A Simple Software I²C Routine for the COP8

National Semiconductor
Application Note 1155
Jeffrey Wright
May 2000

INTRODUCTION

This brief paper describes a simple "Master mode'' software I²C driver for the COP8 microcontroller. Two instances are presented. In the first I included a bit-filtering routine as alluded to in the Phillips spec. This improves the "robustness" of the communication channel in noisy environments. Since this added feature tends to slow things down rather markedly, a second instance is coded via conditional assembly that skips the bit-filtering altogether, doing only a quick and dirty read of the SDA pin. This version runs slightly faster, if you need the speed. The version you choose will depend entirely upon your application requirements. The routines have been successfully tested with National's LM75 and LM84 I²C temperature sensors.

DISCUSSION

Among other things, the Philips spec describes their bus as follows:

- It requires only two lines: SDA (Serial Data) and SCL (Serial Clock).
- It is multi-master — i.e., there can be more than one device on the bus that initiates a transmission.
- Every device on the bus possesses a unique address.
- Nominal 100 kbit/sec transfer rate, and up to 400 kbit/sec in "fast mode''.
- Describes "bit filtering'' within each node, serving to reject spurious noise thereby improving data integrity.
- The number of devices on the bus limited only by the maximum bus capacitance of 400 pF.

Examination of software I²C implementations reveal that most of them have "deimplemented'' the multi-master features of the Philips spec. This is simply because the overhead required to fully comply with the Philips spec (e.g., arbitration; clock synchronization; bit-filtering; etc.) would result in much slower speeds than would otherwise be achievable — and most of the time these features are just not needed. If one's application is solely a "single master'' system, most of this additional overhead is indeed unnecessary. However, because of the potential benefits of bit filtering in noisy environments, I've written this code to allow inclusion of this feature if desired.

The code comprises essentially two routines:

Send_I2C

- Transmits 'B' bytes held in the transmit buffer (I2CtxBuff)
- All transmissions begin with the 'START' pattern...
- The slave's address is sent first, the LSBit of which is the R/W indicator.
- After each byte is sent, the addressed slave is expected to 'ACK' by pulling the SDA line low.
- If slave fails to 'ACK' at any time, an error flag is set...
- Otherwise, we're done!

Read_I2C

- Prior to entry, a slave device must have been previously addressed and instructed that the master wishes to READ from it.
- Clocks in 'B' bytes from the pre-addressed slave and places them into the I2C receive buffer (I2CrxBuff).
- Following each received byte, an 'ACK' is sent, indicated by pulling the SDA line low...
- After receiving 'B' bytes, the 'Not Acknowledged' (NACK) is sent.
- Followed immediately by the 'STOP' pattern.

If one chooses to include the bit-filtering routine, it simply samples the SDA pin at equally spaced intervals and takes a majority vote as to its disposition. This adds 23 cycles, which would otherwise be unnecessary.

Source Code

Main.asm

```
.TITLE ,'Main Module, V1.0, 07.13.98'
.incld cop8sgr.inc
;.incld  cop888gg.inc
.incld  main.inc
.incld  optrex2x.inc
.incld  keys.inc
.incld  si2c.inc
; PUBLIC VARIABLES
.public Mflags, flags, REG0, REG1, REG2, REG3
; EXTERNAL PROCEDURES
.extrn  Display_Init, Keyboard_Init, I2C_Init, Service_Keyboard, MenuInit
.extrn  Menu_Selector, TaskScheduler, Service_Display, LM75_Init
.extrn  InputBuffInit
; EXTERNAL CONSTANTS
.extrn  Init_Msg:ROM, InitMPB
; EXTERNAL VARIABLES
.extrn  Key_State:B, Key_Code:B, Key_Flags:B, Key_State_Matrix:B
.extrn  MPBuffer:B, LM75_Timer:REG, Util_Flags:B
; VARIABLE DEFINITIONS
.sect variables,SEG             ; Place all vars in upper ram segment
flags:          .dsb 1          ; Main general purpose flags
Mflags:         .dsb 1          ; Main general purpose flags
Beeper_State:   .dsb 1
.endsect

;.sect stack,RAM                ; Stack is always seg0
;USERSTACK:     .dsb 20         ; 20 bytes of Stack allocation
;.endsect
;  REGISTER DEFINITIONS
.sect ScratchRegs,REG,ABS=0xF0
Scratch0:       .dsb    1       ; These are scratch and general purpose
Scratch1:       .dsb    1
Scratch2:       .dsb    1
Scratch3:       .dsb    1
.endsect

.sect MainRegs,REG
MainLoopCounter:        .dsb    1  ; Used for implementing real time delays
.endsect

.sect MAIN,ROM
;******************************************************************************
COLDSTART:
rbit    GIE,PSW                 ; Ensure interrupts are disabled
ld      SP,#topofstack          ; Init stack pointer
ld      MainLoopCounter,#0      ; Clear Main loop Timer
jsr     RTC_Init                ; Init timer1
ld      S,#01                   ; Set ram segment to user variables
sbit    TRUN,CNTRL              ; Start timer
sbit    GIE,PSW                 ; Enable all interrupts
;***** Initialize hardware interfaces
jsr     Keyboard_Init           ; Init Keyboard
jsr     Display_Init
jsr     I2C_Init                ; Init I2C interface
jsr     LM75_Init               ; Init the LM75 Temp sensor
jsr     MenuInit                ; Initiallize menu system
jsr     InputBuffInit           ; Init ASCii buffer ptrs
ld      flags,#0                ; Clear various flag regs
ld      Mflags,#0
ld      Util_Flags,#0
;******************************************************************************
; NAME: Main_Loop  Main loop in program.
;
;******************************************************************************
```

```
Main_Loop:
rbit    TimerTick,Mflags        ; Reset scheduling tick
jsr     Service_Keyboard        ; Scan Keyboard
                                ; Key thrown?
ifeq    Key_State,#PressTransitionState
jsr     Menu_Selector           ; Yes-Execute requested service
jsr     TaskScheduler           ; Execute current Task
jsr     Service_Display         ; Update Display
;        jsr     ComputeUtilization
$WaitForTick:
ifbit   TimerTick,Mflags        ; Wait for realtime to expire
jp      Main_Loop
jp      $WaitForTick
;*****************************************************************************
;*                     Interrupt Eingang
;*****************************************************************************
.sect   intr,ROM,ABS=000FF
; 7 cycles from detection of interrupt to here
push    A                       ;save A                          3
ld      A,B                     ;save B                          3
push    A                       ;                                3
ld      A,PSW                   ;save PSW                        3
push    A                       ;                                3
ld      A,X                     ;save X                          3
push    A                       ;                                3
vis                             ; vector to highest pending.     5
                                ;                               ---
                                ;                               26 cycles
;*****************************************************************************
;                      Interrupt ausgang
;*****************************************************************************
IntrX:
pop     A                       ;restore regs as stacked
x       A,X
ld      B,#PSW                  ;
pop     A                       ;
rc                              ;
ifbit   #6,A                    ;C=1? set carry
sbit    #6,[B]                  ;
ifbit   #7,A                    ;HC=1?; set half-carry
sbit    #7,[B]
pop     A                       ;
x       A,B
pop     A                       ;
reti
;*******************************
; Interrupt Service Vector Table
;*******************************
.=01E0                          ; rank - name
.addrw  Trap                    ;  16  - Default VIS
.addrw  Trap                    ;  15  - Port L/Wakeup
.addrw  Trap                    ;  14  - Timer T3B
.addrw  Trap                    ;  13  - Timer T3A/Underflow
.addrw  Trap                    ;  12  - Timer T2B
.addrw  Trap                    ;  11  - Timer T2A/Underflow
.addrw  Trap                    ;  10  - UART Transmit
.addrw  Trap                    ;   9  - UART Receive
.addrw  Trap                    ;   8  - Future TBD
.addrw  Trap                    ;   7  - Microwire/Plus Busy Low
.addrw  Trap                    ;   6  - Timer T1B reload/capture
.addrw  T1A_isr                 ;   5  - Timer T1A reload/capture
.addrw  Trap                    ;   4  - Idle timer (T0) Underflow
.addrw  Trap                    ;   3  - External G0 int
.addrw  Trap                    ;   2  - NMI
.addrw  SW_TRAP                 ;   1  - Software Interrupt
.endsect
```

```
.sect MAIN,ROM
Trap:
jmp     SW_TRAP                 ;
SW_TRAP:
rpnd                            ; These are recommended
rpnd
jp      .                       ;wait untill watchdog generates reset
;*************************************************************************
; Name: Timer_Init
;*************************************************************************
RTC_Init:
ld      CNTRL,#084              ; AutoReload mode; G0 neg edge int
ld      TMR1LO,#LOW(Main_Period); Load realtime interrupt values
ld      TMR1HI,#HIGH(Main_Period)
ld      T1RALO,#LOW(Main_Period)
ld      T1RAHI,#HIGH(Main_Period)
sbit    ENTI,PSW                ; Enable T1 interrupt
ret
;*************************************************************************
; Name: T1A_isr         Timer1 Underflow isr
;*************************************************************************
T1A_isr:                        ; 33 cycles to this point
rbit    T1PNDA,PSW              ; Reset pending flag                    4
sbit    TimerTick,Mflags        ; Is this the Rx line?                  4
ld      a,MainLoopCounter       ; Increment loop counter
inc     a
x       a,MainLoopCounter
ld      a,LM75_Timer            ; See if Other timers need adjusting
ifeq    a,#0
jp      T1A_isr_exit
dec     a
x       a,LM75_Timer
T1A_isr_exit:
jmp     IntrX                   ; get out
.endsect
.end COLDSTART
```

