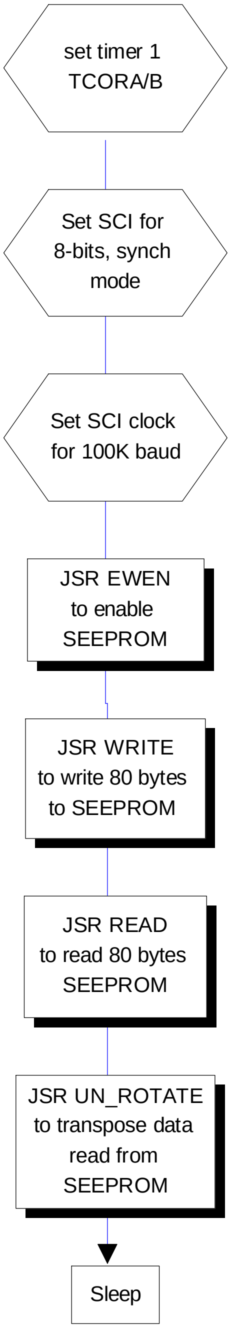
In the example shown, the on-chip baud rate generator supplied the transmit and receive clock. Unfortunately this means the CPU must continue to feed data to the TDR to hold the serial clock active for the duration of the ERASE cycle. As an alternative, timer 0 or one of the PWMs could be used as an external clock source for the SCI and SEEPROM. This technique avoids the additional CPU overhead needed to maintain the clock but requires use of another peripheral.

The third generally available interface, the 4-wire handshake, is almost identical to the 3-wire interface but separates the RDY and DO signals. The addition of a 4th line, RDY/BUSY*, makes this the easiest format to use with the H8/33x. Rather than forcing the receiver to act as a pseudo port line to detect the end of a write or erase cycle, the CPU can poll the RDY/BUSY* line.

FLOWCHARTS

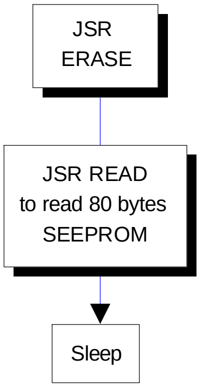
The following flow charts diagram software routines presented in the next section.

Initialization & Demo



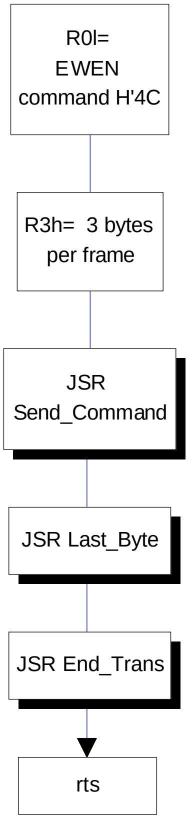
80 bytes of data are written to the SEEPROM then read back and stored beginning at address H'FDD0 (read_data)

After viewing data, the SEEPROM is erased by executing the ERASE routine. The erase is verified by reading back the memory contents and saving the results again beginning at H'FDD).



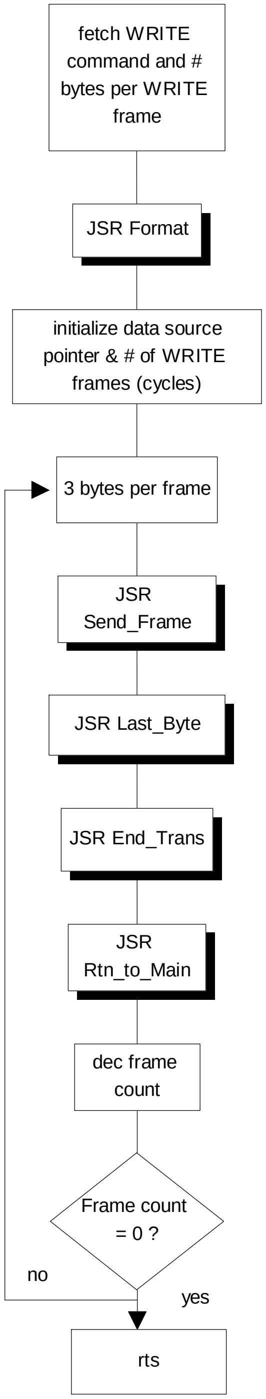
Erase \ Write Enable (EWEN)

