Memo to: Randy Larimer
From: Matthew Handley
Date: April 15, 2014
Regarding: EE 465-01, Lab 5 – Real Time Clock and I2C

Summary:

This lab built on the previous ones, by an I2C Real Time Clock (RTC) to the system. The goal was to have the user enter a date and time with the keypad and LCD at startup, write that time to the RTC via I2C, then read back the time every few seconds from the RTC and display it on the LCD. Rather than using the built-in I2C hardware module a software bit-banging I2C module was written, based on AN1820.

Preliminary Solutions:

As before, the TPM module was used to toggle the heartbeat led. Additionally the TPM ISR handled polling the RTC for time and displaying that on the LCD, after it had been set. As shown in the block diagram of Figure 1 in Appendix A, all modules were initilized and the TPM interrupt was enabled before the RTC had been set, so that the heartbeat LED would flash while the user is setting the date and time.

Setup:

To begin programming, the DS1337 RTC was added to the breadboard along with the appropriate I2C pull-up and current limiting resistors, as shown in the Lab 5 Schematic. Because of the small and delicate nature of the 32.768 kHz crystal oscillator, it was soldered to a 2-pin 0.1 inch pitch header and covered in heat shrink tubing.

Solution:

The i2c_driver.asm file was written to implement the subroutines needed to communicate over I2C. This driver was very closely based on the sample code given in AN1820. One subroutine that was not given in AN1820 was i2c_rx_byte. This subroutine will receive a byte from the I2C bus.

The implantation of the rtc_driver.asm, which contains subroutines for writing and reading from the RTC, was fairly straightforward. This driver also contains the subroutines for prompting the user to enter a date and time as well as a subroutine for displaying the date and time on the LCD. This allows the main loop and TPM ISR to be very simple and high-level with regard to the RTC.

Summary Comments:

Having dealt with I2C before, and being given example code for implementing the I2C driver, there were no big hurdles for this lab. The final implementation worked as designed.

The following is a summary of the memory usage for this lab, as found in the project's .map file.