Display.asm

```
.TITLE ,'Display Module, V1.0, 07.13.98'
.incld cop8sgr.inc
;.incld  cop888gg.inc
.incld  main.inc
.incld  optrex2x.inc
.incld  keys.inc
.incld  si2c.inc
.incld  utility.inc
; PUBLIC PROCEDURES
.public  Display_Init, Service_Display, Display_Message
; PUBLIC VARIABLES
.public Display_Buffer, Display_State
; EXTERNALS REQ'D
.extrn  Load_Constant, Key_Msg:ROM
.extrn  KeyFlags:B, Key_Buffer:B, Key_State:B, flags:B
; VARIABLE DEFINITIONS
.sect variables,SEG             ; Place all vars in upper ram segment
Display_State:  .dsb 1
display_data:   .dsb 1
Display_Buffer: .dsb 16         ; Densitron display width
.endsect

.sect DISPLAY,ROM
;*************************************************************************
; NAME:  Service_Display
;
; PARAMETERS:   -
;
; RETURNED:
```

```
;
; REGS USED:    a,b
;
; CALLS:
;*****************************************************************************
Service_Display:
ifbit   Disp_Update,Display_State
jp      DState_0                ; Update req'd?
jp      DState_1                ; Idle state?
DState_0:
jsr     Update_Display
rbit    Disp_Update,Display_State
jp      Svc_D_Exit
DState_1:
nop
Svc_D_Exit:
ret
;*****************************************************************************
; NAME:  Display_Init    Initializes the display
;
; PARAMETERS:   -
;
; RETURNED:
;
; REGS USED:    a,b
;
; CALLS:        Display_Control
;*****************************************************************************
Display_Init:
rbit    RS,PORTD                ; Clear RS, R/W, and E
rbit    RW,PORTD
rbit    E,PORTD
ld      a,#DISPLAY_FORMAT       ; The M1641 is initialized as a Two
sbit    CMD,flags               ; Indicate this is control data
jsr     Display_Control         ; line display, dispite it being only
sbit    CMD,flags               ; Indicate this is control data
ld      a,#0E                   ; a 16X1
jsr     Display_Control         ; **Display=ON, Cursor=ON, Blink=Off
ld      a,#01                   ; **Clear display
sbit    CMD,flags               ; Indicate this is control data
jsr     Display_Control
ld      Display_State,#0        ; Reset state
ret
;*****************************************************************************
; NAME:  Display_Message
;
; PARAMETERS:   acc = #LOW(Message Label)
;                 b = offset of message from start of Message buffer
;
; RETURNED:
;
; REGS USED:    a,b
;
; CALLS:        Display_Control
;*****************************************************************************
Display_Message:
push    a                       ; Save message address
ld      a,#Display_Buffer
add     a,b                     ; Compute message address
x       a,b
pop     a
;**        jsr     Load_Constant
sbit    Disp_Update,Display_State
;**        jsr     Update_Display
ret
;*****************************************************************************
; NAME:  Update_Display  Writes contents of display buffer to display
```

```
;
; PARAMETERS:
;
; RETURNED:
;
; REGS USED:    a,b
;
; CALLS:        Display_Control
;*****************************************************************************
.set    Byte_Count,REG0
Update_Display:
ld      a,#080                  ; **Set address to 0
sbit    CMD,flags               ; Indicate this is a control byte
jsr     Display_Control
ld      b,#Display_Buffer       ; Point index at buffer
ld      Byte_Count,#8
rbit    CMD,flags               ; Now these are data bytes
DSL1:
ld      a,[b+]                  ; Load byte and increment ptr
jsr     Display_Control         ; Send to display
drsz    Byte_Count
jp      DSL1
ld      a,#0c0                  ; **Set address to position9
sbit    CMD,flags               ; Control byte
jsr     Display_Control
rbit    CMD,flags
ld      Byte_Count,#8
DSL2:
ld      a,[b+]                  ; Load byte and increment ptr
jsr     Display_Control         ; Send to display
drsz    Byte_Count
jp      DSL2
ret
;*****************************************************************************
; NAME:  Display_Control
;
; PARAMETERS:   acc = display command
;               flags.CMD = 1 if data is a control byte
;
; RETURNED:
;
; REGS USED:    a,x
;
; CALLS:
;*****************************************************************************
Display_Control:
push    a                       ; Save command
ld      a,PORTCC
and     a,#0f0                  ; Make PortC(0:3) inputs
x       a,PORTCC
ld      a,PORTLC
and     a,#0f0                  ; Make PortL(0:3) inputs
x       a,PORTLC
sbit    RW,PORTD                ; Set R/W line high
rbit    RS,PORTD                ; Ensure Instruction reg is selected
DCL1:
sbit    E,PORTD                 ; Set E line
ld      a,PORTCP                ; Read display data
rbit    E,PORTD                 ; Clr E line
ifbit   DBUSY,a                 ; Loop until busy flag=0
jp      DCL1
ld      a,PORTLC
or      a,#0f                   ; Make PortL(0:3) outputs.  These are
x       a,PORTLC                ; bits 0..3
ld      a,PORTCC
or      a,#0f                   ; Make PortC(0:3) outputs.  These are
x       a,PORTCC                ; bits 4..7
```

```
ld      a,PORTLD                ; Get lower nibble data port
and     a,#0f0                  ; Strip lower nibble
x       a,x                     ; Exchange with control byte
pop     a                       ; Retrieve command
push    a
and     a,#0f                   ; Strip upper nibble for merging with
or      a,x                     ; original data
x       a,PORTLD                ; Write lower nibble of command
ld      a,PORTCD                ; Get upper nibble data port
and     a,#0f0                  ; Strip lower nibble
x       a,x                     ; Exchange with control byte
pop     a                       ; Retrieve command
and     a,#0f0                  ; Strip lower nibble for merging with
swap    a                       ; Put upper nibble in lower nibble
or      a,x                     ; Merge with original port data
x       a,PORTCD                ; Write upper nibble of command
rbit    RW,PORTD
ifbit   CMD,flags               ; Is this a control byte?
jp      DCL2                    ; Yes - Select the instruction register
sbit    RS,PORTD                ; Clr R/W line
jp      DCL3
DCL2:
rbit    RS,PORTD
DCL3:
sbit    E,PORTD                 ; Set E line
rbit    E,PORTD                 ; Clr E line
ret
.endsect
```

