I^2C implementation: Accessing the 24LC01 Serial EEPROM

Introduction

This application note presents programming techniques for reading from and writing to a serial EEPROM using I^2C data transfer protocol. This implementation uses the Parallax demo board and takes advantage of their *SX demo* software's UART and user interface features to allow simple access to the EEPROM contents.

Additions to the Parallax SX Demo interface

Three new commands have been added to the SX demo UART interface to access the EEPROM, as follows:

1) **S**ample:	(a) **S**	- sample the analog to digital converter ADC1 and store it in current memory address
	(b) **S xx**	- put the hex value **xx** into current memory address
2) **V**iew:	(a) **V**	- display all currently stored values
	(b) **V xx**	- display the value at hex memory address **xx**
	(c) **V FF**	- display all of the EEPROM contents
3) **E**rase:	**E**	- write zeroes to the entire EEPROM

How the circuit and program work

Thanks to the basic hardware requirements of the I^2C protocol, the circuit is very simple, using only two port pins (PortA pins 0 and 1) of the SX to provide serial access to the 24LC01 EEPROM. PortA.0 functions as the serial data clock *SCL* which provides the timing reference for data transfer to and from the EEPROM, and PortA bit 1 is *SDA*, the actual data bit stream. As on the demo board, a 10K[1] pull-up resistor should be connected from the SDA[2] pin to V_{dd} since the EEPROM's data port is open-collector.

The two main functions of the program are to read to and write from the EEPROM. Data transfers to and from the 24LC01 are composed of 8 bit data bytes which can be read/written in a random access format (i.e. one byte at a time) or in a sequential[3] format, the latter not being implemented here.

To write to the 24LC01 in random access mode, the SX must initiates the write operation by sending the EEPROM a 'START' signal, followed by a control byte 10100000b (which identifies the 24LC01 as the device to be accessed and signals that the operation to be performed is a write), followed by the address where the byte is to be written to, followed by the data byte to be written, followed by a STOP signal. It should be noted that after each byte of this sequence is sent, the program toggles the I/O status of the SDA line to read an acknowledge signal (that a byte has been received) from the EEPROM. Both the write and read sequences, as implemented here, use *acknowledge polling*. This technique sends a repeating control byte query to the EEPROM until a valid

---

[1]A value of 10K is sufficient for the data transfer rate used here. For faster rates, the pull-up may need to be reduced in order to allow successful operation. If speed is not an important issue, the external pull-up may be eliminated entirely by increasing the *t_all* bus timing delay and using the SX's internal pull-up resistor feature (see SX data sheet for programming details) on the SDA port pin.

[2]No pull-up is needed for the SCL line since it is always driven high or low by the SX

[3]The maximum number of bytes allowed during a sequential write is 8 for the 24LC01, though sequential reads have no byte count limit.

acknowledge (ACK) signal is received, before sending the address byte and then writing or reading the data byte. This is done because the EEPROM enters into an internal write cycle after each write operation, and cannot be accessed until the preceeding write process is complete, which for the 24LC01 is on the order of 10 msec. Thus by using acknowledge polling, subsequent write or read operations are executed as soon as possible after a preceeding write.

To read from the 24LC01 in random access mode, the procedure is essentially identical to the write process except that after the initial control byte and address byte have been sent and an ACK received, a new START signal is then sent followed by a read control byte (10100001b). The SDA line is then switched to an input, and data is clocked in from the EEPROM instead of sent out. The procedure is signaled as complete, as during a write, by generating a final STOP signal.

A START signal is generated by toggling the SDA line from high to low (creating a falling edge) while the SCL line is held high. A STOP signal is generated in the same manner except that SDA is toggled from low to high, thus creating a rising edge. An ACK signal is received after 8 control, address or data bits have been sent, and is considered valid if the SDA line is held low during the following (i.e. the 9th) SCL toggle cycle.

During all operations, the timing between changes in the SCL and SDA lines is a crucial factor. In this case, a generic delay time has been selected for all required START, STOP, data I/O, and ACK delays. As given, the program is capable of reading the EEPROM at approximately 200kbps[4] with the SX in turbo mode.