This routine enables erase and write operations to the SEEPROM



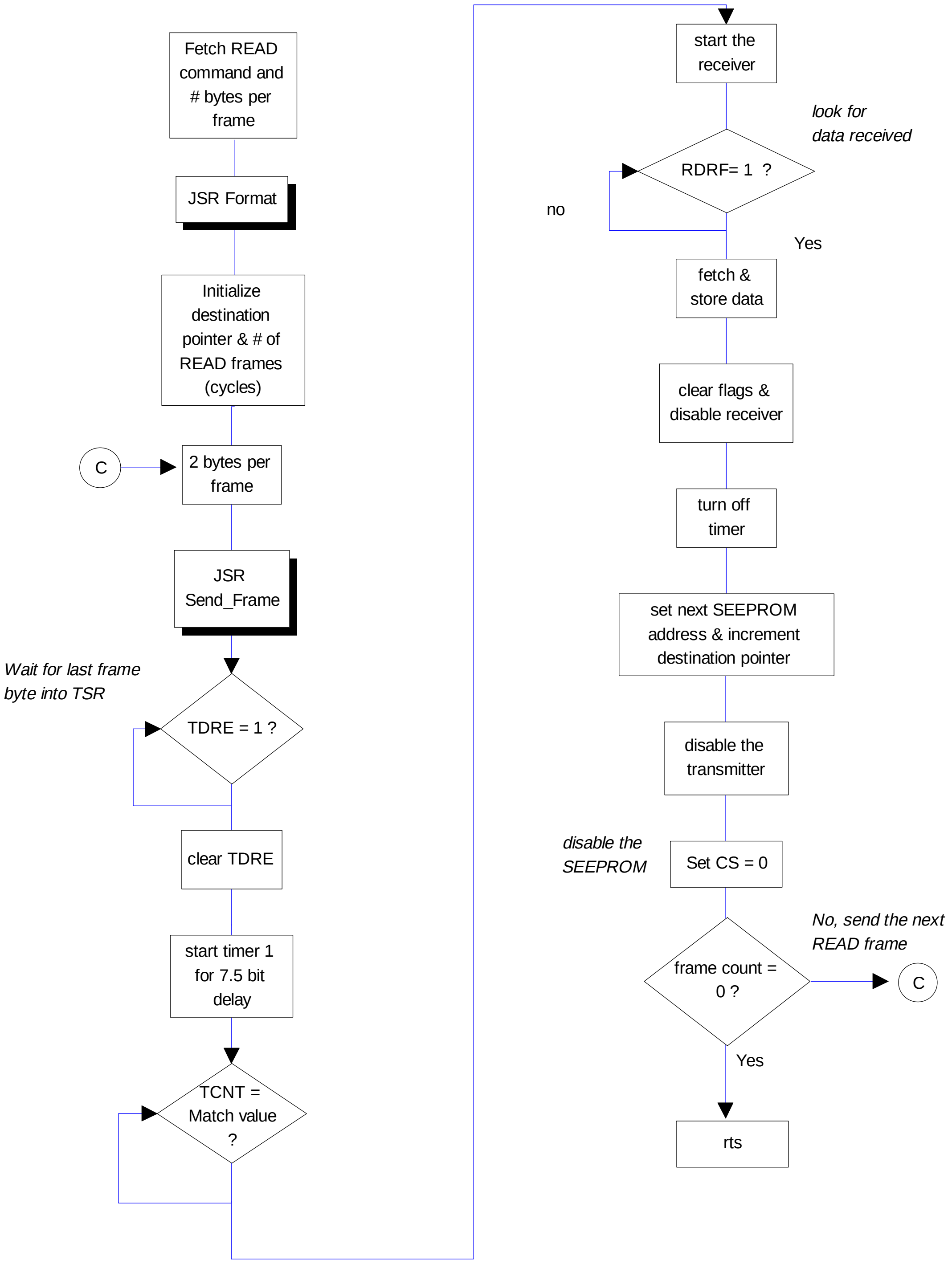
WRITE Cycle

This routine writes 1 to 255 bytes to 4K Serial EEPROM
(for demo purpose this value is set to 80 bytes)



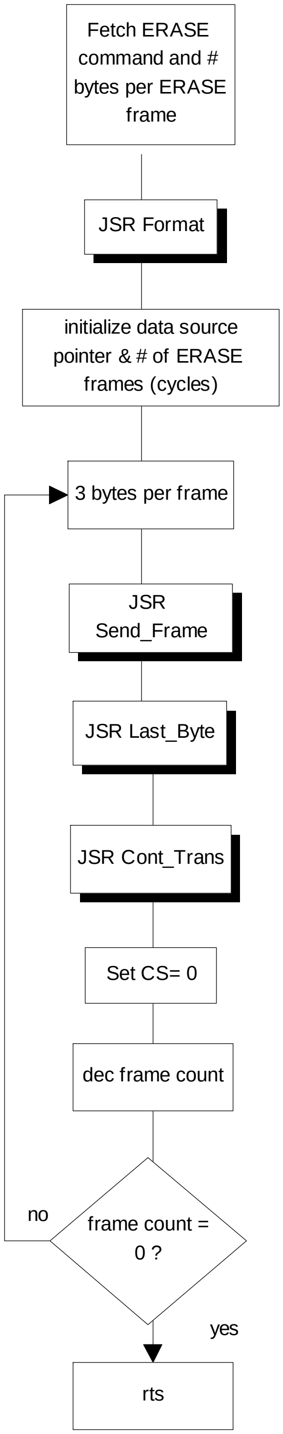
Read Cycle

This routine reads 1 to 255 bytes of data from a 4K Serial EEPROM
(for demo purpose this value is set to 80 bytes)

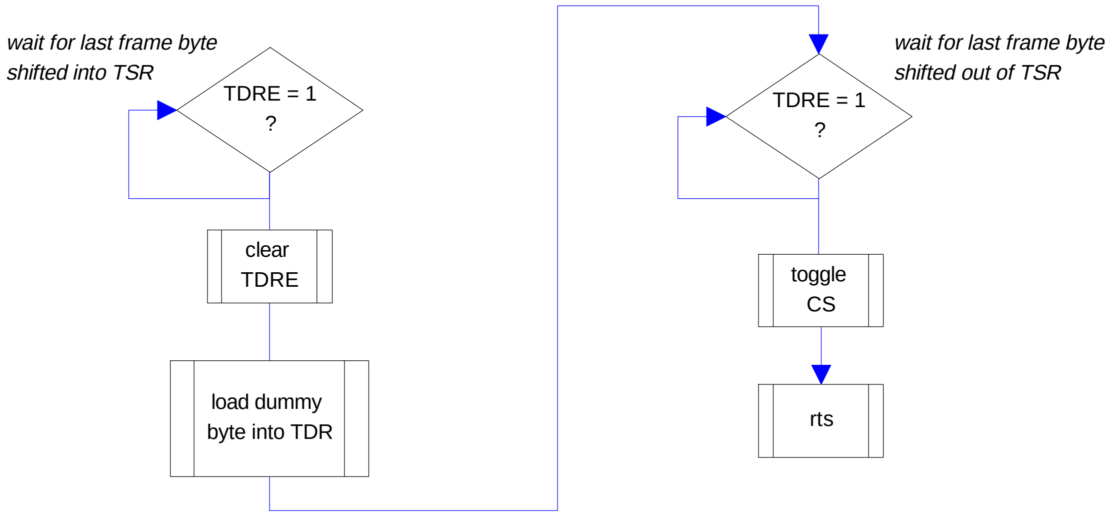

Fetch READ command and # bytes per frame
JSR Format
Initialize destination pointer & # of READ frames (cycles)
C
2 bytes per frame
JSR Send_Frame
Wait for last frame byte into TSR
TDRE = 1 ?
clear TDRE
start timer 1 for 7.5 bit delay
TCNT = Match value ?
start the receiver
look for data received
RDRF= 1 ?
no
Yes
fetch & store data
clear flags & disable receiver
turn off timer
set next SEEPROM address & increment destination pointer
disable the transmitter
disable the SEEPROM
Set CS = 0
No, send the next READ frame
frame count = 0 ?
C
Yes
rts