fasti2c.asm

```
.TITLE ,'Fast I2C Module'
.incld  cop8sgr.inc
.incld  main.inc
.incld  FastI2
; PUBLIC VARIABLES
.public I2CRxBuff, I2CTxBuff, I2CFlags
; PUBLIC PROCEDURES
.public  I2C_Init, Send_I2C, Read_I2C, Send_I2C_Stop
; EXTERNAL PROCEDURES
; EXTERNALS REQ'D
; VARIABLE DEFINITIONS
.sect variables,SEG
I2CFlags:       .dsb    1       ; I2C control/condition flags
I2CRxBuff:      .dsb    2       ; Receive buffer
I2CTxBuff:      .dsb    4       ; Transmit buffer
.endsect

;  REGISTER DEFINITIONS
.FORM
.sect I2C,ROM
;*****************************************************************************
; Name: I2C_Init        Initializes I2C port and flags
;
; PARAMETERS:   -
;
; RETURNED:     -
;
; REGS USED:    -
;
; CALLS:        -
;*****************************************************************************
I2C_Init:
ld      a,I2CPort               ; Make SDA and SCL  HiZ inputs
and     a,#(NOT(SDAT ! SCLK))   ;
x       a,I2CPort
ld      a,I2CPTD
and     a,#(NOT(SDAT ! SCLK))   ;
```

```
x       a,I2CPTD
ld      I2CFlags,#0             ; Clear I2C flags
ret
.FORM
;*************************************************************************
; Name: Filter_Input    Samples the Rx pin three times at even intervals and
;                       then takes the majority as the bit decision. The SCL
;                       line is low upon entering and is released to its high
;                       state prior to sampling.
;                    _____________________
;                   |                     |
;    SCL   _________|                     |_____________________
;          __   ____________________________________   _____
;            \_/            x x x                  \_/
;    SDA   __/ \ ___________|_|_|_____________________ / \_____
;            3 samples  ->  | | |
;
; PARAMETERS:   SCL is in the "low" state
;               I2CFlags.STRT either set or cleared
;
; RETURNED:     Carry = rec'd bit
;
; REGS USED:    a,b
;
; CALLS:        -
;
; Exec time:    23 cycles
;*************************************************************************
Filter_Input:
rbit    SCL,I2CPort     ; Release clock line (high)
ld      b,#I2CPTP       ; Point b at Port's pin reg
clr     a               ;
nop                     ; Duty cycle equalizing nop
ifbit   SDA,[b]         ; Sample 3 times, 2 cycles apart
inc     a               ;
ifbit   SDA,[b]         ; Sample #2
inc     a               ;
ifbit   SDA,[b]         ; Sample #3
inc     a               ;
rc                      ; Reset Carry for bit decision
ifgt    a,#1            ; Majority rules
sc                      ; Set Carry if bit is a one
; Otherwise, bit is a zero
ret                     ;
.FORM
;*************************************************************************
; Name: Send_I2C        Transmits 'B' bytes held in I2CTxBuffer. After each
;                       byte is sent an "ACK" is expected from the slave
;                       device. All transmissions begin with the "START"
;       condition, followed by the device address.  The lsbit of the address
;       is the R/W bit.  Both SCL and SDA must be in the "high" state upon
;       entering.
;            ___________             ___________             ___________
;                 |      |           |     |     |           |           |
;       SCL       |      |___________|     |     |___________|           |_____
;                 |<-  ->|                 | <- Slave reads bit here
;            _____|   s  |       _________________________  _____________________
;                 |   t         /        MSB           \/
;       SDA       |__ a________/_________________________/\_____________________
;                     r        |
;                     t        | <- Master shifts out 1st bit
;
;
; PARAMETERS:   I2CTxBuff = byte(s) to be sent (always starts with address)
;               a = num bytes to xmit
;               SDA,SCL must be in high state upon entry
;
```

```
; RETURNED:     b = I2CPort
;               I2CFlags.ACKERR set if not ACK from slave
;               SDA,SCL both left in high state
;
; REGS USED:    a,b,x
;
; CALLS:        Filter_Input
;*****************************************************************************
.set    Count,REG0
.set    Byte_Count,REG1
.set    Delay,REG2
Send_I2C:
x       a,Byte_Count
rbit    GIE,PSW         ; No interrupts allowed!
ld      x,#I2CTxBuff    ; Pointer = I2CTxBuff
_Send_START_Bit:
ld      b,#I2CPort      ; Pointer = Port's control reg
sbit    SDA,[b]         ; Pulls SDA low to indicate start of xmit
_ByteWriteLoop:                 ; Min spec'd delay is 4us to SCK edge
ld      Count,#8        ; Set for 8-bit data
_BitWriteLoop:
sbit    SCL,[b]         ; Pull SCL line low
ld      a,[x]           ; Get next bit from I2CTxBuff and shift out
rlc     a               ; to SDA line.
x       a,[x]           ; Save shifted byte
ifc                     ; Bit=1?
jp      _SendOne        ; Yes - let pin pull high via Rpullup
_SendZero:
nop                     ; Path equalization nops
nop                     ;
sbit    SDA,[b]         ; No - pull SDATA pin low by configuring pin
jp      _Send01         ; as a low output
_SendOne:
rbit    SDA,[b]         ; Set SDATA pin high by making pin HiZ
nop                     ; Equalizes 1 and 0 path Delays
nop                     ;
nop                     ;
_Send01:
DELAY_3Cycles           ; Give some extra setup time, just in case
rbit    SCL,[b]         ; Let loose of clock line to set it high
DELAY_SCKHIGH           ; Equalize hi and lo times
drsz    Count           ; All 8-bits sent?
jp      _BitWriteLoop   ; No, Loop
ld      a,[x]           ; Must shift a 9th time to reset to correct
rlc     a               ; state.
x       a,[x+]          ; Bump ptr to next xmit byte
_CheckACK:                      ; After byte sent, check for "ACK" from slave
sbit    SCL,[b]         ; Pull SCL low
DELAY_SCKLOW            ; Wait for equalization
rbit    SDA,[b]         ; Release SDA line to check for ACK
jsr     Filter_Input    ; Read the input with filtering
ifc                     ; ACK should be a zero!
jp      _ACKERR         ;
rbit    ACKERR,I2CFlags ; Clear error
ld      b,#I2CPort      ; Pointer = Port's control reg
drsz    Byte_Count      ; All bytes sent?
jmp     _ByteWriteLoop  ; No, loop
jp      _Send_I2C_Exit  ; Yes, get out
_ACKERR:
sbit    ACKERR,I2CFlags ; Set error flag
_Send_I2C_Exit:
sbit    GIE,PSW         ; reEnable all interrupts
ret
.FORM
;*****************************************************************************
; Name: Send_I2C_Stop   Transmits the "Stop" sequence on the I2C bus as
;                       indicated below.
```

```
;
; PARAMETERS:   SDA and SCL must be in HIGH state upon entry
;
; RETURNED:     b = #I2CPort
;               SDA,SCL both left in high state
;
; REGS USED:    -
;
; CALLS:        -
;*****************************************************************************
.set    Delay,REG2
Send_I2C_Stop:
ld      b,#I2CPort
sbit    SCL,[b]         ; Pull SDC line to prepare for STOP
DELAY_12C               ; Delay to rising edge of SCL
sbit    SDA,[b]
DELAY_9C                ; Delay for equalization
rbit    SCL,[b]         ; Release SCL line for high
DELAY_9C                ; Delay for equalization
rbit    SDA,[b]         ; Release SDA line to indicate STOP
ret
.FORM
;*****************************************************************************
; Name: Read_I2C        Clocks in B bytes from a slave and places them into the
;                       I2CRxBuffer.  After each byte is read, an "ACK" is sent.
;       After the last byte is read a "NACK" is sent, followed by the "STOP"
;       bit. This routine assumes prior sending of the I2C device address and
;       "read" bit - i.e. - no "Start" bit is sent here.
;
;                ____________________                    ___
;              |<---     23 cyc    --->|<---    23 cyc     --->|
;    SCL   ____|   23cyc   | | |       |_____________________|
;          ________________|_|_|_____________________   _____________
;                          | | |                    \_/
;    SDA   ________________|_|_|____________________ / \_____________
;            3 samples  -> | | |
;
; PARAMETERS:   SCL is assumed to be in the low state upon entering
;               a = no. of bytes to read
;
; RETURNED:     I2CRxBuff holds slave data byte(s)
;
; REGS USED:    a,b,x
;
; CALLS:        Filter_Input
;*****************************************************************************
.set    Count,REG0
.set    Byte_Count,REG1
Read_I2C:
rbit    GIE,PSW         ; No interrupts allowed!
ld      x,#I2CRxBuff    ; Pointer = RcvBuffer
x       a,Byte_Count    ; Setup for byte loop
sbit    SCL,I2CPort     ; Pull clock line low
_ByteReadLoop:
ld      Count,#8        ; Set for 8-bit data
_BitReadLoop:
jsr     Filter_Input    ; Read the input with filtering
sbit    SCL,I2CPort     ; Toggle clock line low
ld      a,[x]           ; Get rec'd byte in progress
rlc     a               ; and rotate in the new bit
x       a,[x]           ;
drsz    Count           ; 8 bits rec'd?
jp      _BitReadLoop    ; Loop until all 8 bits are rec'd
drsz    Byte_Count      ; All bytes read?
jp      _Send_ACK_Bit   ; No, send "ACK" and loop
_Send_NACK_Bit:         ; Yes, send "NACK" and "STOP"
rbit    SDA,I2CPort     ; Release SDA line to indicate NACK
```

```
rbit    SCL,I2CPort     ; Release SCL line for high
DELAY_9C                ; Wait for end of high period
jsr     Send_I2C_Stop   ; and then send the STOP condition
jp      _ReadExit
_Send_ACK_Bit:
sbit    SDA,I2CPort     ; Yes - Pull SDA line low
rbit    SCL,I2CPort     ; Release SCL line for high
x       a,x             ; Increment Rx buffer pointer
inc     a               ;
x       a,x             ;
DELAY_9C                ; Wait for end of high period
sbit    SCL,I2CPort     ; Pull SCL line low
rbit    SDA,I2CPort     ; Release SDA line for next bit
DELAY_9C                ; Delay before looping for equalization
jmp    _ByteReadLoop    ;
_ReadExit:
sbit    RBF,I2CFlags    ; Set buffer full  flag
sbit    GIE,PSW         ; reEnable all interrupts
ret                     ;
.endsect
```