When calling the *I2C_write* and *I2C_read* subroutines, the program register bank must be set to the *I2C* bank. For random access mode, the address of the byte to be written/read must be pre-loaded into the *address*[5] program register, and the sequential flag *seq_flag* must be set to low. For writes, the byte to be written must be also be pre-loaded into the *data* program register, and for reads, the *data* program register will contain the value received from the EEPROM upon completion of the read procedure.

Modifications and further options

To optimize access speed to the 24LC01, the specific event and signal timings should be taken from the 24LC01 data sheet, and the appropriate reduced delay values inserted into the various bit operation subroutines. The *Bus_delay* subroutine can be accessed to produce a customized delay by loading the W register with the delay value and then calling *Bus_delay:custom*. In turbo mode each custom call will cause the following timing delay: $delay\ [usec] = 1/xtal[MHz] * (6 + 4 * (W-1))$, where *xtal* is the oscillator frequency in MHz and *W* is the value pre-loaded into the W register. For example, a value in W=62 will cause a 5 usec delay at 50 MHz.

Performing sequential writes and reads will also speed up the rate at which the 24LC01 can be accessed, and especially significantly increase the rate at which the 24LC01 can be written (since up to 8 bytes can be written simultaneously, reducing the need for separate internal EEPROM write delays).

To perform a **sequential write**, a specific series of steps must be followed. First the sequential flag *seq_flag* must be set high. The first byte to be written is then written as usual, but the following bytes (up to 7 more) are written by calling the write routine at the *I2C_write:sequential* entry point. Take note that *seq_flag* must be reset to low before the final byte of the group is sent, though the entry point called to write this final byte is still *I2C_write:sequential*. This generates the required stop bit to initiate the EEPROM write sequence.

To perform a **sequential read**, a similar series of steps must be followed. First the sequential flag *seq_flag* must be set high. The first byte to be read is read as usual, but the following bytes (up to the length of

---

[4]Since this implementation of the I2C access is coupled with a with a program that uses the SX's internal RTCC interrupt, the actual timing of the EEPROM access will vary per read/write, depending on how often interrupts occur during the read/write sequence.
[5]Take care to set the appropriate register bank, if needed.

the EEPROM[6]) are read by calling the read routine at the *I2C_read:sequential* entry point. Take note that *seq_flag* must be reset to low before the final byte of the group is read, though the entry point called to read this final byte is still *I2C_read:sequential*. This generates the required stop signal to end the sequential read operation.

After any write/read operation, the internal address pointer of the EEPROM is set to the byte following the last byte written or read. To read this next byte without using sequential mode, the program may call the read subroutine at the *I2C_read:current* entry point. This provides a slight increase in speed over the normal random access point and also eliminates the need to pre-load the *address* register before the call.

[6]In practise, the length of sequential reads can be infinite and the address pointer will simply loop around to zero after the end of the EEPROM has been reached. This can be useful for implementing wave tables and similar repeating-loop data.