Flash Used: 1839 bytes
RAM Used: 98 bytes

Vectors Used:
_Vtpmovf
_Startup

Appendix A – Figures

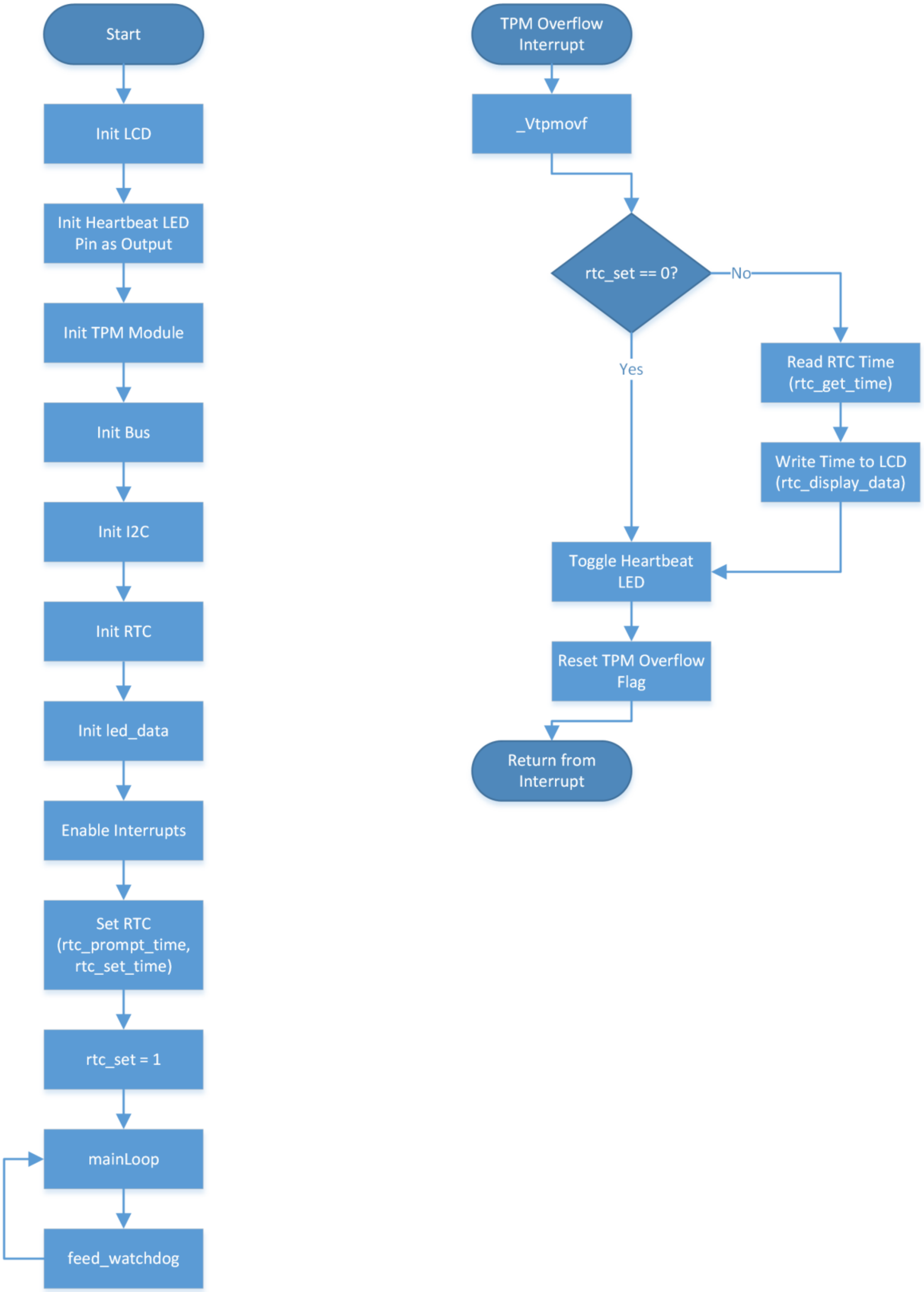

Figure 1: Top Level Flow Chart

Appendix B – Source Code

```
;*************************************************************
;* File Name       : main.asm
;* Program Name    : Lab#05 - RTC and I2C
;* Author Names    : Matthew Handley
;* Date                  : 2014-04-03
;* Description     : Prompts user for a date and time, writes that
;*                         date and time to a DS1337 RTC, then reads the
;*                         time back every 1 second and displays the time
;*                         on an LCD.
;*
;*************************************************************

; Include derivative-specific definitions
            INCLUDE 'derivative.inc'


; export symbols
            XDEF _Startup, main, _Vtpmovf, SUB_delay, SUB_delay_cnt
            ; we export both '_Startup' and 'main' as symbols. Either can
            ; be referenced in the linker .prm file or from C/C++ later on


            XREF __SEG_END_SSTACK   ; symbol defined by the linker for the end of the
stack

            XREF bus_init, bus_read, bus_write, bus_addr, bus_data

            XREF led_write, led_data

            XREF keypad_interpret, keypad_scan, keypad_get_keypress
            XREF keypad_data_0, keypad_data_1

            XREF lcd_init, lcd_write, lcd_char, lcd_str, lcd_num_to_char, lcd_clear,
lcd_goto_addr, lcd_goto_row0, lcd_goto_row1
            XREF lcd_data, lcd_char_data, lcd_col_idx

            XREF adc_init, adc_read_ch26_avg, adc_read_ch2_avg, adc_read_avg,
adc_data_0, adc_data_1

            XREF math_mul_16
            XREF INTACC1, INTACC2

            XREF i2c_init, i2c_start, i2c_stop, i2c_tx_byte, i2c_rx_byte

            XREF rtc_init, rtc_set_time, rtc_get_time, rtc_display_data,
rtc_prompt_time
            XREF Sec, Min, Hour, Date, Month, Year


; variable/data section
MY_ZEROPAGE: SECTION  SHORT

                  SUB_delay_cnt:          DS.B  3            ; counter for SUB_delay
subroutine

                  num_samples:            DS.B   1           ; number of samples to
take on the ADC

                  temp:                   DS.B  1            ; some space to hold stuff
                  temp_k:                         DS.B  1           ; some space to hold
stuff

                  rtc_set:                  DS.B  1           ; 0x01 when the rtc has
been set, 0x00 otherwise

MY_CONST: SECTION
; Constant Values and Tables Section

                  str_prompt:               DC.B  "Enter n: "
                  str_prompt_length:        DC.B  9

                  str_TK:                         DC.B  "T,K:"
                  str_TK_length:            DC.B  4
                  str_TC:                         DC.B  " T,C:"
                  str_TC_length:            DC.B  5

; code section
MyCode:     SECTION
main:
_Startup:
            LDHX   #__SEG_END_SSTACK ; initialize the stack pointer
            TXS

            ; init bus
            JSR         bus_init

            ;*** init LCD and RS, RW pins ***
                  JSR         lcd_init

                  LDA         #$00
                  STA         lcd_col_idx

                  ;*** init TPM module - for heartbeat LED ***
                  ; TPMMODH:L Registers
                  LDA   #$00
                  STA         TPMMODH
                  LDA   #$00
                  STA         TPMMODL
                  ; TPMSC Register
                  LDA   #$4E              ; TOIE clear, CLKS: Bus clock, Prescale: 128
                  STA         TPMSC

                  ;*** init led_data variable ***
                  LDA         #$00
                  STA         led_data

                  ; init i2c
                  JSR         i2c_init

                  ; init rtc
                  JSR         rtc_init
                  LDA         #$01
                  STA         rtc_set

                  CLI               ; enable interrupts

                  ; set rtc
                  JSR         rtc_prompt_time
                  JSR         rtc_set_time
                  LDA         #$00
                  STA         rtc_set
```

```
mainLoop:
                  feed_watchdog

                  BRA         mainLoop

;***************************************************************
;* Subroutine Name: _Vtpmovf
;* Description: Interrupt service routine for the TPM overflow
;*                      interrupt. Toggles the heartbeat LED (PortA[0])
;*                      and resets TPM overflow flag.
;* Registers Modified: None
;* Entry Variables: None
;* Exit Variables: None
;***************************************************************
_Vtpmovf:

                  ; check if rtc is ready
                  LDA         rtc_set
                  BNE         _Vtpmovf_heartbeat

                  ; read rtc time
                  JSR         rtc_get_time

                  ; display RTC time on LCD
                  JSR         rtc_display_data

                  ; update heatbeat led
                  JSR         led_write

_Vtpmovf_heartbeat:

                  ; Toggle Heartbeat LED
                  LDA         led_data                ; load current LED pattern
                  EOR         #$80                    ; toggle bit 7
                  STA         led_data                ; Store pattern to var

                  ; clear TPM ch0 flag
                  LDA         TPMSC                   ; read register
                  AND         #$4E                    ; clear CH0F bit, but leav
others alone
                  STA         TPMSC                   ; write back register

                  ; Done, Return from Interrupt
                  RTI


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