Keys.asm

```
.TITLE ,'Keys Module, V1.0, 07.13.98'
.incld cop8sgr.inc
;.incld  cop888gg.inc
.incld  main.inc
.incld  keys.inc
.incld  optrex2x.inc
.incld  si2c.inc
; PUBLIC PROCEDURES
.public Service_Keyboard, Keyboard_Init
.public Key_Flags, Key_Buffer, Key_State, Key_Code, Key_State_Matrix
; EXTERNALS REQ'D
.extrn  Key_Msg
; VARIABLE DEFINITIONS
.sect variables,SEG     ; Place all vars in upper ram segment
Key_State:              .dsb    1  ; Used for transitioning
Key_Buffer:             .dsb    1  ; Holds ASCii value of current Key
Key_Code:               .dsb    4  ; Circ buffer holding last 4 keycodes
Key_Flags:              .dsb    1  ;
Key_State_Matrix:       .dsb    4  ; Holds the prev and current states of kbd
Key_Change_Matrix:      .dsb    4  ; Holds the changes occurring in the kbd
.endsect

;
; REGISTER DEFINITIONS
.sect KeysRegs,REG
.endsect

.FORM
.sect KEYS,ROM
;*************************************************************************
; NAME:  Keyboard_Init
; PARAMETERS:   -
;
; RETURNED:
;
; REGS USED:
;
; CALLS:
;*************************************************************************
Keyboard_Init:
ld      Key_State,#0            ; Start in state #0
ld      Key_Flags,#0            ; Clear flags
ld      a,ROWS_PORT
or      a,#0f                   ; Set all row drivers
x       a,ROWS_PORT
```

```
ld      b,#Key_State_Matrix
ld      [b+],#0ff               ; Clear State pattern buffer
ld      [b+],#0ff
ld      [b+],#0ff
ld      [b],#0ff
ret
.FORM
.sect KEYSTATE,ROM,INPAGE
;*****************************************************************************
; NAME:  Sevice_Keyboard  Entry point for key module
;
; PARAMETERS:   -
;
; RETURNED:
;
; REGS USED:    a,b
;
; CALLS:        CompareKeyState, Scan_Keys
;*****************************************************************************
Service_Keyboard:
ld      a,Key_State
ifgt    a,#4                    ; Keystate is LT 4 or error
jsr     Keyboard_Init           ; Reinit if error
add     a,#LOW(Key_State_Table) ; If ok, jump to current state process
jid
Key_State_Table:
.byte  LOW(KState_0)
.byte  LOW(KState_1)
.byte  LOW(KState_2)
.byte  LOW(KState_3)
.byte  LOW(KState_4)
KState_0:                               ; *** Idle state ***
jsr     Scan_Keys               ; Read current keyboard state
jsr     CompareKeyState
ifbit   CHANGE,Key_Flags        ; Key CHANGE detected?
ld      Key_State,#Debounce_State ; Yes - adjust state variable
jp      SvcKey_Exit             ; for debounce
KState_1:                               ; *** Debounce state ***
jsr     Scan_Keys               ; Get keyboard state
jsr     CompareKeyState         ; Change in keyboard?
ifbit   CHANGE,Key_Flags        ; If last change detected has endured
jp      _KS1a                   ; debounce test, then go read change
ld      Key_State,#Key_Idle_State; If not, reset state to idling
jp      SvcKey_Exit             ; and exit
_KS1a:
ifbit   PRESS,Key_Flags         ; Is this a press??
jp      _KS1b                   ; Yes - go
ld      Key_State,#ReleaseTransitionState
jp      _KS1c                   ; Ignore releases (for now)
_KS1b:
jsr     Find_Key                ; Decode key position
ld      a,Key_Code              ; Get offset previously determined
add     a,#LOW(Key_Table)
laid                            ; Load key from ROM
x       a,Key_Buffer            ; Save in buffer
ld      Key_State,#PressTransitionState
_KS1c:
jsr     UpdateKeyState          ; Write current pattern to last pattern
jp      SvcKey_Exit
KState_2:                       ; *** Transition State ***
ld      Key_State,#Key_Idle_State
jp      SvcKey_Exit             ; buffer
KState_3:
ld      Key_State,#Key_Idle_State
jp      SvcKey_Exit
KState_4:
ld      Key_State,#Key_Idle_State
```

```
SvcKey_Exit:
ret
Key_Table:
.byte '1', '2', '3', 'A', '4', '5', '6', 'B', '7', '8', '9', 'C'
.byte 'P', '0', 'E', 'D'        ; P=Prev  E=Enter
.endsect

.FORM
;*************************************************************************
; NAME:  Scan_Keys      Scans the Keyboard by sequentially clearing each row
;                       driver and then reading and storing all four column
;                       bits into the low nibble of the associated byte of the
;       Key_State_Matrix buffer (that byte being the currently selected row).
;       This buffer comprises four (4) bytes:  One byte for each row of the
;       keyboard.  The upper nibble of each byte holds the 'previous state',
;       while the lower nibble holds the current state....
;
;                        Previous ST. Current ST.
;                        ____________ ___________
;               Row1:  C4 C3 C2 C1 C4 C3 C2 C1        Byte 0
;               Row2:  C4 C3 C2 C1 C4 C3 C2 C1        Byte 1
;               Row3:  C4 C3 C2 C1 C4 C3 C2 C1        Byte 2
;               Row4:  C4 C3 C2 C1 C4 C3 C2 C1        Byte 3
;
; PARAMETERS:   -
;
; RETURNED:
;
; REGS USED:    a,b,x, REG0
;
; CALLS:
;*************************************************************************
.set    ROWS_MASK, REG0                 ; Temporary
.set    Byte_Count, REG1                ; Temporary
Scan_Keys:
ld      b,#Key_State_Matrix     ; Point b index at key state buffer
ld      Byte_Count,#4           ; Set loop counter
ld      x,#ROWS_PORT            ; Point x index at rows port
ld      ROWS_MASK,#0fe          ; Set mask to select first row
Scan_Loop:
ld      a,[x]                   ; Read current Row pattern to enable
or      a,#0f                   ; only the one indicated by ROWS_MASK
and     a,ROWS_MASK
x       a,[x]                   ; Write new row pattern
ld      a,COLS_PORT             ; Read column pattern
and     a,#0f                   ; Strip upper nibble of Port
x       a,[b]                   ; Retrieve current/last row
and     a,#0f0                  ; Keep last info and strip low nibble
or      a,[b]                   ; Merge in new pattern and
x       a,[b+]                  ; Save in buffer - increment ptr
ld      a,ROWS_MASK             ; Now shift the mask
sc
rlc     a                       ; Shift row mask left
x       a,ROWS_MASK             ; Save next row pattern
drsz    Byte_Count
jp      Scan_Loop               ; All four rows scanned ?
ret
.FORM
;*************************************************************************
; NAME:  CompareKeyState        Determines if/where change(s) has occured in
;                               the Keyboard.  The Key_State_Matrix buffer
;       comprises four (4) bytes:  One byte for each row of the keyboard.
;       The upper nibble of each byte holds the 'previous state', while the
;       lower nibble holds the current state.  Next, the exact CHANGES in the
;       Keyboard are determined and saved in the Key_Change_Matrix. A '1' in
;       the upper nibble indicates a 'PRESS', and, conversely, a '0' indicates
;       a 'RELEASE'.  i.e....
```

```
;
;            [Key_State_Matrix]                    [Key_Change_Matrix]
;
;          Prev state  Curr state                  PRESS/REL    CHANGED
;          C  C  C  C  C  C  C  C                C  C  C  C  C  C C  C
;          o  o  o  o  o  o  o  o                o  o  o  o  o  o o  o
;          l  l  l  l  l  l  l  l                l  l  l  l  l  l l  l
;          4  3  2  1  4  3  2  1                4  3  2  1  4  3 2  1
;          ---------- ----------                ---------- ----------
; Byte1:   0  1  1  1  1  1  1  0      Row1      1  0  0  0  1  0 0  1
; Byte2:   1  1  1  1  1  1  1  1      Row2      0  0  0  0  0  0 0  0
; Byte3:   1  1  1  1  1  1  1  1      Row3      0  0  0  0  0  0 0  0
; Byte4:   1  1  1  1  1  1  1  1      Row4      0  0  0  0  0  0 0  0
;
;       For example, the above data indicates the key[11] has been pressed
;       and key[14] has been released.
;
; PARAMETERS:   -
;
; RETURNED:     -
;
; REGS USED:    a, b, x, REG0
;
; CALLS:
;*****************************************************************************
.set    TempReg, REG0                   ; Temporary
.set    Byte_Count, REG1                ; Temporary
CompareKeyState:
ld      b,#Key_State_Matrix
ld      x,#Key_Change_Matrix
ld      Byte_Count,#4           ; Do 4 rows
rbit    PRESS,Key_Flags         ; Reset press flag for debounce test
rbit    CHANGE,Key_Flags        ; Reset change flag for debounce test
CKSLoop0:
ld      a,[b]                   ; Load a row pattern
swap    a                       ; Exchange nibble order for testing
xor     a,[b]                   ; Compare states
ifeq    a,#0                    ; Any change in new pattern?
jp      _CKSa                   ; No - continue
sbit    CHANGE,Key_Flags        ; Yes - indicate change
_CKSa:
x       a,TempReg               ; Determine the nature of the changes.
ld      a,TempReg
and     a,[b]                   ; Test for PRESS/RELEASE by 'ANDing'
and     a,#0f0                  ; with the current row pattern
;*****  TEST CODE
ifgt    a,#0
sbit    PRESS,Key_Flags         ; Yes - indicate change
;*****
x       a,TempReg
and     a,#0f                   ;
or      a,TempReg               ; Merge with change(d) bits
x       a,[x]                   ; and save in change matrix
;*****
ld      a,[b+]                  ; Increment ptr to state matrix
ld      a,[x+]                  ; Increment ptr to change matrix
drsz    Byte_Count              ; Looked at all 4 rows?
jp      CKSLoop0                ; No - loop
ret
.FORM
;*****************************************************************************
; NAME:  UpdateKeyState  Updates the "LastKeyState matix with current state
;                        The 'last' and 'current' states are held the upper
;                        and lower nibbles of the same bytes.
;
; PARAMETERS:   -
;
```

```
; RETURNED:
;
;
; REGS USED:    a,b
;
; CALLS:
;*****************************************************************************
.set    Byte_Count, REG1                ; Temporary
UpdateKeyState:
ld      b,#Key_State_Matrix     ; Point indexers at buffers
ld      Byte_Count,#4           ; set up loop counter
UKSL0:
ld      a,[b]                   ; Get row from current state
swap    a                       ; put current pattern in last state
and     a,#0f0                  ; Strip old 'last state' bits
x       a,[b]                   ;
and     a,#0f                   ; Strip old 'last state' bits
or      a,[b]                   ; Merge together with current bits
x       a,[b+]                  ; and save
drsz    Byte_Count              ; Four rows complete?
jp      UKSL0                   ; No - loop
ret
.FORM
;*****************************************************************************
; NAME:  Find_Key       Determines offset for ROM table lookup of key pressed
;                       Sets error flag if not found
;
; PARAMETERS:
;
; RETURNED:     Key_Code holds Offset in num bytes for key table lookup
;
; REGS USED:    a,b
;
; CALLS:
;*****************************************************************************
.set    RowCount, REG0          ; Temporary
.set    ColumnCount, REG1       ; Temporary
Find_Key:
ld      b,#Key_Change_Matrix    ;
ld      RowCount,#4             ; Examine only 4 rows
_FKloop1:
ld      a,[b+]                  ; Examine 1st row for presses
and     a,#0f0                  ; Interested in PRESSES only
ifgt    a,#0                    ; Any key pressed in this row??
jp      _Found1inThisRow
drsz    RowCount
jp      _FKloop1
sbit    KeyError,Key_Flags      ; No rows with changes? ERROR
jp      _KSexit
_Found1inThisRow:
swap    a                       ; Must exchange here !!
x       a,b                     ; Yes, get row offset from start
;**        push    a           ; For use in mulit-key presses
                                ; Save last row address for next key
sc
subc    a,#Key_Change_Matrix+1  ; Offset of 1 req'd since ld a,[b+]
rlc     a                       ; Multiply by four (num columns)
rlc     a                       ;
and     a,#0c                   ; Strip any trash rotated in
ld      ColumnCount,#4          ; Set up for four column tests
_FKloop2:
x       a,b                     ; Xchange to test columns
rrc     a                       ; Test column bit in carry
ifc                             ; Transition detected in column?
jp      _FK1c                   ; Yes - jump
x       a,b                     ; No - try next column, and increment
inc     a                       ; key code (offset into ROM array)
```

```
drsz    ColumnCount
jp      _FKloop2
sbit    KeyError,Key_Flags      ; Can't find column?  ERROR
jp      _KSexit
_FK1c:
x       a,b                     ; This is the offset into the Key Matrix
x       a,Key_Code              ; Save Keycode for use by other routines
_KSexit:
ret
.endsect
```