Program Listing

```
;
; Device
;
             device         pins28,pages2,banks8,oschs
             device         turbo,stackx,optionx
             id             'I2C demo'
             reset          reset_entry
;
;
; Equates
;
rx_pin       =      ra.2
tx_pin       =      ra.3
led_pin      =      rb.6
spkr_pin     =      rb.7
pwm0_pin     =      rc.0
pwm1_pin     =      rc.2
adc0_out_pin =      rc.4
adc0_in_pin  =      rc.5
adc1_out_pin =      rc.6
adc1_in_pin  =      rc.7
;
;
; Variables
;
             org    8

temp         ds     1
byte         ds     1
cmd          ds     1
number_low   ds     1
number_high  ds     1
hex          ds     1
string       ds     1
;
;************************** Variables Added **************************
;
flags        EQU    0fh                         ;program flags register
;
             org    70H                         ;I2C bank
I2C          EQU    $                           ;
;
data         DS     1                           ;data byte from/for R/W
address      DS     1                           ;byte address
count        DS     1                           ;bit count for R/W
delay        DS     1                           ;timing delay for write cycle
byte_count   DS     1                           ;number of bytes in R/W
num_bytes    DS     1                           ;number of byte to view at once
save_addr    DS     1                           ;backup location for address
;
scl          EQU    RA.0                        ;I2C clock
sda          EQU    RA.1                        ;I2C data I/O
```

```
in_bit        EQU    byte.0                       ;bit to receive on I2C
out_bit       EQU    byte.7                       ;bit to transmit on I2C
seq_flag      EQU    flags.0                      ;R/W mode (if sequential=1)
got_hex       EQU    flags.1                      ;flags hex value after command
got_ack       EQU    flags.2                      ;flags if we got ack signal
erasing       EQU    flags.3                      ;high while erasing eeprom
;
control_r     =      10100001b                    ;control byte: read E2PROM
control_w     =      10100000b                    ;control byte: write E2PROM
portsetup_r   =      00000110b                    ;Port A config: read bit
portsetup_w   =      00000100b                    ;Port A config: write bit
eeprom_size   =      128                          ;storage space of EEPROM
;
t_all         =      31                           ;bit cycle delay (62=5 usec)
;***********************************************************************

              org    0


              org    10h                          ;bank0 variables

timers        =      $

timer_low     ds     1                            ;timer
timer_high    ds     1
timer_accl    ds     1
timer_acch    ds     1

freq_low      ds     1                            ;freq
freq_high     ds     1
freq_accl     ds     1
freq_acch     ds     1

              org    30h                          ;bank1 variables

analog        =      $

port_buff     ds     1                            ;buffer - used by all

pwm0          ds     1                            ;pwm0
pwm0_acc      ds     1

pwm1          ds     1                            ;pwm1
pwm1_acc      ds     1

adc0          ds     1                            ;adc0
adc0_count    ds     1
adc0_acc      ds     1

adc1          ds     1                            ;adc1
adc1_count    ds     1
adc1_acc      ds     1

              org    50h                          ;bank2 variables
```

```
serial          =       $

tx_high         ds      1                           ;tx
tx_low          ds      1
tx_count        ds      1
tx_divide       ds      1

rx_count        ds      1                           ;rx
rx_divide       ds      1
rx_byte         ds      1
rx_flag         ds      1

                org     0
;
;
; Interrupt routine - virtual peripherals
;
interrupt       bank    timers                      ;1

                clc                                 ;1     ;timer
                add     timer_accl,timer_low   ;2
                addb    timer_acch,c                ;2
                add     timer_acch,timer_high  ;2
:toggle_LED     movb    led_pin,timer_acch.7   ;4 =11

                clc                                 ;1     ;freq
                add     freq_accl,freq_low     ;2
                addb    freq_acch,c                 ;2
                add     freq_acch,freq_high    ;2
                movb    spkr_pin,freq_acch.7   ;4 =11

                bank    analog                      ;1

                clr     port_buff                   ;1

                add     pwm0_acc,pwm0               ;2     ;pwm0
                snc                                 ;1
                setb    port_buff.0                 ;1 =4

                add     pwm1_acc,pwm1               ;2     ;pwm1
                snc                                 ;1
                setb    port_buff.2                 ;1 =4

                mov     w,>>rc                      ;1     ;adc0/adc1
                not     w                           ;1     ;complement inputs to
outputs
                and     w,#%01010000                ;1
                or      port_buff,w                 ;1 =4

                mov     rc,port_buff                ;2 =2 ;update port pins

                sb      port_buff.4                 ;1     ;adc0
                inc     adc0_acc                    ;1     ;if was high, inc acc