```
;**************************************************************
;* Subroutine Name: SUB_delay
;* Description: Decrements SUB_delay_cnt until it reaches zero.
;*                      1 count in SUB_delay_cnt is approx 4.019 us
;*
;* Registers Modified: None.
;* Entry Variables: SUB_delay_cnt - 3 byte variable, determines length
;*                            of time the SUB_delay routine will take to execute.
;* Exit Variables: SUB_delay_cnt - will be zero at exit.
;**************************************************************
SUB_delay:
                  ; save the existing values of registers
                  PSHH
                  PSHX
                  PSHA

                  ; load address of SUB_delay_cnt
                  LDHX #SUB_delay_cnt

SUB_delay_loop_0:

                  feed_watchdog

                  ; if byte[0] == 0
                  LDA   0, X
                  BEQ         SUB_delay_loop_1        ; jump to SUB_delay_outer_loop

                  ;else
                  DECA                                      ; decrement byte[0]
                  STA         0, X

                  ;repeat
                  BRA SUB_delay_loop_0

SUB_delay_loop_1:

                  ; if byte[1] == 0
                  LDA   1, X
                  BEQ         SUB_delay_loop_2        ; branch to done

                  ;else
                  DECA                                      ; decrement byte[1]
                  STA         1, X

                  LDA         #$FF                          ; reset byte[0]
                  STA         0,X

                  ;repeat
                  BRA SUB_delay_loop_0

SUB_delay_loop_2:

                  ; if byte[2] == 0
                  LDA   2, X
                  BEQ         SUB_delay_done                ; branch to done

                  ;else
                  DECA                                      ; decrement byte[2]
                  STA         2, X

                  LDA         #$FF                          ; reset byte[1]
                  STA         1, X
```

```
                   LDA         #$FF                           ; reset byte[0]
                   STA         0, X

                   ;repeat
                   BRA SUB_delay_loop_0

SUB_delay_done:

                   ; restore registers to previous values
                   PULA
                   PULX
                   PULH

                   RTS
;***************************************************************
```

```
;**************************************************************
;* File Name    :       bus.asm
;* Author Names :       Matthew Handley
;* Date         :       2014-03-04
;* Description  :       Contains subroutines for controlling the
;*                            bus.
;*
;**************************************************************

; EQU statements
mDataBus          EQU   $F0         ; Mask for the data bus pins on PortB
mAddrBus          EQU   $0F         ; Mask for the address bus pins on PortB


; Include derivative-specific definitions
            INCLUDE 'MC9S08QG8.inc'

; export symbols
            XDEF bus_init, bus_write, bus_read, bus_addr, bus_data

; import symbols
                  XREF _Startup, main, _Vtpmov


; variable/data section
MY_ZEROPAGE: SECTION  SHORT

                  bus_addr:         DS.B  1     ; only use lower 3 bits
                  bus_data:         DS.B  1     ; only use lower 4 bits