ERASE Cycle

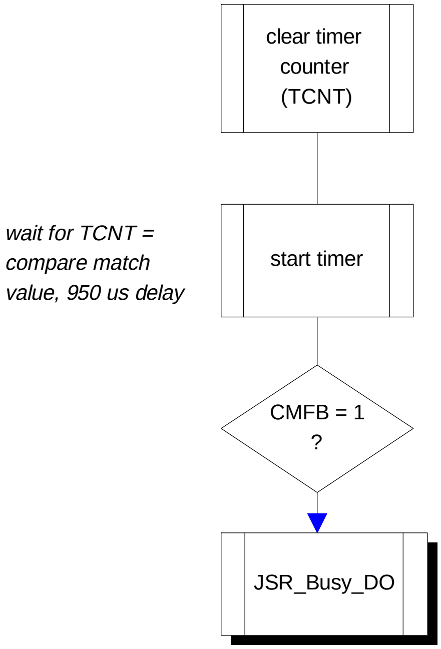
This routine erases 1 to 255 addresses of 4K Serial EEPROM
(for demo purpose this value is set to 80 bytes)

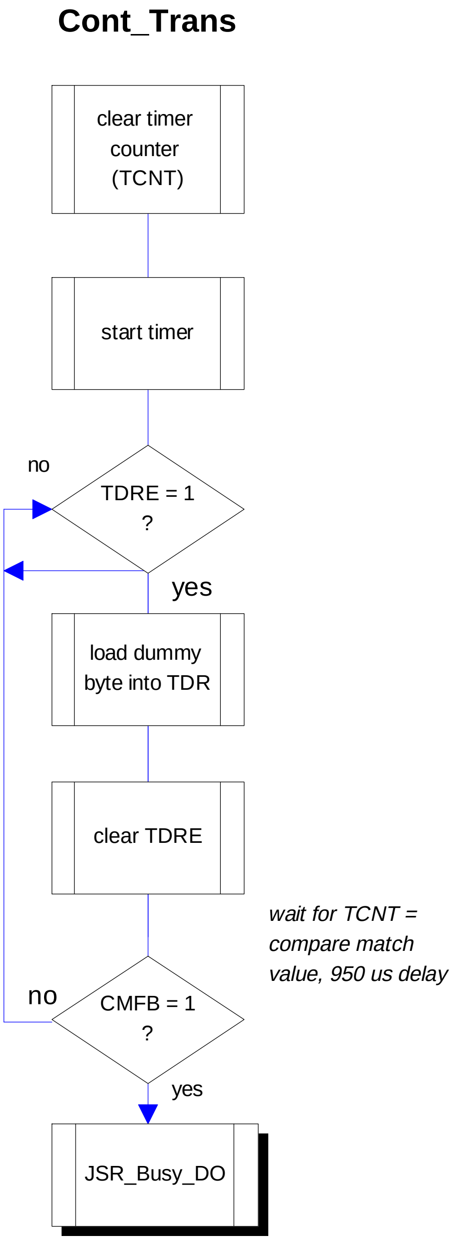


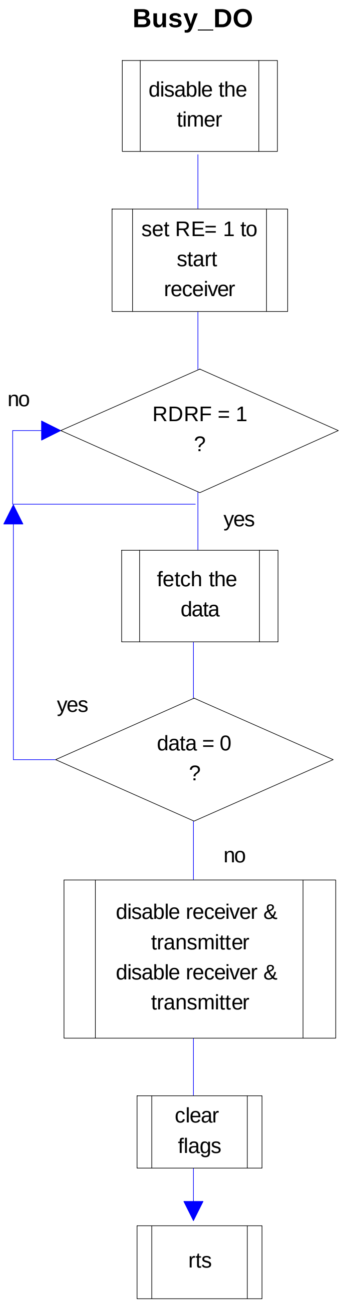
Last_Byte



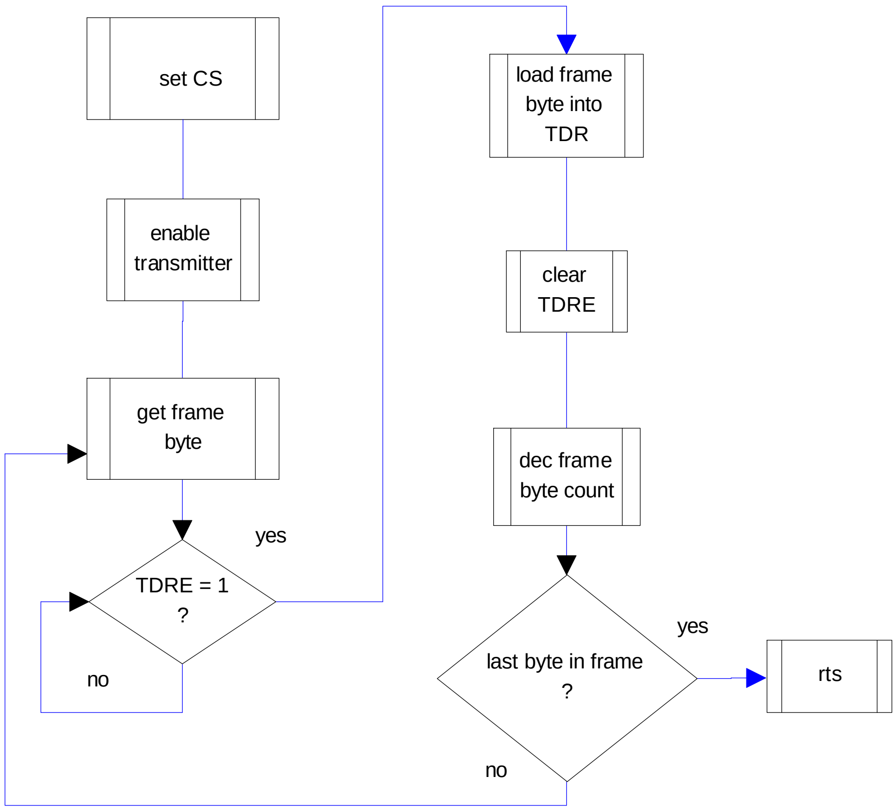
Rtn_to_Main




Cont_Trans
clear timer counter (TCNT)
start timer