LM75.asm

```
.TITLE ,'LM75 Module'
.incld  cop8sgr.inc
.incld  main.inc
.incld  LM75.inc
.incld  si2c.inc
.incld  optrex2x.inc
.incld  utility.inc
.incld  keys.inc
; PUBLIC VARIABLES
.public LM75_Timer
; PUBLIC PROCEDURES
.public  LM75_Init, Tamb_Init, Tos_Init, Thys_Init, Tcfg_Init
.public  Read_Tamb, Read_Tcfg, AdjTregVals
; EXTERNAL PROCEDURES
.extrn  Send_I2C, Read_I2C, Send_I2C_Stop
.extrn  Display_Message, Copy_Array, Int2ASC, Input_Data, Fill_Buffer
.extrn  InputBuffInit, Ascii2Sint, Load_String, Center_Field
; EXTERNALS REQ'D
.extrn  I2CFlags, I2CRxBuff, I2CTxBuff
.extrn  Display_State:B, DegreesC_Msg:ROM, ASCii_Buff:B, Key_Buffer:B
.extrn  Display_Buffer:B, Mflags:B, Util_Flags:B, Key_State:B
.extrn  EnterData_Msg
; VARIABLE DEFINITIONS
.sect variables,SEG
TcfgState:      .dsb    1       ; State variable for Tcfg task
TregState:      .dsb    1       ; State variable for Tcfg task
.endsect

;  REGISTER DEFINITIONS
.sect LM75Regs,REG
LM75_Timer:     .dsb    1       ; Used to schedule LM75 accesses
.endsect

.FORM
.sect LM75,ROM
.FORM
;*****************************************************************************
; Name: LM75_Init       Initializes LM75 related state and I2C registers
;
; PARAMETERS:   -
;
; RETURNED:     -
;
; REGS USED:    -
;
; CALLS:        -
;*****************************************************************************
LM75_Init:
ld      LM75_Timer,#0           ; Reset access timer
ld      TcfgState,#0            ; Reset state variables
ld      TregState,#0
; Load I2C Txbuffer with address of LM75
ld      I2CTxBuff,#(LM75_Address ! I2C_Read) ; Default to 'read'
ld      I2CTxBuff+1,#T_Reg                   ; Load to point to temperature reg.
ret
```

```
.FORM
;*****************************************************************************
; Name: Read_LM75T      Reads data in preselected register from addressed
;                       LM75. First, the ”read” command is sent to the LM75,
;               followed by reading B bytes of data.
;
; PARAMETERS:   I2CTxBuff[0] = Address of LM75. LSb indicates R/W
;               I2CTxBuff[1] = Pointer byte
;               a = # of bytes to read
;
; RETURNED:     Display Buffer is written with ASCII temperature data.
;               If error occured, I2CFlags.ACKERR will be set
;
; REGS USED:    -
;
; CALLS:        Send_I2C, Read_I2C
;*****************************************************************************
Read_LM75T:
ifeq    LM75_Timer,#0           ; Access allowed?
jp      _RLM75a
jp      _RLM75exit              ; Not yet, get out
_RLM75a:
ld      LM75_Timer,#10          ; This allows access every 10 Mainloops
push    a                       ; Save #bytes to read
ld      a,I2CTxBuff             ; Set the ”read” bit in command
or      a,#I2C_Read
x       a,I2CTxBuff
ld      a,#1
jsr     Send_I2C                ; Send address and read bit
pop     a                       ; #bytes to read
ifbit   ACKERR,I2CFlags         ; LM75 Acknowledged transmission?
;*****
jp      _RLM75exit
;*****
jsr     Read_I2C                ; a holds #bytes to read
_RLM75exit:
ret
.FORM
;*****************************************************************************
; Name: Write_LM75T     Writes data to selected register of addressed LM75.
;                       First, the ”write” command is sent to the LM75,
;               followed by the data to be written, whether temperature setting
;               or pointer register value.
;
; PARAMETERS:   I2CTxBuff[0] = Address of LM75. LSb indicates R/W
;               I2CTxBuff[1] = Pointer byte
;               I2CTxBuff[2] = MSbyte of set temperature
;               I2CTxBuff[3] = LSbyte of set temperature
;               a = # of bytes to send
;
; RETURNED:     If error occured, I2CFlags.ACKERR will be set
;
; REGS USED:    -
;
; CALLS:        Send_I2C, Read_I2C
;*****************************************************************************
Write_LM75T:
push    a                       ; Save #bytes to written
ld      a,I2CTxBuff             ; Set for the ”write” command
and     a,#NOT(I2C_Read)
x       a,I2CTxBuff
pop     a                       ; #bytes to read
jsr     Send_I2C                ; Send address and data
jsr     Send_I2C_Stop           ; Send STOP pattern
ifbit   ACKERR,I2CFlags         ; LM75 Acknowledged transmission?
;*****
jp      _WLM75exit
```

```
;*****
_WLM75exit:
ret
.FORM
;*************************************************************************
; NAME: Read_Tamb       Reads Ambient Temperature data from addressed LM75.
;                       The LM75's pointer register was previously set to the
;                       temperature register during the init routine.
;       The 2's compliment temperature data is read and formatted for display.
;
; PARAMETERS:
;
; RETURNED:
;
; REGS USED:    a
;
; CALLS:        Read_LM75T
;*************************************************************************
Read_Tamb:
ld      a,#2                    ; Get two bytes
jsr     Read_LM75T              ; Call the I2C routine for reading temp
ifbit   RBF,I2CFlags            ; Did we actually recv data ?
jsr     FormatTemperature       ; Format and display
rbit    RBF,I2CFlags            ; Allow next read
ret
.FORM
;*************************************************************************
; NAME: Tamb_Init       Performs Initialization code for the Read_Tamb
;                       routine. Sets the LM75's pointer to correct register
;                       for reading the ambient temp.
;
; PARAMETERS:
;
; RETURNED:
;
; REGS USED:    a
;
; CALLS:        Write_LM75T
;*************************************************************************
Tamb_Init:
ld      b,#ASCii_Buff           ; Ensure Buffer is cleared
ld      x,#ASCbuffLen           ;
ld      a,#0x20                 ; Fill with spaces for ASCII display
jsr     Fill_Buffer
ld      I2CTxBuff+1,#T_Reg      ; Point to the Temperature register
ld      a,#2                    ; Must send both addrs and ptr bytes
jsr     Write_LM75T             ; Set LM75 to correct register
ret
.FORM
;*************************************************************************
; NAME: Tos_Init        Initialization code for the Read_Tos routine. Sets
;                       the LM75's pointer register to point to the Tos reg.
;                       Then reset the Tos task's state variable.
; PARAMETERS:
;
; RETURNED:
;
; REGS USED:    a
;
; CALLS:        Write_LM75T
;*************************************************************************
Tos_Init:
ld      I2CTxBuff+1,#Tos_Reg    ; Point to the Temperature register
ld      a,#2                    ; Must send both addrs and ptr bytes
jsr     Write_LM75T             ; Set LM75 to correct register
ld      TregState,#0            ; Init state variable
ret
```

```
.FORM
;*************************************************************************
; NAME: Thys_Init       Performs any necessary Initialization code for the
;                       Read_Thys routine.
;
; PARAMETERS:
;
; RETURNED:
;
; REGS USED:    a
;
; CALLS:
;*************************************************************************
Thys_Init:
ld      I2CTxBuff+1,#Thys_Reg   ; Point to the Temperature register
ld      a,#2                    ; Must send both addrs and ptr bytes
jsr     Write_LM75T             ; Set LM75 to correct register
ld      TregState,#0            ; Init state variable
ret
.FORM
;*************************************************************************
; NAME: Tcfg_Init       Performs any necessary Initialization code for the
;                       Read_Tcfg routine.
;
; PARAMETERS:
;
; RETURNED:
;
; REGS USED:    a
;
; CALLS:
;*************************************************************************
Tcfg_Init:
ld      I2CTxBuff+1,#Tcfg_Reg   ; Point to the Temperature register
ld      a,#2                    ; Must send both addrs and ptr bytes
jsr     Write_LM75T             ; Set LM75 to correct register
ret
.FORM
;*************************************************************************
; NAME: Read_Tcfg       Reads the Configuration register from addressed LM75.
;                       The LM75's pointer register is altered to point to the
;                       Tcfg register. Next, the 2's compliment setting data
;       is read and formatted for display. Finally, the LM75's pointer register
;       is reset to point to the temperature reg.
;
; PARAMETERS:
;
; RETURNED:     I2CRxBuff[] holds two-byte temperature data.  If error occured,
;               I2CFlags.ACKERR will be set
;
; REGS USED:    -
;
; CALLS:        Read_LM75T
;*************************************************************************
Read_Tcfg:
ld      a,#1                    ; Get one byte
jsr     Read_LM75T              ; Call the I2C routine for reading temp
ret
.FORM
;*************************************************************************
; NAME: FormatTemperature       Formats the 2's compliment temperature data
;                               for display.
;
; PARAMETERS:
;
; RETURNED:     I2CRxBuff[] holds two-byte temperature data.  If error occured,
;               I2CFlags.ACKERR will be set
```

```
;
; REGS USED:    -
;
; CALLS:
;*****************************************************************************
FormatTemperature:
ld      b,#I2CRxBuff            ; ptr = buffer which holds the 2's
jsr     Int2ASC                 ; Convert 2's comp to ASCII for display
ld      x,#Display_Buffer+24+(DispLen-1)/2
ld      b,#ASCii_Buff           ; Now compute center and place
ld      a,#ASCbuffLen           ; temperature data in display
jsr     Center_Field
sbit    Disp_Update,Display_State
ret
.FORM
.sect TREGS,ROM,INPAGE
;*****************************************************************************
; NAME: AdjTregVals     Reads the Temperature data from addressed Temperature
;                       register in the addressed LM75. The LM75's pointer
;                       register must have previously been altered to point
;       to the desired register. Next, the 2's compliment temperature data
;       is read and formatted for display.
;
; PARAMETERS:
;
; RETURNED:     I2CRxBuff holds two-byte temperature data.  If error occured,
;               I2CFlags.ACKERR will be set
;
; REGS USED:    -
;
; CALLS:        Read_LM75T, Fill_Buffer, FormatTemperature, InputBuffInit
;               Load_String, Ascii2Sint, Write_LM75T
;*****************************************************************************
AdjTregVals:
ld      a,TregState             ; Vector to current state
ifgt    a,#3                    ; Check for bounds
ld      TregState,#0            ; Reinit if error
add     a,#LOW(Treg_State_Table) ; If ok, jump to current state process
jid
Treg_State_Table:
.byte  LOW(TregState0)
.byte  LOW(TregState1)
.byte  LOW(TregState2)
.byte  LOW(TregState3)
TregState0:                             ; *** Read state ***
ld      a,#2                    ; Get two bytes
jsr     Read_LM75T              ; Call the I2C routine for reading temp
jsr     FormatTemperature       ; Format and display
ld      TregState,#1            ; Next pass, go to next state
jp      RdTregRet
TregState1:
ld      b,#ASCii_Buff           ; *** Prepare for changes state ***
ld      x,#ASCbuffLen           ; Clear the Buffer for fresh entries
ld      a,#0x20                 ; Fill with spaces
jsr     Fill_Buffer
jsr     InputBuffInit           ; Init buffer hd and tail ptrs
ld      TregState,#2            ; Next pass, go to state 2
jp      RdTregRet
TregState2:                             ; *** Wait for input state ***
ifeq    Key_State,#PressTransitionState
jp      _TregS2a                ; Has choice been made?
jp      RdTregRet               ; No - return and look again next pass
_TregS2a:
ld      a,Key_Buffer            ; Get key from buffer
ifeq    a,#'E'                  ; Enter Key? Indicated desire to alter
jp      _TregS2b
jp      RdTregRet               ; No - return and look again next pass
```

```
_TregS2b:
ld      a,#LOW(EnterData_Msg)   ;  'New:            ',0
ld      x,#HIGH(EnterData_Msg)
ld      b,#Display_Buffer
jsr     Load_String
sbit    Disp_Update,Display_State
ld      TregState,#3            ; Next pass, go to state 3
RdTregRet:
ret
TregState3:                     ; *** Edit setting state ***
jsr     Input_Data              ; Get new Treg settings
ifbit   EOB,Util_Flags          ; Enter Key pressed?
jp      _TregS3a
jp      RdTregRet
_TregS3a:
rbit    EOB,Util_Flags          ; Now we must convert the ASCii data
jsr     Ascii2Sint              ; to 2's format.
x       a,I2CTxBuff+3           ; Place input data in I2C Tx buff for
x       a,b                     ; programming
x       a,I2CTxBuff+2
ld      a,#4                    ; Send address,Ptr, and data bytes
jsr     Write_LM75T
ld      TregState,#0            ; Once a value is entered, next pass
; read it out for confirmation
jp      RdTregRet
.endsect
```