                mov     w,adc0_acc                  ;1     ;get acc into w
                inc     adc0_count                  ;1     ;done?
```

```
                snz                              ;1    ;if so, update adc0
                mov     adc0,w                   ;1
                snz                              ;1    ;if so, reset acc
                clr     adc0_acc                 ;1 =8

                sb      port_buff.6              ;1    ;adc1
                inc     adc1_acc                 ;1    ;if was high, inc acc
                mov     w,adc1_acc               ;1    ;get acc into w
                inc     adc1_count               ;1    ;done?
                snz                              ;1    ;if so, update adc1
                mov     adc1,w                   ;1
                snz                              ;1    ;if so, reset acc
                clr     adc1_acc                 ;1 =8

                bank    serial                   ;1

                clrb    tx_divide.4              ;1    ;serial transmit
                inc     tx_divide                ;1    ;only execute every 16th
time
                mov     w,tx_divide              ;1
                and     w,#$10                   ;1
                sz                               ;1
                test    tx_count                 ;1    ;busy?
                clc                              ;1    ;ready stop bit
                sz                               ;1    ;if busy, shift bits
                rr      tx_high                  ;1
                sz                               ;1
                rr      tx_low                   ;1
                sz                               ;1    ;if busy, dec counter
                dec     tx_count                 ;1
                movb    tx_pin,/tx_low.6    ;4 =17     ;output next bit

                movb    c,rx_pin                 ;4    ;serial receive
                test    rx_count                 ;1    ;waiting for stop bit?
                jnz     :rxbit                   ;3,2  ;if not, :bit
                mov     w,#9                     ;1    ;in case start, ready 9
bits
                sc                               ;1    ;if start, set rx_count
                mov     rx_count,w               ;1
                mov     rx_divide,#16+8+1   ;2   ;ready 1.5 bit periods
:rxbit          djnz    rx_divide,:rxdone   ;3,2 ;8th time through?
                setb    rx_divide.4              ;1    ;yes, ready 1 bit period
                dec     rx_count                 ;1    ;last bit?
                sz                               ;1    ;if not, save bit
                rr      rx_byte                  ;1
                snz                              ;1    ;if so, set flag
                setb    rx_flag                  ;1 =20
:rxdone
;****Changed****
                mov     w,#-167                  ;1    ;interrupt every 164
clocks
;***************
                retiw                            ;3
;
;
```

```
; Data
;
_hello          dw      13,10,13,10,'SX Virtual Peripheral Demo'
_cr             DW      13,10,0
_prompt         dw      13,10,'>',0
_error          dw      'Error!',13,10,0
_hex            dw      '0123456789ABCDEF'
;
;*****Added*****
_space          DW      ' ',0
_sample         DW      13,10,'Sample=',0
_view           DW      13,10,'Bytes stored:',0
;***************
;
;***************
;* Subroutines *
;***************
;
;
; Get byte via serial port
;
get_byte        jnb     rx_flag,$
                clrb    rx_flag
                mov     byte,rx_byte                ;followed by send_byte
;
;
; Send byte via serial port
;
send_byte       bank    serial

:wait           test    tx_count                    ;wait for not busy
                jnz     :wait

                not     w                           ;ready bits
                mov     tx_high,w
                setb    tx_low.7

                mov     tx_count,#10                ;1 start + 8 data + 1 stop bit

                RETP                                ;leave and fix page bits
;
;
; Send hex byte (2 digits)
;
send_hex        mov     w,#13                       ;send cr lf
                call    send_byte
                mov     w,#10
                call    send_byte

:num_only       mov     w,<>number_low              ;send first digit
                call    :digit

                mov     w,number_low                ;send second digit

:digit          and     w,#$F                       ;read hex chr
```

```
                mov     temp,w
                mov     w,#_hex
                clc
                add     w,temp
                mov     m,#0
                iread
                mov     m,#$F

                jmp     send_byte                       ;send hex chr
;
;
; Send string at w
;
send_string     mov     string,w                        ;send string at w

:loop           mov     w,string                        ;read chr at w
                mov     m,#0
                iread
                mov     m,#$F

                test    w                               ;if 0, exit
                snz
                RETP                                    ;leave and fix page bits

                call    send_byte                       ;not 0, send chr

                inc     string                          ;next chr
                jmp     :loop
;
;
; Make byte uppercase
;
uppercase       csae    byte,#'a'
                ret

                sub     byte,#'a'-'A'