; code section
MyCode:     SECTION

;**************************************************************
;* Subroutine Name: bus_init
;* Description: Reads data from the device whose address is
;*                      the lower 3 bits of bus_addr, and store the
;*                      data to the lower 4 bits of bus_data.
;*
;* Registers Modified: None
;* Entry Variables: None
;* Exit Variables: None
;**************************************************************
bus_init:
                  ; preserve registers
                  PSHA

                  ;*** init Data & Address Busses ***
                  LDA         mAddrBus                      ; Set Address Bus pins as
output by default, leave data as input
                  STA         PTBDD
                  LDA         $00                                 ; Leave all of PortB
as input at start
                  STA         PTBD

                  ; restore registers
                  PULA

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

```
;**************************************************************
;* Subroutine Name: bus_read
;* Description: Reads data from the device whose address is
;*                      the lower 3 bits of bus_addr, and store the
;*                      data to the lower 4 bits of bus_data.
;*
;* Registers Modified: None
;* Entry Variables: bus_addr
;* Exit Variables: bus_data
;**************************************************************
bus_read:
                  ; preserve accumulator A
                  PSHA

                  ; make address bus output, data bus an input
            LDA         #mAddrBus
            STA         PTBDD

                  ; pull the address low
            LDA   bus_addr                ; load address
            AND         #$07                    ; mask off the lower 3 bits to be
sure, will leave G2A low
            STA         PTBD                    ; write data to address bus, and clear
data bus

            ; read data from the bus
            LDA         PTBD
            NSA                                 ; shift data down to the lower 4 bits
            AND         #$0F                    ; mask off the lower 4 bits to be sure
            STA         bus_data                ;

                  ; pull the address high
            LDA   #$08                    ; G2A_not high
            STA         PTBD                    ; write, clears address bus

                  ; restore accumulator A
                  PULA

                  ; return from subroutine bus_read
                  RTS

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

```
;**************************************************************
;* Subroutine Name: bus_write
;* Description: Writes the lower 4 bits of bus_data to the
;*                      device on whose address is the lower 3 bits
;*                      of bus_addr.
;* Registers Modified: None
;* Entry Variables: bus_addr, bus_data
;* Exit Variables: None
;**************************************************************
bus_write:
                  ; preserve accumulator A
                  PSHA

                  ; make data and address busses outputs
            LDA         #$FF
            STA         PTBDD

            ; prep data for the bus
            LDA         bus_data
            NSA                                     ; swap the lower 4 bits to be the
upper 4 bits
            AND         #$F0                        ; mask off the upper 4 bits to be sure

                  ; prep the addr, G2A_not low, Yx goes low
            ORA         bus_addr                    ; add in the address
            STA         PTBD                        ; write data and address bus, with
G2A_not low

            ORA         #$08                        ; leave data and address, set G2A_not
high - Yx goes high
            STA         PTBD

                  ; restore accumulator A
                  PULA

                  ; return from subroutine bus_write
                  RTS

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

```
;**************************************************************
;* File Name    :       i2c_driver.asm
;* Author Names :       Matthew Handley
;* Date         :       2014-03-25
;* Description  :       Contains subroutines for a bit-banging
;*                            software I2C driver, based on AN1820.
;*
;**************************************************************

; EQU statements
SCL         EQU 3             ;Serial clock bit number
SDA         EQU 2             ;Serial data bit number

; Include derivative-specific definitions
            INCLUDE 'MC9S08QG8.inc'

; export symbols
            XDEF i2c_init, i2c_start, i2c_stop, i2c_tx_byte, i2c_rx_byte
            ;XDEF

; import symbols
                  ;XREF


; variable/data section
MY_ZEROPAGE: SECTION  SHORT

                  BitCounter:       DS.B  1           ; Used to count bits in a Tx
                  Value:                  DS.B  1           ; Used to store rx data
value

; code section
MyCode:     SECTION

;**************************************************************
;* Subroutine Name: i2c_init
;* Description: Initilizes the software I2C driver.
;*
;* Registers Modified: None
;* Entry Variables: None
;* Exit Variables: None
;**************************************************************
i2c_init:
                  ;Initialize variables
                  CLR   Value                  ;Clear all RAM variables
                  CLR   BitCounter

                  ;*** init SDA and SCL pins as outputs
                  BSET  SDA, PTADD
                  BSET  SCL, PTADD

                  ;*** init SDA and SCL pins to high
                  BSET  SDA, PTAD
                  BSET  SCL, PTAD

                  RTS

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

```
;**************************************************************
;* Subroutine Name: i2c_start
;* Description: Generate a START condition on the bus.
;*
;* Registers Modified: None
;* Entry Variables: None
;* Exit Variables: None
;**************************************************************
i2c_start:
                  ; crate falling edge on SDA while SCL high
                  BCLR  SDA, PTAD
                  JSR   i2c_bit_delay
                  BCLR  SCL, PTAD
                  RTS