no
TDRE = 1 ?
yes
load dummy byte into TDR
clear TDRE
wait for TCNT = compare match value, 950 us delay
no
CMFB = 1 ?
yes
JSR_Busy_DO

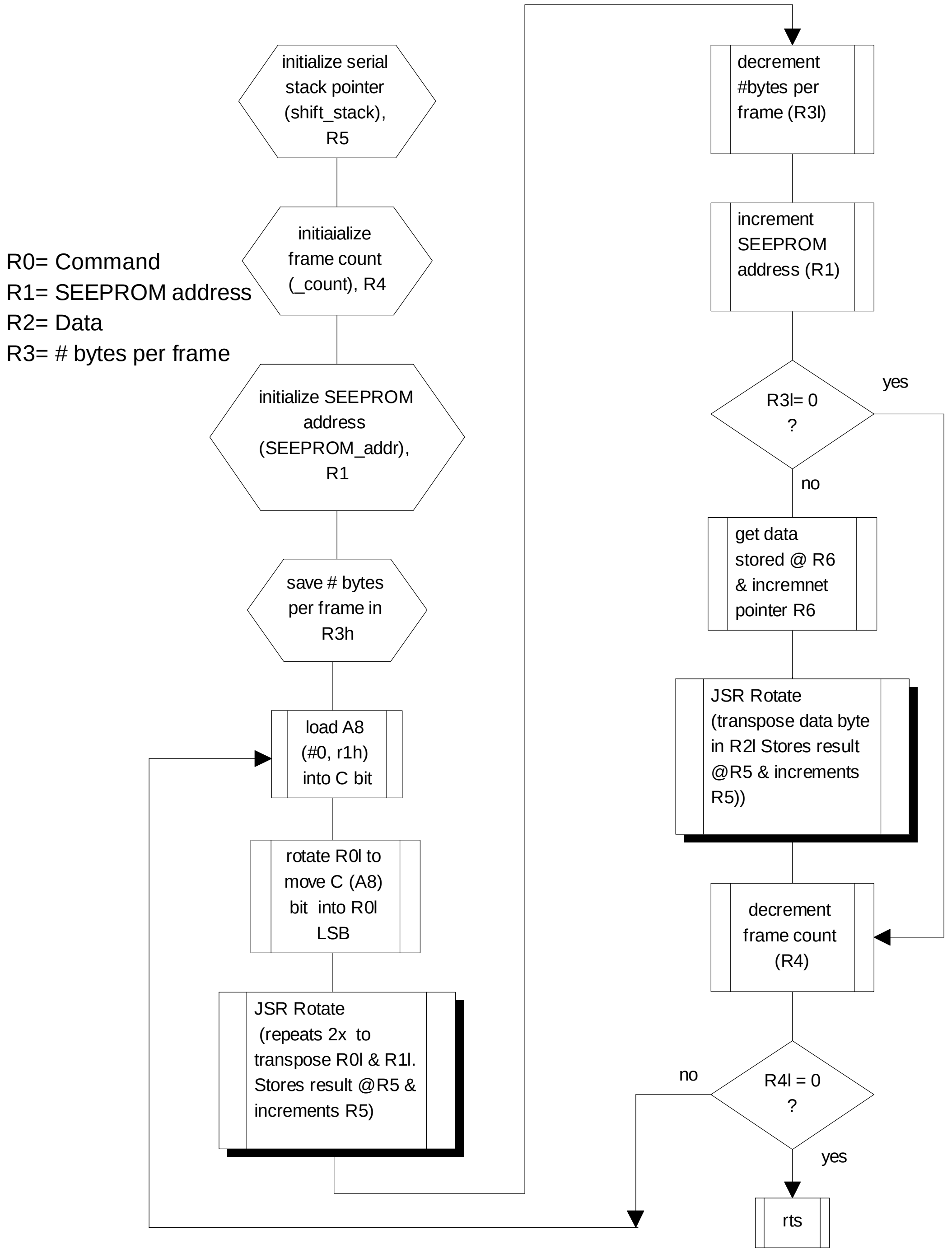

Busy_DO
disable the timer
set RE= 1 to start receiver
no
RDRF = 1 ?
yes
fetch the data
yes
data = 0 ?
no
disable receiver & transmitter disable receiver & transmitter
clear flags
rts