                RETP                                    ;leave and fix page bits
;
;
; Get hex number
;
get_hex         clr     number_low                      ;reset number
                clr     number_high
;*****Added*****
                CLRB    got_hex                         ;reset hex value flag
;***************

:loop           call    get_byte                        ;get digit
                cje     byte,#' ',:loop                 ;ignore spaces

                mov     w,<>byte                        ;get <>byte into hex
                mov     hex,w

                cjb     byte,#'0',:done                 ;if below '0', done
                cjbe    byte,#'9',:got                  ;if '0'-'9', got hex digit
```

```
              call   uppercase                     ;make byte uppercase
              cjb    byte,#'A',:done               ;if below 'A', done
              cja    byte,#'F',:done               ;if above 'F', done
              add    hex,#$90                      ;'A'-'F', adjust hex digit

:got          mov    temp,#4                       ;shift digit into number
:shift        rl     hex
              rl     number_low
              rl     number_high
              djnz   temp,:shift
;*****Added*****
              SETB   got_hex                       ;flag that we got a value
;***************
              jmp    :loop                         ;next digit

:cr           call   get_byte                      ;wait for cr
:done         cjne   byte,#13,:cr

              RETP                                 ;leave and fix page bits
;
;
;
;******************************** I2C Subroutines ****************************
;
; These routines write/read data to/from the 24LCxx EEPROM at a rate of approx.
; 200kHz. For faster* reads (up to 400 kHz max), read, write, start amd stop
; bit cycles and time between each bus access must be individually tailored
; using the CALL Bus_delay:custom entry point with appropriate values in the W
; register - in turbo mode: delay[usec] = 1/xtal[MHz] * (6 + 4 * (W-1)).
; Acknowledge polling is used to reduce delays between successive operations
; where the first of the two is a write operation. In this case, the speed
; is limited by the EEPROM's storage time.
;
; Note: These subroutines are in the 2nd memory page, so appropriate care
; should be used for accessing them ion regards to setting page select bits.
              ORG    200h
;
;
;****** Subroutine(s) : Write to I2C EEPROM
; These routines write a byte to the 24LCxxB EEPROM. Before calling this
; subroutine, the address and data registers should be loaded accordingly. The
; sequential mode flag should be clear for normal byte writing operation.
; To write in page mode, please see application note.
;
;       Input variable(s) : data, address, seq_flag
;       Output variable(s) : none
;       Variable(s) affected : byte, temp, count, delay
;       Flag(s) affected : none
;       Timing (turbo) : approx. 200 Kbps write rate
;                        approx. 10 msec between succesive writes
;
I2C_write     CALL   Set_address                   ;write address to slave
:page_mode    MOV    W,data                        ;get byte to be sent
              CALL   Write_byte                    ;Send data byte
```

```
                JB      seq_flag,:done                  ;is this a page write?
                CALL    Send_stop                       ;no, signal stop condition
:done           RETP                                    ;leave and fix page bits
;
Set_address     CALL    Send_start                      ;send start bit
                MOV     W,#control_w                    ;get write control byte
                CALL    Write_byte                      ;Write it & use ack polling
                JNB     got_ack,Set_address        ; until EEPROM ready
                MOV     W,address                       ;get EEPROM address pointer
                CALL    Write_byte                      ; and send it
                RETP                                    ;leave and fix page bits
;
Write_byte      MOV     byte,W                          ;store byte to send
                MOV     count,#8                        ;set up to write 8 bits
:next_bit       CALL    Write_bit                       ;write next bit
                RL      byte                            ;shift over to next bit
                DJNZ    count,:next_bit                 ;whole byte written yet?
                CALL    Read_bit                        ;yes, get acknowledge bit
                SETB    got_ack                         ;assume we got it
                SNB     in_bit                          ;did we get ack (low=yes)?
                CLRB    got_ack                         ;if not, flag it
;
; to use the LED as a 'no_ack' signal, the ':toggle_led' line in the interrupt
;  section must be commented out, and the next 3 instructions uncommented.
;               CLRB    led_pin                         ;default: LED off
;               SNB     in_bit                          ;did we get ack (low=yes)?
;               SETB    led_pin                         ; if not, flag it with LED
;
                RETP                                    ;leave and fix page bits
;
Write_bit       MOVB    sda,out_bit                     ;put tx bit on data line
                MOV     !ra,#portsetup_w           ;set Port A up to write
                JMP     :delay1                         ;100ns data setup delay
:delay1         JMP     :delay2                         ; (note: 250ns at low power)