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

;**************************************************************
;* Subroutine Name: i2c_stop
;* Description: Generate a STOP condition on the bus.
;*
;* Registers Modified: None
;* Entry Variables: None
;* Exit Variables: None
;**************************************************************
i2c_stop:
                  ; crate rising edge on SDA while SCL high
                  BCLR  SDA, PTAD
                  BSET  SCL, PTAD
                  BSET  SDA, PTAD
                  JSR   i2c_bit_delay
                  RTS

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

```
;**************************************************************
;* Subroutine Name: i2c_tx_byte
;* Description: Transmit the byte in Acc to the SDA pin
;*                      (Acc will not be restored on return)
;*
;*                      Must be careful to change SDA values only
;*                      while SCL is low, otherwise a STOP or START
;*                      could be implied.
;*
;* Registers Modified: A, X
;* Entry Variables: None
;* Exit Variables: None
;**************************************************************
i2c_tx_byte:
                  ;Initialize variable
                  LDX   #$08
                  STX   BitCounter

tx_nextbit:
                  ROLA                                ; Shift MSB into Carry
                  BCC         tx_send_low             ; Send low bit or high bit

tx_send_high:
                  BSET  SDA, PTAD               ; set the data bit value
                  JSR         i2c_setup_delay         ; Give some time for data

tx_setup:
                  BSET  SCL, PTAD               ; clock in data
                  JSR         i2c_bit_delay           ; wait a bit
                  BRA         tx_continue             ; continue

tx_send_low:
                  BCLR  SDA, PTAD               ; set the data bit value
                  JSR         i2c_setup_delay         ; Give some time for data
                  BRA         tx_setup                      ; clock in the bit

tx_continue:
                  BCLR  SCL, PTAD               ; Restore clock to low state
                  DEC         BitCounter              ; Decrement the bit counter
                  BEQ         tx_ack_poll             ; Last bit?
                  BRA         tx_nextbit              ; Do the next bit

tx_ack_poll:
                  BSET  SDA, PTAD
                  BCLR  SDA, PTADD              ; Set SDA as input
                  JSR   i2c_setup_delay         ; wait

                  BSET  SCL, PTAD               ; clock the line
                  JSR         i2c_bit_delay           ; wait

                  BRCLR SDA, PTAD, tx_done      ; check SDA for ack

tx_no_ack:
                  ; do error handling here

tx_done:
                  BCLR  SCL, PTAD               ; restore the clock line
                  BSET  SDA, PTADD              ; SDA back to output
                  RTS                                       ; done
;**************************************************************
```

```
;**************************************************************
;* Subroutine Name: i2c_rx_byte
;* Description: Recieves a byte from the I2C bus.
;*                      Will Ack the byte if Accu A != 0
;*                      Data returned in Accu A
;*
;* Registers Modified: A, X
;* Entry Variables: None
;* Exit Variables: None
;**************************************************************
i2c_rx_byte:

                  ; clear output var
                  CLR         Value

                  ; set BitCounter
                  LDX   #$08
                  STX   BitCounter

                  ; set SDA to input and pull clock low
                  BCLR  SDA, PTADD
                  BCLR  SCL, PTAD

rx_nextbit:
                  ; wait for a bit
                  JSR         i2c_bit_delay

                  ; shift the last bit recieved left (and fill LSB with zero)
                  LSL         Value

                  ; clock the line and wait
                  BSET  SCL, PTAD
                  JSR         i2c_setup_delay

                  ; grab bit from bus
                  BRCLR SDA, PTAD, rx_low

rx_high:
                  ; store a 1 to Value
                  BSET  0, Value
                  BRA         rx_continue

rx_low:
                  ; do nothing since LSL fills with 0

rx_continue:
                  BCLR  SCL, PTAD               ; Restore clock to low state
                  DEC         BitCounter              ; Decrement the bit counter
                  BNE         rx_nextbit              ; More bits?


                  ; set SDA back to output
                  BSET  SDA, PTADD

                  ; test Accu A == 0
                  CBEQA #$00, rx_nack
                  BRA         rx_ack

rx_ack:
                  ; clear data bit to acknowledge
                  BCLR  SDA, PTAD
                  BRA         rx_done
```

```
rx_nack:
                  ; set data bit to not acknowledge
                  BSET  SDA, PTAD

rx_done:
                  ; let ack/nack settle
                  JSR         i2c_setup_delay

                  ;clock the ack/nack
                  BSET  SCL, PTAD
                  JSR         i2c_bit_delay

                  ; retun clock to low
                  BCLR  SCL, PTAD

                  ; load Value into Accu A
                  LDA         Value

                  RTS


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

;***************************************************************
;* Subroutine Name: i2c_setup_delay
;* Description: Provide some data setup time to allow
;*                      SDA to stabilize in slave device
;*                      Completely arbitrary delay (10 cycles?)
;*
;* Registers Modified: None
;* Entry Variables: None
;* Exit Variables: None
;***************************************************************
i2c_setup_delay:

                  NOP
                  NOP
                  RTS

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

;***************************************************************
;* Subroutine Name: i2c_setup_delay
;* Description: Bit delay to provide (approximately) the desired
;*                      SCL frequency
;*                      Again, this is arbitrary (16 cycles?)
;*
;* Registers Modified: None
;* Entry Variables: None
;* Exit Variables: None
;***************************************************************
i2c_bit_delay:

                  NOP
                  NOP
                  NOP
                  NOP
                  NOP
                  RTS

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