Send _Frame


set CS
enable transmitter
get frame byte
TDRE = 1 ?
yes
no
load frame byte into TDR
clear TDRE
dec frame byte count
last byte in frame ?
yes
no
rts

Format

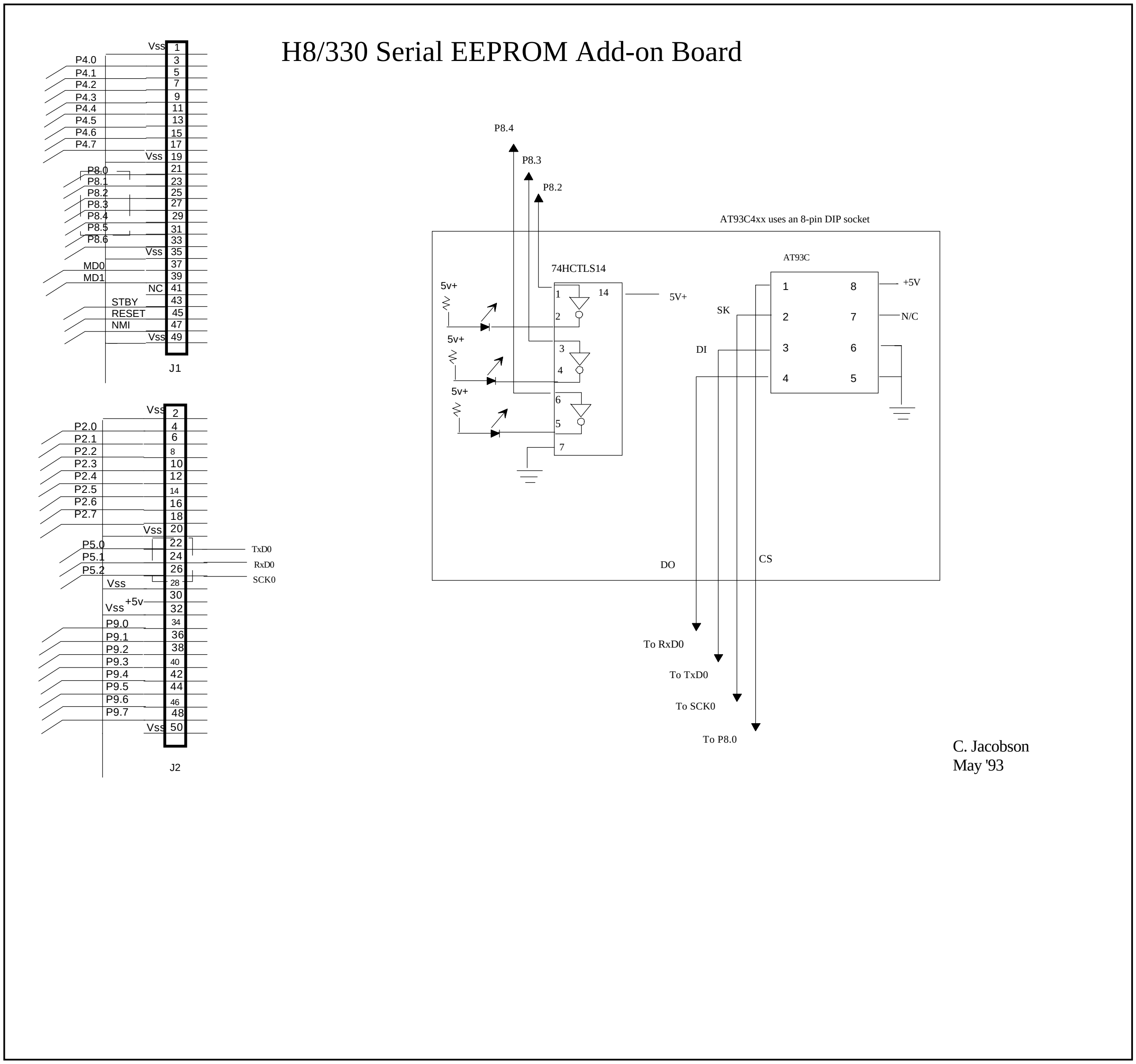
This routine formats the command byte, SEEPROM address and data into a sequential serial frame. Each frame is stored at the location pointed to by R5.


initialize serial stack pointer (shift_stack), R5
initiaialize frame count (_count), R4
R0= Command
R1= SEEPROM address
R2= Data
R3= # bytes per frame
initialize SEEPROM address (SEEPROM_addr), R1
save # bytes per frame in R3h
load A8 (#0, r1h) into C bit
rotate R0l to move C (A8) bit into R0l LSB
JSR Rotate (repeats 2x to transpose R0l & R1l. Stores result @R5 & increments R5)
decrement #bytes per frame (R3l)
increment SEEPROM address (R1)
R3l= 0 ?
yes
no
get data stored @ R6 & incremnet pointer R6
JSR Rotate (transpose data byte in R2l Stores result @R5 & increments R5))
decrement frame count (R4)
no
R4l = 0 ?
yes
rts