:delay2         SETB    scl                             ;flip I2C clock to high
;               MOV     W,#t_high                            ;get write cycle timing*
                CALL    Bus_delay                       ;do delay while bus settles
                CLRB    scl                             ;return I2C clock low
                MOV     !ra,#portsetup_r           ;set sda->input in case ack
;               MOV     W,#t_low                        ;get clock=low cycle timing*
                CALL    Bus_delay                       ;allow for clock=low cycle
                RETP                                    ;leave and fix page bits
;
Send_start      SETB    sda                             ;pull data line high
                MOV     !ra,#portsetup_w           ;setup I2C to write bit
                JMP     :delay1                         ;100ns data setup delay
:delay1         JMP     :delay2                         ; (note: 250ns at low power)
:delay2         SETB    scl                             ;pull I2C clock high
;               MOV     W,#t_su_sta                     ;get setup cycle timing*
                CALL    Bus_delay                       ;allow start setup time
:new            CLRB    sda                             ;data line goes high->low
;               MOV     W,#t_hd_sta                     ;get start hold cycle timing*
                CALL    Bus_delay                       ;allow start hold time
                CLRB    scl                             ;pull I2C clock low
;               MOV     W,#t_buf                        ;get bus=free cycle timing*
```

```
                CALL   Bus_delay                   ;pause before next function

                RETP                               ;leave and fix page bits
;
Send_stop       CLRB   sda                         ;pull data line low
                MOV    !ra,#portsetup_w       ;setup I2C to write bit
                JMP    :delay1                     ;100ns data setup delay
:delay1         JMP    :delay2                     ; (note: 250ns at low power)
:delay2         SETB   scl                         ;pull I2C clock high
;               MOV    W,#t_su_sto                 ;get setup cycle timing*
                CALL   Bus_delay                   ;allow stop setup time
                SETB   sda                         ;data line goes low->high
;               MOV    W,#t_low                    ;get stop cycle timing*
                CALL   Bus_delay                   ;allow start/stop hold time

                RETP                               ;leave and fix page bits
;
Bus_delay       MOV    W,#t_all                    ;get timing for delay loop
:custom         MOV    temp,W                      ;save it
:loop           DJNZ   temp,:loop                  ;do delay
                RETP                               ;leave and fix page bits
;
;****** Subroutine(s) : Read from I2C EEPROM
; These routines read a byte from a 24LCXXB E2PROM either from a new address
; (random access mode), from the current address in the EEPROM's internal
; address pointer (CALL Read_byte:current), or as a sequential read. In either
; the random access or current address mode, seq_flag should be clear. Please
; refer to the application note on how to access the sequential read mode.
;
;       Input variable(s) : address, seq_flag
;       Output variable(s) : data
;       Variable(s) affected : byte, temp, count, delay
;       Flag(s) affected : none
;       Timing (turbo) : reads at approx. 200Kbps
;
I2C_read        CALL   Set_address                 ;write address to slave
:current        CALL   Send_start                  ;signal start of read
                MOV    W,#control_r                ; get read control byte
                CALL   Write_byte                  ; and send it
:sequential     MOV    count,#8                    ;set up for 8 bits
                CLR    byte                        ;zero result holder
:next_bit       RL     byte                        ;shift result for next bit
                CALL   Read_bit                    ;get next bit
                DJNZ   count,:next_bit             ;got whole byte yet?
                MOV    data,byte                   ;yes, store what was read
                SB     seq_flag                    ;is this a sequential read?
:non_seq        JMP    Send_stop                   ; no, signal stop & exit
                CLRB   out_bit                     ; yes, setup acknowledge bit
                CALL   Write_bit                   ;   and send it
                RETP                               ;leave and fix page bits
;
Read_bit        CLRB   in_bit                      ;assume input bit low
                MOV    !ra,#portsetup_r       ;set Port A up to read
                SETB   scl                         ;flip I2C clock to high
;               MOV    W,#t_high                   ;get read cycle timing*
```

```
              CALL   Bus_delay                        ;Go do delay
              SNB    sda                              ;is data line high?
              SETB   in_bit                           ;yes, switch input bit high
              CLRB   scl                              ;return I2C clock low
;             MOV    W,#t_buf                         ;get bus=free cycle timing*
              CALL   Bus_delay                        ;Go do delay
              RETP                                    ;leave and fix page bits
;