message.asm

```
.TITLE ,'Message Module, V1.0, 04.15.99'
.incld  cop8sgr.inc
; PUBLIC MESSAGES
.public Init_Msg, TCfg_Msg, Tos_Msg, Thy_Msg, Sel_Msg, TSel_Msg
.public Tamb_Msg, EnterData_Msg, DevSel_Msg
;*****************************************************************************
.sect PAGE0,ROM,ABS=0x0200,INPAGE       ; Every code page begins with this
; sequence to enable checksumming and
laid                            ; to facilitate the loading of constants
ret
;         ***** Message format *****
;**    Msg_label:     .DB    '       ASCII Text      ',0 (terminator)
Init_Msg:       .DB     '   Good afternoon...    '
.db     ' ...press the any key   ', 0
DevSel_Msg:     .db     '(A) LM75  (B) LM84      '
.db     '(C) MIC   (D) KEY       ',0
Sel_Msg:        .DB     '(A) Read Temperature(s) '
.db     '(B) Configure settings  ',0
TSel_Msg:       .DB     '(A) Ambient (B) OverTemp'
.db     '(C) Hysterisis          ',0
Thy_Msg:        .DB     'Hysterisis Temperature: '
.db     '                  (#)adj',0
.endsect

;*****************************************************************************
.sect PAGE1,ROM,ABS=0x0300,INPAGE       ; Every code page begins with this
; sequence to enable checksumming and
laid                            ; to facilitate the loading of constants
ret
Tamb_Msg:       .DB     '  Ambient Temperature:  '
.db     '                        ',0
Tos_Msg:        .DB     '  OverTemp/Shutdown:    '
.db     '                  (#)adj',0
TCfg_Msg:       .DB     'Configuration Settings: '
.db     '                     Adj?',0
EnterData_Msg:  .DB     'Press # to accept (D) =.'
.db     '                        ',0
.endsect
```

optrex2x.asm

```
.TITLE ,'Display Module, V1.0, 07.13.98'
.incld cop8sgr.inc
;.incld  cop888gg.inc
.incld  main.inc
.incld  optrex2x.inc
.incld  keys.inc
.incld  si2c.inc
.incld  utility.inc
; PUBLIC PROCEDURES
.public  Display_Init, Service_Display, Display_Message, ClearDisplay
; PUBLIC VARIABLES
.public Display_Buffer, Display_State
; EXTERNALS REQ'D
.extrn  Load_Constant, Key_Msg:ROM
.extrn  KeyFlags:B, Key_Buffer:B, Key_State:B, flags:B
; VARIABLE DEFINITIONS
.sect variables,SEG             ; Place all vars in upper ram segment
Display_State:  .dsb 1
display_data:   .dsb 1
Display_Buffer: .dsb 48         ; Densitron display width
.endsect

.sect DISPLAY,ROM
;*****************************************************************************
; NAME:  Service_Display
;
; PARAMETERS:   -
;
; RETURNED:
;
; REGS USED:    a,b
;
; CALLS:
;*****************************************************************************
Service_Display:
ifbit   Disp_Update,Display_State
jp      SDState_0               ; Update requested?
jp      SvcDispExit             ; Idle state?
SDState_0:
ifbit   DispLine2,Display_State ; Which line?
jp      _ServD2
ld      a,#080                  ; **Set address to 0
sbit    DispLine2,Display_State ; Setup for line two on next pass
ld      b,#Display_Buffer       ; Point index at buffer
jp      _ServD3
_ServD2:
rbit    DispLine2,Display_State
ld      a,#0xC0                 ; **Set address to position24
rbit    Disp_Update,Display_State
ld      b,#Display_Buffer+24    ; Write line #2
_ServD3:
jsr     Update_Display
SvcDispExit:
ret
.FORM
;*****************************************************************************
; NAME:  Display_Init    Initializes the display
;
; PARAMETERS:   -
;
; RETURNED:
;
; REGS USED:    a,b
;
; CALLS:        Display_Control
```

```
;*************************************************************************
Display_Init:
rbit    RS,PORTD                ; Clear RS, R/W, and E
rbit    RW,PORTD
rbit    E,PORTD
ld      a,#DISPLAYFORMAT        ; The M1641 is initialized as a Two
sbit    CMD,flags               ; Indicate this is control data
jsr     Display_Control         ; line display, dispite it being only
sbit    CMD,flags               ; Indicate this is control data
ld      a,#0E                   ; a 16X1
jsr     Display_Control         ; **Display=ON, Cursor=ON, Blink=Off
ld      a,#CLEARDISPLAY         ; **Clear display
sbit    CMD,flags               ; Indicate this is control data
jsr     Display_Control
ld      Display_State,#0        ; Reset state
ret
;*************************************************************************
; NAME:  ClearDisplay   Sends the 'Clear display' command to the display
;
; PARAMETERS:   -
;
; RETURNED:
;
; REGS USED:    a,b
;
; CALLS:        Display_Control
;*************************************************************************
ClearDisplay:
ld      a,#CLEARDISPLAY         ; Load 'clear' command
sbit    CMD,flags               ; Indicate this is control data
jsr     Display_Control         ;
ret
.FORM
;*************************************************************************
; NAME:  Display_Message
;
; PARAMETERS:   acc = #LOW(Message Label)
;                 b = offset of message from start of Message buffer
;
; RETURNED:
;
; REGS USED:    a,b
;
; CALLS:        Display_Control
;*************************************************************************
Display_Message:
push    a                       ; Save message address
ld      a,#Display_Buffer
add     a,b                     ; Compute message address
x       a,b
pop     a
;**        jsr     Load_Constant
sbit    Disp_Update,Display_State
;**        jsr     Update_Display
ret
;*************************************************************************
; NAME:  Update_Display  Writes contents of display buffer to display
;
; PARAMETERS:   a = command byte position
;               b = Display_Buffer[0] or Display_Buffer[25]
;
; RETURNED:
;
; REGS USED:    a
;
; CALLS:        Display_Control
;*************************************************************************
```

```
.set    Byte_Count,REG0
Update_Display:
sbit    CMD,flags               ; Indicate this is a control byte
jsr     Display_Control         ; Set display position
ld      Byte_Count,#DispLen
rbit    CMD,flags               ; Now these are data bytes
DSL1:
ld      a,[b+]                  ; Load byte and increment ptr
jsr     Display_Control         ; Send to display
drsz    Byte_Count
jp      DSL1
ret
;*****************************************************************************
; NAME:  Display_Control
;
; PARAMETERS:   acc = display command
;               flags.CMD = 1 if data is a control byte
;
; RETURNED:
;
; REGS USED:    a,x
;
; CALLS:
;*****************************************************************************
Display_Control:
push    a                       ; Save command
ld      a,PORTCC
and     a,#0f0                  ; Make PortC(0:3) inputs
x       a,PORTCC
ld      a,PORTLC
and     a,#0f0                  ; Make PortL(0:3) inputs
x       a,PORTLC
sbit    RW,PORTD                ; Set R/W line high
rbit    RS,PORTD                ; Ensure Instruction reg is selected
DCL1:
sbit    E,PORTD                 ; Set E line
ld      a,PORTCP                ; Read display data
rbit    E,PORTD                 ; Clr E line
ifbit   DBUSY,a                 ; Loop until busy flag=0
jp      DCL1
ld      a,PORTLC
or      a,#0f                   ; Make PortL(0:3) outputs.  These are
x       a,PORTLC                ; bits 0..3
ld      a,PORTCC
or      a,#0f                   ; Make PortC(0:3) outputs.  These are
x       a,PORTCC                ; bits 4..7
ld      a,PORTLD                ; Get lower nibble data port
and     a,#0f0                  ; Strip lower nibble
x       a,x                     ; Exchange with control byte
pop     a                       ; Retrieve command
push    a
and     a,#0f                   ; Strip upper nibble for merging with
or      a,x                     ; original data
x       a,PORTLD                ; Write lower nibble of command
ld      a,PORTCD                ; Get upper nibble data port
and     a,#0f0                  ; Strip lower nibble
x       a,x                     ; Exchange with control byte
pop     a                       ; Retrieve command
and     a,#0f0                  ; Strip lower nibble for merging with
swap    a                       ; Put upper nibble in lower nibble
or      a,x                     ; Merge with original port data
x       a,PORTCD                ; Write upper nibble of command
rbit    RW,PORTD
ifbit   CMD,flags               ; Is this a control byte?
jp      DCL2                    ; Yes - Select the instruction register
sbit    RS,PORTD                ; Clr R/W line
jp      DCL3
```

```
DCL2:
rbit    RS,PORTD
DCL3:
sbit    E,PORTD                 ; Set E line
rbit    E,PORTD                 ; Clr E line
ret
.endsect
```