SOURCE LISTING

```
;*****************************************************************************
;
;H8/330 to SEEPROM Driver/ Demo Routines
;
;The following routines rely upon availability of all CPU registers.  Before calling
these as subroutines the user may want ;to store all register values.
;
                .include         "c:\demos\H8330.inc"


command                 .equ    h'ff10          ;RAM storage for command
_count                  .equ    H'50            ;number of addresses/data
read_data               .equ    h'fdd0          ;RAM destination for READ
                                                ;cycle data, fdd0 - fd20
ram_data                .equ    h'fd80          ;RAM source, for WRITE
                                                ;cycle data, fd80 - fdd0
SEEPROM_addr            .equ    h'80            ;SEEPROM starting address
shift_stack             .equ    h'fe20          ;storage for serial frames
                                                ;240 locations, h'fe20 - h'ff10

READ_led                .equ    h'02                    ;READ LED
ERASE_led               .equ    h'03                    ;ERASE LED
WRITE_led               .equ    h'04                    ;WRITE LED
Read_cmd                .equ    h'0006
Write_cmd               .equ    h'0005
Erase_cmd               .equ    h'0007

                        .org    h'8000

start:
                        mov.w   #top_ram,r7;
;
;**********************************************************************************

;
;  -  All routines require heavy use of the CPU registers.
;
;  -  NOP instructions are placed at the end of each routine only to differentiate
;code sections.  The NOP's have no effect on code operation.
;
;**********************************************************************************
;Load sample data into memory for demo

        mov.w   #_count, r4
        mov.w   #ram_data,r2            ;addr/data stack

set:    mov.b   #h'AA,r0h

line:   mov.w   #1000,r1

data:   mov.b   r0h,@r2
        dec     r4l
        beq     timer1
        inc     r2l
        inc     r1l
        cmp     r1h,r1l
        bne     data
        add.b   #h'11,r0h
        cmp     #h'10,r0h
        bne     line
        bra     set

        nop
        nop
        nop
        nop

;***********************************************************************************
;set-up timer1 Match B for write cycle delay & Match A for read cycle delay
timer1:
        mov.b   #h'85,r0l
        mov.b   r0l,@tmr1_tcorb         ;set match b= 85,
        mov.b   #h'53,r0l
        mov.b   r0l,@tmr1_tcora         ;set match a=

        nop
        nop
        nop
        nop

;***********************************************************************************

;Set Up SCI Port & port 8

        mov.b   #h'80,r0l
        mov.b   r0l,@sci0_smr           ;sync mode 8 bits system clock
        mov.b   #h'80,r0l
        mov.b   r0l,@sci0_ssr
        mov.b   #h'18,r0l
        mov.b   r0l,@sci0_brr
        mov.b   #h'1f,r0l
        mov.b   r0l,@p8_ddr
        mov.b   #h'00,r0l
        mov.b   r0l,@p8_dr              ;pull CS & LEDs low
;***********************************************************************************

;Begin Demo:
;
;This demo writes a block of data, initially stored at location
;RAM_data (H'FB80), to SEEPROM starting at address H'80.  The Data is
;read back, transposed and stored at location Read_data (H'FC80).  The trans-
;position step actually occurs before sending to SEEPROM and after reading
;SEEPROM.  This step can be eliminated if the SEEPROM will only be accessed
;by the H8/300.

;After viewing memory contents, the SEEPROM is erased and verified by executing
;code located at H'8080

        jsr     @EWEN           ;Enable the SEEPROM
        jsr     @WRITE          ;Write data from RAM_data to SEEPROM
        jsr     @READ           ;Read data and store at H'FC80
        jsr     @UN_ROTATE      ;Reverse data bytes LSB to MSB
        sleep                   ;end of demo, push NMI to return EVB to monitor
                                ;control and view data
        nop
        nop
        nop
        nop
;***********************************************************************************

        .org h'8080

        jsr    @erase           ;erase data written in 1st alf of demo
        jsr    @read            ;view results of erase cycle.
        sleep                   ;end of second half of demo. push NMI to return
                                ;EVB to monitor control and view data
        nop
        nop
        nop
        nop
;***********************************************************************************

;***********************************************************************************

EWEN:
        mov.b   #h'32,r0l       ;load ewen (b'0100 1100) command into r0l,
                                ;MSB into LSB (b' 0011 0010)
        mov.b   #h'03,r3h       ;3 bytes for ewen
S_ewen:
        jsr     @send_command
        jsr     @last_byte
        jsr     @end_trans
        rts

        nop
        nop
        nop
        nop

;***********************************************************************************

;***********************************************************************************

READ:
        mov.w   #read_cmd,r0            ;read command @command
        mov.w   r0,@command
        mov.b   #h'02,r3h               ;2 bytes per read frame

        jsr     @Format                 ;format the command frames and
                                        ;store @R5

        mov.w   #READ_data,r6           ;RAM destination pointer
        mov.w   #shift_stack,r5         ;re-initialize shift stack
        mov.w   #_count, r4             ;re-initialize frame count

        bset    #READ_led,@p8_dr        ;set the read LED

rd_frame:
        mov.b   #h'02,r3h               ;2 bytes per read frame
        jsr     @send_frame             ;send command frame
        mov.b   #0,r0l
tdre0:  btst    #7,@sci0_ssr            ;wait for TDRE= 1 to ensure
                                        ;2nd byte into TSR
        beq     tdre0
        mov.b   r0l,@sci0_tdr

;start timer 1 for delay

        mov.b   r0l,@tmr1_tcnt          ;clear the timer
        bclr    #6,@tmr1_tcsr           ;clear match A
        mov.b   #h'09,r0h
        mov.b   r0h,@tmr1_tcr           ;start timer 1 for a polled delay
 T1A:   btst    #h'6,@tmr1_tcsr         ;wait for match A
        beq     T1A

        bset    #4,@sci0_scr            ;start the receiver @ 100K
```

```
rdrfa:  btst    #6,@sci0_ssr            ;look for RDRF = 1
        beq     rdrfa

        mov.b   @sci0_rdr, r3l          ;get the data
        mov.b   r3l,@r6