```
;**************************************************************
;* File Name    :       keypad.asm
;* Author Names :       Matthew Handley
;* Date         :       2014-03-04
;* Description  :       Contains subroutines for reading the
;*                            keypad.
;*
;**************************************************************

; EQU statements


; Include derivative-specific definitions
            INCLUDE 'derivative.inc'

; export symbols
            XDEF keypad_interpret, keypad_scan, keypad_data_0, keypad_data_1,
keypad_data_0_old, keypad_data_1, keypad_data_cmp, keypad_get_keypress

; import symbols
                  XREF bus_read, bus_write, bus_addr, bus_data
                  XREF led_write, led_data
                  XREF lcd_char


; variable/data section
MY_ZEROPAGE: SECTION  SHORT

                  keypad_data_0:    DS.B  1     ; bit flags representing what keys are
pressed on they 4x4 keypad
                  keypad_data_1:    DS.B  1

                  keypad_data_0_old:      DS.B  1     ; bit flags representing which
keys were pressed on the keypad, the last time it was scanned
                  keypad_data_1_old:      DS.B  1

                  keypad_data_cmp:  DS.B  1     ; tempory holder for keypad data
comparison in keypad_interpret

; code section
MyCode:     SECTION



;**************************************************************
;* Subroutine Name: keypad_scan
;* Description: Scans the greyhill 4x4 keypad, and saves the
;*                      result to variable.
;*                      Note that this method will overwrite values in
;*                      the bus_addr and bus_data variables.
;*
;* Registers Modified: None
;* Entry Variables: None
;* Exit Variables: keypad_data_0, keypad_data_1
;**************************************************************
keypad_scan:
                  ; preserve registers
                  PSHA

;*** save old value of keypad_data, before we overwrite it

                  LDA         keypad_data_0
```

```
                STA         keypad_data_0_old
                LDA         keypad_data_1
                STA         keypad_data_1_old

;*** scan row 0 ***

            ;* set row 0 to low, other rows to high *

            ; set address of keypad driver DFF
            LDA   #$02
            STA         bus_addr

            ; set the data
            LDA   #%00001110
            STA         bus_data

            ; write the data
            JSR         bus_write

            ;* read data from row *

                  ; set the address
            LDA   #$03
            STA         bus_addr

            ; read the data
            JSR         bus_read

            ;* save row data to variable *

                LDA         bus_data          ; load in data nibble
                COMA                          ; compliment bits, so 1=button press
                AND         #$0F              ; mask off the lower 4 bits
                STA         keypad_data_0     ; store to vairable


;*** scan row 1 ***

            ;* set row 1 to low, other rows to high *

            ; set address of keypad driver DFF
            LDA   #$02
            STA         bus_addr

            ; set the data
            LDA   #%00001101
            STA         bus_data

            ; write the data
            JSR         bus_write

            ;* read data from row *

                  ; set the address
            LDA   #$03
            STA         bus_addr

            ; read the data
            JSR         bus_read

            ;* save row data to variable *
```

```
                  LDA         bus_data           ; load in data nibble
                  COMA                           ; compliment bits, so 1=button press
                  NSA                                  ; swap our data to the upper
nibble
                  AND         #$F0               ; mask off the data
                  ORA         keypad_data_0      ; add the lower 4 bits in
                  STA         keypad_data_0      ; store to vairable


;*** scan row 2 ***

            ;* set row 2 to low, other rows to high *

            ; set address of keypad driver DFF
            LDA   #$02
            STA         bus_addr

            ; set the data
            LDA   #%00001011
            STA         bus_data

            ; write the data
            JSR         bus_write

            ;* read data from row *

                  ; set the address
            LDA   #$03
            STA         bus_addr

            ; read the data
            JSR         bus_read

            ;* save row data to variable *

                  LDA         bus_data           ; load in data nibble
                  COMA                           ; compliment bits, so 1=button press
                  AND         #$0F               ; mask off the lower 4 bits
                  STA         keypad_data_1      ; store to vairable