;
Take_sample   BANK   analog                           ;switch to analog bank
              MOV    W,ADC1                           ;get ADC1 value
              BANK   I2C                              ;switch to EEPROM bank
              SNB    got_hex                          ;did user enter a value?
              MOV    W,number_low                     ;yes, load it instead
              MOV    data,W                           ;save ADC1 value
              CALL   I2C_Write                        ;store it in EEPROM
              INC    address                          ;move to next address
              INC    byte_count                       ;adjust # bytes stored
              MOV    W,eeprom_size                    ;get memory size
              MOV    W,address-W                      ;are we past end?
              SNZ                                     ;if not, skip ahead
              CLR    address                          ;if so, reset it
:done         RETP                                    ;leave and fix page bits
;
View_Mem      MOV    W,byte_count                     ;get # bytes stored
:all          MOV    num_bytes,W                      ;store it into view count
              MOV    W,#_view                         ;get view message
              PAGE   send_string                      ;set up for long call
              CALL   send_string                      ;dump it
              BANK   I2C                              ;switch to EEPROM bank
              MOV    number_low,byte_count  ;get byte storage count
              PAGE   send_hex                         ;set up for long call
              CALL   send_hex:num_only      ;dump it
              BANK   I2C                              ;switch to I2C bank
              MOV    W,#0                             ;Address = start of EEPROM
              JMP    :address                         ;Go store address
:single       MOV    num_bytes,#1                     ;only a single byte
              MOV    W,number_low                     ;get the address pointer
:address      MOV    address,W                        ;store requested address
              MOV    W,#_cr                           ;get carriage return
:dump         PAGE   send_string                      ;set up for long call
              CALL   send_string                      ;send it
              BANK   I2C                              ;Switch to I2C bank
              SB     erasing                          ;viewing after erase cycle
              SNB    got_hex                          ; or special hex value?
              JMP    :viewloop                        ;yes, go dump it
              TEST   save_addr                        ;no, is EEPROM empty?
              SNZ                                     ;if not, skip ahead
              JMP    :done                            ;yes, so leave
:viewloop     CALL   I2C_read                         ;fetch byte from EEPROM
              MOV    number_low,data                  ;setup to send it
              PAGE   send_hex                         ;set up for long call
              CALL   send_hex:num_only      ;transmit it (RS232)
              BANK   I2C                              ;switch to I2C bank
              DEC    num_bytes                        ;decrement byte count
```

```
            SNZ                                 ;skip ahead if not done
            JMP     :done                       ;all bytes dumped, exit
            INC     address                     ;move to next address
            MOV     W,#00001111b                ;keep low nibble
            AND     W,address                   ; of address pointer
            MOV     W,#_space                   ;default=send a space
            SNZ                                 ;have we done 16 bytes?
            MOV     W,#_cr                      ;yes, point to a <cr>
            JMP     :dump                       ;go dump it and continue
:done       MOV     address,save_addr      ;restore address pointer
            RETP                                ;leave and fix page bits
;
Erase_Mem   CLR     address                     ;restore address pointer
            SETB    erasing                     ;flag erase operation
            MOV     num_bytes,#eeprom_size ;wipe whole mem
:wipeloop   CLR     data                        ;byte to wipe with=0
;           MOV     data,address                ;byte to wipe with=addr
            CALL    I2C_write                   ;wipe EEPROM byte
            INC     address                     ;move to next address
            DJNZ    num_bytes,:wipeloop    ;Erased enough yet?
            CLR     byte_count                  ;done, reset stored count
            CLR     save_addr                   ;reset backup address
            MOV     W,#eeprom_size              ;load mem size into W
            CALL    View_mem:all                ; and view cleared memory
            CLRB    erasing                     ;flag operation done
            RETP                                ;leave and fix page bits
;************************* End of I2C Subroutines ****************************
;
;********
;* Main *
;********
;
            ORG     140h
;
; Reset entry
;
reset_entry
;***Changed*****
            mov      ra,#%1011                  ;init ra
            mov     !ra,#%0100
            mov      rb,#%10000000              ;init rb
            mov     !rb,#%00001111
;**************

            clr     rc                          ;init rc