        adds.w  #1,r6                   ;increment destination pointer

        bclr    #4,@sci0_scr            ;turn receiver off
        bclr    #5,@sci0_ssr            ;clear the overflow bit
        bclr    #6,@sci0_ssr            ;clear the RDRE bit
        mov.b   r0l,@tmr1_tcr           ;turn off the timer
        bclr    #6,@tmr1_tcsr           ;clear the match flag

        jsr     @end_trans              ;turn off transmitter
        bclr    #0,@p8_dr               ;CS= low

        adds.w  #1,r1                   ;set up next SEEPROM address

        dec     r4l                     ;decrement r4, frame count
        bne     rd_frame                ;send the next frame

        bclr    #READ_led,@p8_dr        ;clear the read LED
        rts


        nop
        nop
        nop
        nop
;*********************************************************************************

;*********************************************************************************

;This routine writes an block of 80 bytes to the SEEPROM.
;
WRITE:
        mov.w   #write_cmd,r0
        mov.w   r0,@command             ;write command @command
        mov.b   #h'03,r3h               ;3 bytes for write command frame
        mov.w   #RAM_data,r6            ;load data location in RAM•

        jsr     @Format

        mov.w   #shift_stack,r5         ;re-initialize shift stack
        mov.w   #_count, r4             ;re-initialize frame count

        ldc     #0,ccr                  ;enable interrupts
        bset    #WRITE_led,@p8_dr       ;turn on WRITE LED

nxt_wr:
        mov.b   #h'03,r3h               ;3 bytes per write frame

        jsr     @send_frame
        jsr     @last_byte
        jsr     @end_trans
        jsr     @rtn_to_main


        dec     r4l                     ;decrement frame count
        bne     nxt_wr                  ;write the next byte
        bclr    #WRITE_led,@p8_dr       ;turn off the write LED
        rts

        nop
        nop
        nop
        nop
;***********************************************************************************

;This routine erases a block of 80 bytes starting as SEEPROM address h'80
;
ERASE:
        mov.w   #erase_cmd,r0           ;erase command in r0
        mov.w   r0,@command
        mov.b   #h'02,r3h               ;2 bytes for erase command frame

        jsr     @Format

        mov.w   #shift_stack,r5         ;re-initialize shift stack
        mov.w   #_count, r4             ;re-initialize frame count

        ldc     #0,ccr                  ;enable interrupts
        bset    #ERASE_led,@p8_dr       ;turn on ERASE LED

nxt_er:
        mov.b   #h'02,r3h               ;2 bytes per erase frame
        jsr     @send_frame             ;send address to erase
        jsr     @last_byte              ;wait for last byte & toggle CS
        jsr     @cont_trans             ;start timer 1 and continue serial clock
        bclr    #0,@p8_dr               ;clear CS

;the serial clock continues to operate until timer 1 reaches the match B value and
;issues an interrupt. The interrupt routine, BUSY_D0, starts the receiver to look
;for rising edge or high on the D0 line indicating the erase cycle is complete.

        dec     r4l                     ;decrement frame count
        bne     nxt_er                  ;erase the next byte

        bclr    #ERASE_led,@p8_dr       ;turn off the erase LED
        rts
        nop
        nop
        nop
        nop

;***********************************************************************************

;***********************************************************************************

WRAL:
;This routine writes a fixed data value to all addresses.  The data value is stored
;in r1l,  r0l contains the WRAL command. In this example the value written is 'AA'

        mov.w   #h'0022,r0      ;load 5-bit command (b' 010001) into r0l loading
                                ;MSB into r0l LSB (re-align the command)
        mov.b   #h'aa,r1l
        mov.b   #h'03,r3h       ;3 bytes per frame
S_wral:
        jsr     @send_frame
        jsr     @end_trans
        jsr     @stop

        nop
        nop
        nop
        nop

;*********************************************************************************

ERAL:
        mov.b   #h'12,r0l       ;load eral (b'01001000) command into r0l, MSB into
                                ;LSB (b' 00010010)
        mov.b   #h'02, r3h      ;2 bytes for eral
S_eral:
        jsr     @send_command
        jsr     @cont_trans
        jsr     @end_trans
        jsr     @stop

        nop
        nop
        nop
        nop

;*********************************************************************************

;*********************************************************************************

EWDS:
        mov.b   #h'02,r0l               ;load ewen (b' 01000000) command into r0l,
                                        ;MSB into LSB (b' 00000010)
S_ewds:
        mov.b   #h'03,r3h               ;3 bytes for ewen
        jsr     @send_command
        jsr     @end_trans
        jsr     @stop

        nop
        nop
        nop
        nop

;*********************************************************************************

;*********************************************************************************

send_frame:
        bset    #0,@p8_dr               ;set SE2PROM CS signal
        bset    #5,@sci0_scr            ;set the TE bit to enable transmission
nxt_byte:
        mov.b   @r5+,r0l
tdref:  btst    #7,@sci0_ssr            ;wait for TDRE= 1
        beq     tdref
        mov.b   r0l,@sci0_tdr
        bclr    #7,@sci0_ssr
        dec     r3h
        bne     nxt_byte
        rts

        nop
        nop
        nop
        nop

;*********************************************************************************

send_command:
        bset    #0,@p8_dr                ;set SEEPROM CS signal
        bset    #5,@sci0_scr             ;set the TE bit to enable transmission
tdrec:  btst    #7,@sci0_ssr             ;wait for TDRE= 1
        beq     tdrec
        mov.b   r0l,@sci0_tdr
        bclr    #7,@sci0_ssr
        dec     r3h
        bne     tdrec
        rts

        nop
        nop
        nop
        nop

;*********************************************************************************

last_byte:
        btst    #7,@sci0_ssr             ;test tdre bit
        beq     last_byte                ;look for last data byte into TSR
        mov.b   r0l,@sci0_tdr
tdre_end:
        bclr    #7,@sci0_ssr
        btst    #7,@sci0_ssr             ;look for dummy byte into TSR,
                                         ;indicating last
        beq     tdre_end                 ;data byte done

        bnot    #0,@p8_dr                ;toggle CS