;*** scan row 3 ***

            ;* set row 3 to low, other rows to high *

            ; set address of keypad driver DFF
            LDA   #$02
            STA         bus_addr

            ; set the data
            LDA   #%00000111
            STA         bus_data

            ; write the data
            JSR         bus_write

            ;* read data from row *

                  ; set the address
            LDA   #$03
            STA         bus_addr
            ; read the data
            JSR         bus_read

            ;* save row data to variable *

                  LDA         bus_data           ; load in data nibble
                  COMA                           ; compliment bits, so 1=button press
                  NSA                                  ; swap our data to the upper
nibble
                  AND         #$F0               ; mask off the data
                  ORA         keypad_data_1      ; add the lower 4 bits in
                  STA         keypad_data_1      ; store to vairable


;*** done ***

                  ; restore registers
                  PULA

                  ; return from subroutine keypad_scan
                  RTS

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

```
;**************************************************************
;* Subroutine Name: keypad_interpret
;* Description: Checks if a numeric key (1..9) was pressed.
;*                      When a key is pressed, it writes it to the LCD
;*                      and returns the numeric value in Accu A.
;*                      Returns 0xFF when (1..9) was not pressed.
;*
;*
;* Registers Modified: None
;* Entry Variables: None
;* Exit Variables: Accu A
;**************************************************************
keypad_interpret:

;*** was a key pressed in the first 2 rows ? ***

                  LDA         keypad_data_0_old
                  COMA
                  AND         keypad_data_0
                  CBEQA #$00, keypad_interpret_lower_rows_jump

                  ; key was pressed
                  STA         keypad_data_cmp

keypad_interpret_1:

                  ; was '1' pressed ?
                  LDA         keypad_data_cmp
                  AND         #%11111110
                  BNE         keypad_interpret_2

                  ; write a '1' to the LCD
                  LDA         #'1'
                  JSR         lcd_char

                  ; return 0x01
                  LDA         #$01
                  RTS


keypad_interpret_2:

                  ; was '2' pressed ?
                  LDA         keypad_data_cmp
                  AND         #%11111101
                  BNE         keypad_interpret_3

                  ; write a '2' to the LCD
                  LDA         #'2'
                  JSR         lcd_char

                  ; return 0x02
                  LDA         #$02
                  RTS

keypad_interpret_3:

                  ; was '3' pressed ?
                  LDA         keypad_data_cmp
                  AND         #%11111011
                  BNE         keypad_interpret_A
                  ; write a '3' to the LCD
                  LDA         #'3'
                  JSR         lcd_char

                  ; return 0x03
                  LDA         #$03
                  RTS


keypad_interpret_A:

                  ; was 'A' pressed ?
                  LDA         keypad_data_cmp
                  AND         #%11110111
                  BNE         keypad_interpret_4

                  ; write a 'A' to the LCD
                  LDA         #'A'
                  JSR         lcd_char

                  ; return 0x0A
                  LDA         #$0A
                  RTS

                  BRA         keypad_interpret_4
keypad_interpret_lower_rows_jump:
                  BRA         keypad_interpret_lower_rows


keypad_interpret_4:

                  ; was '4' pressed ?
                  LDA         keypad_data_cmp
                  AND         #%11101111
                  BNE         keypad_interpret_5

                  ; write a '4' to the LCD
                  LDA         #'4'
                  JSR         lcd_char

                  ; return 0x04
                  LDA         #$04
                  RTS


keypad_interpret_5:

                  ; was '5' pressed ?
                  LDA         keypad_data_cmp
                  AND         #%11011111
                  BNE         keypad_interpret_6

                  ; write a '5' to the LCD
                  LDA         #'5'
                  JSR         lcd_char

                  ; return 0x05
                  LDA         #$05
                  RTS


keypad_interpret_6:
```

```
                  ; was '6' pressed ?
                  LDA         keypad_data_cmp
                  AND         #%10111111
                  BNE         keypad_interpret_B

                  ; write a '6' to the LCD
                  LDA         #'6'
                  JSR         lcd_char

                  ; return 0x06
                  LDA         #$06
                  RTS


keypad_interpret_B:

                  ; was 'B' pressed ?
                  LDA         keypad_data_cmp
                  AND         #%01111111
                  BNE         keypad_interpret_lower_rows

                  ; write a 'B' to the LCD
                  LDA         #'B'
                  ;JSR        lcd_char

                  ; return 0x0B
                  LDA         #$0B
                  RTS



keypad_interpret_lower_rows:
;*** was a key pressed in the second 2 rows ? ***

                  LDA         keypad_data_1_old
                  COMA
                  AND         keypad_data_1
                  CBEQA #$00, keypad_interpret_done_jump

                  ; key was pressed
                  STA         keypad_data_cmp


keypad_interpret_7:

                  ; was '7' pressed ?
                  LDA         keypad_data_cmp
                  AND         #%11111110
                  BNE         keypad_interpret_8

                  ; write a '7' to the LCD
                  LDA         #'7'
                  JSR         lcd_char

                  ; return 0x07
                  LDA         #$07
                  RTS


keypad_interpret_8:
```

```
                  ; was '8' pressed ?
                  LDA         keypad_data_cmp
                  AND         #%11111101
                  BNE         keypad_interpret_9

                  ; write a '8' to the LCD
                  LDA         #'8'
                  JSR         lcd_char

                  ; return 0x08
                  LDA         #$08
                  RTS


keypad_interpret_9:

                  ; was '9' pressed ?
                  LDA         keypad_data_cmp
                  AND         #%11111011
                  BNE         keypad_interpret_C

                  ; write a '9' to the LCD
                  LDA         #'9'
                  JSR         lcd_char

                  ; return 0x09
                  LDA         #$09
                  RTS


keypad_interpret_C:

                  ; was 'C' pressed ?
                  LDA         keypad_data_cmp
                  AND         #%11110111
                  BNE         keypad_interpret_E

                  ; write a 'C' to the LCD
                  LDA         #'C'
                  ;JSR        lcd_char

                  ; return 0x0C
                  LDA         #$0C
                  RTS


                  BRA   keypad_interpret_E
keypad_interpret_done_jump:
                  BRA         keypad_interpret_done


keypad_interpret_E:

                  ; was 'E'/'*' pressed ?
                  LDA         keypad_data_cmp
                  AND         #%11101111
                  BNE         keypad_interpret_0

                  ; write a 'E' to the LCD
                  LDA         #'E'
                  JSR         lcd_char
                  ; return 0x0E
                  LDA         #$0E
                  RTS


keypad_interpret_0:

                  ; was '0' pressed ?
                  LDA         keypad_data_cmp
                  AND         #%11011111
                  BNE         keypad_interpret_F

                  ; write a '0' to the LCD
                  LDA         #'0'
                  JSR         lcd_char

                  ; return 0x00
                  LDA         #$00
                  RTS


keypad_interpret_F:

                  ; was 'F'/'#' pressed ?
                  LDA         keypad_data_cmp
                  AND         #%10111111
                  BNE         keypad_interpret_D

                  ; write a 'F' to the LCD
                  LDA         #'F'
                  ;JSR        lcd_char

                  ; return 0x00
                  LDA         #$00
                  RTS


keypad_interpret_D:

                  ; was 'D' pressed ?
                  LDA         keypad_data_cmp
                  AND         #%01111111
                  BNE         keypad_interpret_done

                  ; write a 'D' to the LCD
                  LDA         #'D'
                  JSR         lcd_char

                  ; return 0x0D
                  LDA         #$0D
                  RTS



keypad_interpret_done:
;*** done ***

                  ; return $FF to indicate no key pressed
                  LDA         #$FF
                  RTS

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