            mov     !rc,#%10101010
            mov     m,#$D                       ;set cmos input levels
            mov     !rc,#0
            mov     m,#$F

;****Changed****
            CLR     FSR                         ;reset all ram starting at 08h
:zero_ram   SB      FSR.4                       ;are we on low half of bank?
            SETB    FSR.3                       ;If so, don't touch regs 0-7
            CLR     IND                         ;clear using indirect addressing
```

```
              IJNZ   FSR,:zero_ram                ;repeat until done
;***************

              bank   timers                       ;set defaults
              setb   timer_low.0
              setb   freq_low.0

              mov    !option,#%10011111     ;enable rtcc interrupt
;
;
; Terminal - main loop
;
terminal      mov    w,#_hello                    ;send hello string
              call   send_string

:loop         mov    w,#_prompt                   ;send prompt string
              call   send_string

              call   get_byte                     ;get command
              call   uppercase
              mov    cmd,byte
              call   get_hex                      ;get any hex number

              cje    cmd,#'T',:timer              ;T xxxx
              cje    cmd,#'F',:freq               ;F xxxx
              cje    cmd,#'A',:pwm0               ;A xx
              cje    cmd,#'B',:pwm1               ;B xx
              cje    cmd,#'C',:adc0               ;C
              cje    cmd,#'D',:adc1               ;D

;**** Added ****
; Command: S [xx] - Store sample (if xx is left out, ADC1 is sampled)
;
              cje    cmd,#'S',:sample       ;S [xx] =store sample
;
; Command: V [xx] - View stored byte(s)
;               - if xx is left out, all stored byted are shown
;               - if xx=ff then whole eeprom is dumped
;
              cje    cmd,#'V',:view               ;V [xx] =View EEPROM contents
;
; Command: E - Erase EEPROM contents and reset storage pointer
;
              cje    cmd,#'E',:erase              ;E = Erase whole EEPROM
;***************
              mov    w,#_error                    ;bad command
              call   send_string                  ;send error string
              jmp    :loop                        ;try again

:timer        bank   timers                       ;timer write
              mov    timer_low,number_low
              mov    timer_high,number_high
              jmp    :loop

:freq         bank   timers                       ;freq write
```

```
                mov     freq_low,number_low
                mov     freq_high,number_high
                jmp     :loop

:pwm0           bank    analog                          ;pwm0 write
                mov     pwm0,number_low
                jmp     :loop

:pwm1           bank    analog                          ;pwm1 write
                mov     pwm1,number_low
                jmp     :loop

:adc0           bank    analog                          ;adc0 read
                mov     number_low,adc0
                call    send_hex
                jmp     :loop

:adc1           bank    analog                          ;adc1 read
                mov     number_low,adc1
                call    send_hex
                jmp     :loop

;**** Added ****
:sample         BANK    I2C                             ;Switch to I2C bank
                PAGE    Take_sample                     ;I2C subroutine page
                CALL    Take_sample                     ;Go take a sample
                MOV     W,#_sample                      ;get sample message
                CALL    send_string                     ;dump it
                BANK    I2C                             ;switch to EEPROM bank
                MOV     number_low,data                 ;byte sent
                CALL    send_hex:num_only          ;dump it
                JMP     :loop                           ;back to main loop
;
:view           BANK    I2C                             ;switch to I2C bank
                MOV     save_addr,address          ;backup address pointer
                SNB     got_hex                         ;Was this "V xx" command?
                JMP     :v_special                      ;if so, jump
                PAGE    View_mem                        ;I2C subroutine page
                CALL    View_mem                        ;no, view stored data
                JMP     :loop                           ;back to main loop
:v_special      MOV     W,++number_low                  ;View whole mem=> "V ff"
                JZ      :v_whole                        ;Was this requested?
                PAGE    View_mem                        ;I2C subroutine page
                CALL    View_mem:single                 ;yes, go dump it
                JMP     :loop                           ;back to main loop
:v_whole        MOV     W,#eeprom_size                  ;Get eeprom mem size
                PAGE    View_mem                        ;I2C subroutine page
                CALL    View_mem:all                    ;Go dump the whole thing
                JMP     :loop                           ;back to main loop
;
:erase          BANK    I2C                             ;switch to I2C bank
                PAGE    Erase_mem                       ;I2C subroutine page
                CALL    Erase_mem                       ;no, wipe whole EEPROM
                JMP     :loop                           ;back to main loop
;**************
```