si2c.asm

```
.TITLE ,'Software I2C Module for the COP8 microcontroller'
.incld  cop8sgr.inc
.incld  main.inc
.incld  si2c.inc
; PUBLIC VARIABLES
.public I2CRxBuff, I2CTxBuff, I2CFlags
; PUBLIC PROCEDURES
.public  I2C_Init, Send_I2C, Read_I2C, Send_I2C_Stop, Filter_Input
; EXTERNAL PROCEDURES
; EXTERNALS REQ'D
; VARIABLE DEFINITIONS
.sect variables,SEG             ; These are in upper RAM, but could be in low
I2CFlags:       .dsb    1       ; I2C control/condition flags
I2CRxBuff:      .dsb    2       ; Receive buffer
I2CTxBuff:      .dsb    4       ; Transmit buffer
.endsect

;  REGISTER DEFINITIONS
.FORM
.sect I2C,ROM
;***************************************************************************
; Name: I2C_Init        Initializes I2C port and flags
;
; PARAMETERS:   -
;
; RETURNED:     -
;
; REGS USED:    -
;
; CALLS:        -
;***************************************************************************
I2C_Init:
ld      a,I2CPort               ; Make SDA and SCL  HiZ inputs
and     a,#(NOT(SDAT ! SCLK))   ;
x       a,I2CPort
ld      a,I2CPTD
and     a,#(NOT(SDAT ! SCLK))   ;
x       a,I2CPTD
ld      I2CFlags,#0             ; Clear I2C flags
ret
.FORM
;***************************************************************************
; Name: Send_I2C        Transmits 'B' bytes held in I2CTxBuffer. After each
;                       byte is sent an "ACK" is expected from the slave
;                       device. All transmissions begin with the "START"
;       condition, followed by the device address.  The lsbit of the address
;       is the R/W bit.  Both SCL and SDA must be in the "high" state upon
;       entering.  During each byte xfer, the Serial Clock is square with
;       the period equal to 38 cycles.
;            _____________           ___________           ___________
;                 |      |<- 19cyc ->|<- 19cyc ->|           |           |
;       SCL       |      |___________|     |     |___________|           |____
;                 |<-  ->|                 | <- Slave reads bit here
;            _____|   s  |       _____________________  _____________________
;                 |   t         /        MSbit        \/
;       SDA       |__ a________/_____________________/\_____________________
;                     r        |
;                     t        | <- Master shifts out MSbit
```

```
;
;
; PARAMETERS:   I2CTxBuff = byte(s) to be sent (always starts with address)
;               a = num bytes to xmit
;               SDA,SCL must be in high state upon entry
;
; RETURNED:     b = I2CPort
;               I2CFlags.ACKERR set if not ACK from slave
;               SDA,SCL both left in high state
;
; REGS USED:    a,b,x
;
; CALLS:        Filter_Input
;*****************************************************************************
.set    Count,REG0
.set    Byte_Count,REG1
.set    Delay,REG2
Send_I2C:
x       a,Byte_Count
rbit    GIE,PSW         ; No interrupts allowed!
ld      x,#I2CTxBuff    ; Pointer = I2CTxBuff
_Send_START_Bit:
ld      b,#I2CPort      ; Pointer = Port's control reg
sbit    SDA,[b]         ; Pulls SDA low to indicate start of xmit
_ByteWriteLoop:                 ; Min spec'd delay is 4us to SCK edge
ld      Count,#8        ; Set for 8-bit data
_BitWriteLoop:
sbit    SCL,[b]         ; Pull SCL line low
ld      a,[x]           ; Get next bit from I2CTxBuff and shift out
rlc     a               ; to SDA line.
x       a,[x]           ; Save shifted byte
ifc                     ; Bit=1?
jp      _SendOne        ; Yes - let pin pull high via Rpullup
_SendZero:
nop                     ; Path equalization nops
nop                     ;
sbit    SDA,[b]         ; No - pull SDATA pin low by configuring pin
jp      _Send01         ; as a low output
_SendOne:
rbit    SDA,[b]         ; Set SDATA pin high by making pin HiZ
nop                     ; Equalizes 1 and 0 path Delays
nop                     ;
nop                     ;
_Send01:
DELAY_3Cycles           ; Stretch to square up duty cycle
rbit    SCL,[b]         ; Let loose of clock line to set it high
DELAY_SCKHIGH           ; Equalize hi and lo times
drsz    Count           ; All 8-bits sent?
jp      _BitWriteLoop   ; No, Loop
ld      a,[x]           ; Must shift a 9th time to reset to correct
rlc     a               ; state.
x       a,[x+]          ; Bump ptr to next xmit byte
_CheckACK:                      ; After byte sent, check for "ACK" from slave
sbit    SCL,[b]         ; Pull SCL low
DELAY_SCKLOW            ; Wait for equalization
rbit    SDA,[b]         ; Release SDA line to check for ACK
.ifndef  FASTI2C
jsr     Filter_Input    ; Read the input with filtering
.else
rbit    SCL,I2CPort     ; Release clock line (high)
ld      a,I2CPTP        ; Get Port's pin reg
rlc     a               ; and put bit in carry for merging
.endif
ifc                     ; ACK should be a zero!
jp      _ACKERR         ;
rbit    ACKERR,I2CFlags ; Clear error
ld      b,#I2CPort      ; Pointer = Port's control reg
```

```
drsz    Byte_Count      ; All bytes sent?
jmp     _ByteWriteLoop  ; No, loop
jp      _Send_I2C_Exit  ; Yes, get out
_ACKERR:
sbit    ACKERR,I2CFlags ; Set error flag
_Send_I2C_Exit:
sbit    GIE,PSW         ; reEnable all interrupts
ret
.FORM
;*************************************************************************
; Name: Send_I2C_Stop   Transmits the "Stop" sequence on the I2C bus as
;                       indicated below.
;
; PARAMETERS:   SDA and SCL must be in HIGH state upon entry
;
; RETURNED:     b = #I2CPort
;               SDA,SCL both left in high state
;
; REGS USED:    -
;
; CALLS:        -
;*************************************************************************
.set    Delay,REG2
Send_I2C_Stop:
ld      b,#I2CPort
sbit    SCL,[b]         ; Pull SDC line to prepare for STOP
DELAY_12C               ; Delay to rising edge of SCL
sbit    SDA,[b]
DELAY_9C                ; Delay for equalization
rbit    SCL,[b]         ; Release SCL line for high
DELAY_9C                ; Delay for equalization
rbit    SDA,[b]         ; Release SDA line to indicate STOP
ret
.FORM
;*************************************************************************
; Name: Read_I2C        Clocks in B bytes from a slave and places them into the
;                       I2CRxBuffer.  After each byte is read, an "ACK" is sent.
;       After the last byte is read a "NACK" is sent, followed by the "STOP"
;       bit. This routine assumes prior sending of the I2C device address and
;       "read" bit - i.e. - no "Start" bit is sent here.  If the code is
;       assembled with the "DEF=FASTI2C" control, bit filtering is excluded,
;       yielding faster execution.
;
;               _____________________                   ___
;             |<---     23 cyc    --->|<---    22 cyc     --->|
;    SCL   ____|           | | |       |_____________________|
;          ________________|_|_|_____________________   _____________
;                          | | |                     \_/
;    SDA   ________________|_|_|_____________________ / \_____________
;            3 samples  -> | | |
;
; PARAMETERS:   SCL must be in the HIGH state upon entering
;               a = no. of bytes to read
;
; RETURNED:     I2CRxBuff holds slave data byte(s)
;
; REGS USED:    a,b,x
;
; CALLS:        Filter_Input
;*************************************************************************
.set    Count,REG0
.set    Byte_Count,REG1
Read_I2C:
rbit    GIE,PSW         ; No interrupts allowed!
ld      x,#I2CRxBuff    ; Pointer = RcvBuffer
x       a,Byte_Count    ; Setup for byte loop
sbit    SCL,I2CPort     ; Pull clock line low
```

```
_ByteReadLoop:
ld      Count,#8        ; Set for 8-bit data
_BitReadLoop:
.ifndef  FASTI2C
jsr     Filter_Input    ; Read the input with filtering
.else
rbit    SCL,I2CPort     ; Release clock line (high)
ld      a,I2CPTP        ; Get Port's pin reg
rlc     a               ; and put bit in carry for merging
.endif
sbit    SCL,I2CPort     ; Toggle clock line low
ld      a,[x]           ; Get rec'd byte in progress
rlc     a               ; and rotate in the new bit
x       a,[x]           ;
drsz    Count           ; 8 bits rec'd?
jp      _BitReadLoop    ; Loop until all 8 bits are rec'd
drsz    Byte_Count      ; All bytes read?
jp      _Send_ACK_Bit   ; No, send "ACK" and loop
_Send_NACK_Bit:                 ; Yes, send "NACK" and "STOP"
rbit    SDA,I2CPort     ; Release SDA line to indicate NACK
rbit    SCL,I2CPort     ; Release SCL line for high
DELAY_9C                ; Wait for end of high period
jsr     Send_I2C_Stop   ; and then send the STOP condition
jp      _ReadExit
_Send_ACK_Bit:
sbit    SDA,I2CPort     ; Yes - Pull SDA line low
rbit    SCL,I2CPort     ; Release SCL line for high
x       a,x             ; Increment Rx buffer pointer
inc     a               ;
x       a,x             ;
DELAY_9C                ; Wait for end of high period
sbit    SCL,I2CPort     ; Pull SCL line low
rbit    SDA,I2CPort     ; Release SDA line for next bit
DELAY_9C                ; Delay before looping for equalization
jmp    _ByteReadLoop    ;
_ReadExit:
sbit    RBF,I2CFlags    ; Set buffer full  flag
sbit    GIE,PSW         ; reEnable all interrupts
ret                     ;
.FORM
;*************************************************************************
; Name: Filter_Input    Samples the Rx pin three times at even intervals and
;                       then takes the majority as the bit decision. The SCL
;                       line is low upon entering and is released to its high
;                       state prior to sampling.
;                     ____________________
;                    |                    |
;    SCL   __________|                    |_____________________
;          __   _____________________________________   _____
;            \_/            x x x                  \_/
;    SDA   __/ \ ____________|_|_|_____________________ / \_____
;            3 samples  ->  | | |
;
; PARAMETERS:   SCL is in the "low" state
;               I2CFlags.STRT either set or cleared
;
; RETURNED:     Carry = rec'd bit
;
; REGS USED:    a,b
;
; CALLS:        -
;
; Exec time:    23 cycles
;*************************************************************************
Filter_Input:
rbit    SCL,I2CPort     ; Release clock line (high)
ld      b,#I2CPTP       ; Point b at Port's pin reg
```

```
clr     a               ;
nop                     ; Duty cycle equalizing nop
ifbit   SDA,[b]         ; Sample 3 times, 2 cycles apart
inc     a               ;
ifbit   SDA,[b]         ; Sample #2
inc     a               ;
ifbit   SDA,[b]         ; Sample #3
inc     a               ;
rc                      ; Reset Carry for bit decision
ifgt    a,#1            ; Majority rules
sc                      ; Set Carry if bit is a one
; Otherwise, bit is a zero
ret                     ;
.endsect
```