        bnot    #0,@p8_dr                ;toggle CS

        rts
        nop
        nop
        nop
        nop

;*********************************************************************************

;*********************************************************************************


end_trans:
        bclr    #5,@sci0_scr             ;clear the TE bit to disable the transmitter
        rts

        nop
        nop
        nop
        nop
;*********************************************************************************

;*********************************************************************************


;After each write frame, end transmission, toggle CS, and start timer 1 for a 950 us
;delay to allow the cycle to complete and D0 to go high again.  The timer 1 match B
;interrupt routine starts the receiver to to look for a high level  on RxD (data >
;than 0).

rtn_to_main:
        mov.b   #0,r0h
        mov.b   r0h,@tmr1_tcnt
        mov.b   #h'13,r0h
        mov.b   r0h,@tmr1_tcr           ;start timer 1
 mtchb1:
        btst    #7,@tmr1_tcsr
        bne     mtchb1
        jsr     @Busy_D0
        rts
        nop
        nop
        nop

;***********************************************************************************

;This routine continues to send dummy bytes on TxD to keep the serial clock active
;during the ERASE cycle.
;
cont_trans:
        mov.b   #0,r2h                  ;dummy byte
        mov.b   #h'13,r4h
        mov.b   r2h,@tmr1_tcnt          ;clear the timer
        bclr    #7,@tmr1_tcsr           ;clear the match b bit
        mov.b   r4h,@tmr1_tcr           ;start timer 1

tdrey:  btst    #7,@sci0_ssr            ;look for TDRE= 1
        beq     tdrey
        mov.b   r2h,@sci0_tdr           ;re-load dummy byte
        bclr    #7,@sci0_ssr
        btst    #7,@tmr1_tcsr           ;timer 1 = match b value ?
        beq     tdrey
        jsr     @Busy_D0
        rts

        nop
        nop
        nop
        nop

;***********************************************************************************

;***********************************************************************************

;THIS SECTION OF CODE PREPARES DATA FOR TRANSMISSION

;This routine formats the start bit, opcode, address and data to produce a command
;frame.  Each byte is transposed to invert MSB to LSB. The routine can format single
;frames to access one address or multiple frames for block operations.  The
;formatted data is stored as command, address, data (if any) beginning at the
;address stored in R5.  The calling routine must load the start bit & opcode in r0l,
;the data pointer in R6 and the number of bytes per frame in r3h.
;
Format:

        mov.w   #shift_stack, r5        ;serial stack pointer
        mov.b   #_count, r4l            ;number of frames to form
        mov.w   #SEEPROM_addr,r1        ;SEEPROM starting address•

;r0l = command   r1h = A8  r1l = A7-A0   r2l =D7 - D0

step1:
        mov.b   r3h,r3l                 ;number of bytes per frame

        mov.w   @command,r0

        bld     #0,r1h          ;mov A8 into C bit
        rotxl   r0l             ;rotate r0l & concatinate A8 with r0l
                                ;r0l now contains start bit + op code + A8

;concatinate r0l, r1l and r2l to form a continuous frame

        mov.b   @r6+,r2l        ;get data byte (if any)

;transpose the bits and store result @R5

step3:
        mov.b   r0l,r0h
        jsr     @rotate                 ;re-align the command byte

        mov.b   r1l,r0h                 ;re-align lower address byte
        jsr     @rotate
        mov.b   r0h,r1l                 ;re-store SEEPROM lower address
        adds.w  #1,r1                   ;next SEEPROM address

        dec     r3l
        dec     r3l                     ;subtract 2 from frame byte count
        beq     nxt

        mov.b   r2l,r0h                ;re-align data byte (if any)
        jsr     @rotate

nxt:    dec     r4l
        bne     step1
        rts
        nop
        nop
        nop
        nop

;***********************************************************************************

BUSY_D0:
;
        mov.b   #0,r0h
        mov.b   r0h,@tmr1_tcr           ;disable the timer
        bset    #4,@sci0_scr            ;start the receiver
rdrf:   btst    #6,@sci0_ssr            ;look for RDRF = 1
        beq     rdrf
        bclr    #6,@sci0_ssr            ;clear the rdre bit
        mov.b   @sci0_rdr, r3l          ;get the data
        cmp.b   #00,r3l                 ;compare the data to 0
        beq     rdrf                   ;if the data is not = 0 then D0 has gone high
        bclr    #4,@sci0_scr            ;turn receiver off
        bclr    #5,@sci0_ssr            ;clear the overflow bit
        bclr    #5,@sci0_scr            ;turn the transmitter off for erase cycle
        bclr    #7,@tmr1_tcsr           ;clear timer 1 match b flag
        bclr    #7,@sci0_ssr

        bnot    #0,@p8_dr
        bnot    #0,@p8_dr
        rts
        nop
        nop

;***********************************************************************************

;***********************************************************************************

UN_ROTATE:
        mov.w   #_count,r4              ;load byte count
        mov.w   #Read_data,r5           ;load data address pointer

un_rvs: mov.b   @r5,r0h                 ;reverse data
        jsr     @rotate
        dec     r4l
        bne     un_rvs                  ;
        rts

;***********************************************************************************

;***********************************************************************************

rotate:
                bld     #7,r0h
                bst     #0,r0l
                bld     #6,r0h
                bst     #1,r0l
                bld     #5,r0h
                bst     #2,r0l
                bld     #4,r0h
                bst     #3,r0l
                bld     #3,r0h
                bst     #4,r0l
                bld     #2,r0h
                bst     #5,r0l
                bld     #1,r0h
                bst     #6,r0l
                bld     #0,r0h
                bst     #7,r0l
                mov.b   r0l,@r5
                adds.w  #1,r5

                rts
                .end

;***********************************************************************************
```



The information contained in this document has been carefully checked, however the contents of this document may be changed and modified without notice. Hitachi America, Ltd. shall assume no responsibility for inaccuracies, or any problem involving patent infringement caused when applying the descriptions in this document. This material is protected by copyright laws. Hitachi America, Ltd. reserves all rights.