main.inc

```
; Special CONSTANTS / Bit Equates
REG0            = 0xF0          ; Used as general purpose regs
REG1            = 0xF1
REG2            = 0xF2
REG3            = 0xF3
TRUE            = 0FF
FALSE           = 00
topofstack      = 06f
TimerTick       = 0
CONV            = 1
;*************************************************************************
;                               CLOCK DEFINITION
;*************************************************************************
;one_millisec   = 1000          ; 10.0 MHZ osc yeilds 1000 ticks/ms
FOSC            = 1000          ; 10.0 MHZ osc yeilds 1000 ticks/ms
one_ms          =  FOSC
five_ms         = 5*one_ms
ten_ms          = 2*five_ms
twenty_ms       = 4*five_ms
fifty_ms        = 10*five_ms
seventy_ms      = 70*one_ms
fifty_us        = one_ms/20
onehundred_us   = one_ms/10
;Main_Period    = fifty_ms       ; Sets main loop period
Main_Period     = twenty_ms      ; Sets main loop period
;*************************************************************************
;            These are software looping reload values @ 6 cycles per loop.
;*************************************************************************
ONE_MS          =  FOSC/6
FIVE_MS         =  5*ONE_MS
TEN_MS          =  10*ONE_MS
;*************************************************************************
;            These are TMRxHI/LO reload values.
;*************************************************************************
RTC_FREQ        = 4*FOSC
```

constant.inc

```
MPB     = 0
MPBSIZE         =       15
Menu_Offset     =       0
Func_Offset     =       2
Fn1_Offset      =       4
Fn2_Offset      =       6
Fn3_Offset      =       8
InitCode_Offset =       10
Prev_Offset     =       12
Exec_Offset     =       14
```

display.inc

```
Disp_Update     =       0
```

```
DispLine2       =       1
DispLen         =       16   ; Length of Display line in chars
RS              =       4    ; Display register select bit
RW              =       5    ; Display Read/ notWrite line
E               =       6    ; Display Enable line
DBUSY           =       3    ; Display Busy line is PortC.3
CMD             =       7    ; Display Command/Data flag
SHIFT_DISPLAY   =       01C
DISPLAY_FORMAT  =       038
CNTRST_FREQ     =       TEN_MS
```

fasti2c.inc

```
;*************************************************************
;   FastI2C.INC   Include file constant and variable definition
;*************************************************************
; Bit position equates
BIT0  =  0x01
BIT1  =  0x02
BIT2  =  0x04
BIT3  =  0x08
BIT4  =  0x10
BIT5  =  0x20
BIT6  =  0x40
BIT7  =  0x80
;***** I2CFlags register bit definitions
TBF           =  0
RBF           =  1
NACK          =  2
ACK           =  3
ACKERR        =  4
;***** I2C Port assignment
I2CPort       =  PORTLC              ; Defines where the I2C port is
I2CPTD        =  PORTLD
I2CPTP        =  PORTLP
SDA           =  4                  ; L.4 and L.5
SCL           =  5
SDAT          =  BIT4               ; L.4 and L.5
SCLK          =  BIT5
;*****************************************************************************
; Delay Macros. Total delay time is (6N + 1) + M cycles, where N is the no.
;               of loops and M is any extra instructions used to add cycles
;*****************************************************************************
.MACRO  DELAY_22C
ld      Delay,#3
ld      Delay,#3
drsz    Delay
jp      .-1
.ENDM
.MACRO  DELAY_SCKHIGH
ld      Delay,#2
drsz    Delay
jp      .-1
.ENDM
.MACRO  DELAY_SCKLOW
ld      Delay,#2
drsz    Delay
jp      .-1
.ENDM
.MACRO  DELAY_15C
nop
ld      Delay,#2
drsz    Delay
jp      .-1
nop
.ENDM
.MACRO  DELAY_13C
ld      Delay,#2
```

```
drsz    Delay
jp      .-1
.ENDM
.MACRO  DELAY_12C
nop
ld      Delay,#1
ld      Delay,#1
drsz    Delay
jp      .-1
nop
.ENDM
.MACRO  DELAY_9C
nop
ld      Delay,#1
drsz    Delay
jp      .-1
nop
.ENDM
.MACRO  DELAY_3Cycles
nop
nop
nop
.ENDM
.MACRO  DELAY_2C
nop
nop
.ENDM
;*****************************************************************************
.MACRO  DELAY_T1                ; Spec requires 4us Delay
.if FOSC > 200
nop
.if FOSC > 400
nop
.if FOSC > 600
nop
.if FOSC > 800
nop
.endif
.endif
.endif
.endif
.ENDM
```

Keys.inc

```
ROWS_PORT               =       PORTD
COLS_PORT               =       PORTI
CHANGE                  =       0          ; Status indicator for keyboard
PRESS                   =       1
KeyError                =       2          ; Keyboard error flag
Key_Idle_State          =       0
Debounce_State          =       1
PressTransitionState    =       2
ReleaseTransitionState  =       3
```

optrex2x.inc

```
Disp_Update     =       0
DispLine2       =       1
DispLen         =       24   ; Length of Display line in chars
RS              =       4    ; Display register select bit
RW              =       5    ; Display Read/ notWrite line
E               =       6    ; Display Enable line
DBUSY           =       3    ; Display Busy line is PortC.3
CMD             =       7    ; Display Command/Data flag
SHIFT_DISPLAY   =       0x1C
DISPLAYFORMAT   =       0x38
CLEARDISPLAY    =       0x01
```

si2c.inc

```
;*************************************************************
;   SI2C.INC   Include file constant and variable definition
;*************************************************************
; Bit position equates
BIT0  =  0x01
BIT1  =  0x02
BIT2  =  0x04
BIT3  =  0x08
BIT4  =  0x10
BIT5  =  0x20
BIT6  =  0x40
BIT7  =  0x80
;***** I2CFlags register bit definitions
TBF           =  0
RBF           =  1
NACK          =  2
ACK           =  3
ACKERR        =  4
;***** I2C Port assignment
I2CPort       =  PORTLC             ; I2C port pins and cntrl reg
I2CPTD        =  PORTLD
I2CPTP        =  PORTLP
SDA           =  4                  ; L.4 and L.5
SCL           =  5
SDAT          =  BIT4               ; L.4 and L.5
SCLK          =  BIT5
;*************************************************************************
; Delay Macros. Total delay time is (6N + 1) + M cycles, where N is the no.
;               of loops and M is any extra instructions used to add cycles
;*************************************************************************
.MACRO  DELAY_SCKHIGH           ; 12 cycle delay
ld      Delay,#0        ; 3 cyc each
ld      Delay,#0
ld      Delay,#0
ld      Delay,#0
.ENDM
.MACRO  DELAY_SCKLOW
ld      Delay,#2
drsz    Delay
jp      .-1
.ENDM
.MACRO  DELAY_12C
nop
ld      Delay,#1
ld      Delay,#1
drsz    Delay
jp      .-1
nop
.ENDM
.MACRO  DELAY_9C
nop
ld      Delay,#1
drsz    Delay
jp      .-1
nop
.ENDM
.MACRO  DELAY_3Cycles
nop
nop
nop
.ENDM
```

Notes

LIFE SUPPORT POLICY

NATIONAL'S PRODUCTS ARE NOT AUTHORIZED FOR USE AS CRITICAL COMPONENTS IN LIFE SUPPORT DEVICES OR SYSTEMS WITHOUT THE EXPRESS WRITTEN APPROVAL OF THE PRESIDENT AND GENERAL COUNSEL OF NATIONAL SEMICONDUCTOR CORPORATION. As used herein:

1. Life support devices or systems are devices or systems which, (a) are intended for surgical implant into the body, or (b) support or sustain life, and whose failure to perform when properly used in accordance with instructions for use provided in the labeling, can be reasonably expected to result in a significant injury to the user.
2. A critical component is any component of a life support device or system whose failure to perform can be reasonably expected to cause the failure of the life support device or system, or to affect its safety or effectiveness.

National Semiconductor Corporation
Americas
Tel: 1-800-272-9959
Fax: 1-800-737-7018
Email: support@nsc.com
www.national.com

National Semiconductor Europe
Fax: +49 (0) 180-530 85 86
Email: europe.support@nsc.com
Deutsch Tel: +49 (0) 69 9508 6208
English Tel: +44 (0) 870 24 0 2171
Français Tel: +33 (0) 1 41 91 8790

National Semiconductor Asia Pacific Customer Response Group
Tel: 65-2544466
Fax: 65-2504466
Email: ap.support@nsc.com

National Semiconductor Japan Ltd.
Tel: 81-3-5639-7560
Fax: 81-3-5639-7507