```
;**************************************************************
;* Subroutine Name: keypad_get_keypress
;* Description: Continously scans and interprets the keypad
;*                      until a key is pressed.
;*
;*
;* Registers Modified: None
;* Entry Variables: None
;* Exit Variables: Accu A
;**************************************************************
keypad_get_keypress:

                  ; feed watchdog
                  feed_watchdog

                  ; update heatbeat led
                  JSR         led_write

                  ; scan the keypad
                  JSR         keypad_scan

                  ; check for keypress
                  JSR         keypad_interpret

                  ; if no key pressed, repeat
                  CBEQA #$FF, keypad_get_keypress

                  ; key was pressed, so we're done
                  RTS


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

```
;**************************************************************
;* File Name    :       lcd.asm
;* Author Names :       Matthew Handley
;* Date         :       2014-03-04
;* Description  :       Contains subroutines for controlling the
;*                            lcd.
;*
;**************************************************************

; EQU statements


; Include derivative-specific definitions
            INCLUDE 'MC9S08QG8.inc'

; export symbols
            XDEF lcd_init, lcd_write, lcd_char, lcd_str, lcd_num_to_char, lcd_clear,
lcd_goto_addr, lcd_goto_row0, lcd_goto_row1
            XDEF lcd_data, lcd_char_data, lcd_col_idx

; import symbols
                  XREF SUB_delay, SUB_delay_cnt
                  XREF bus_read, bus_write, bus_addr, bus_data


; variable/data section
MY_ZEROPAGE: SECTION  SHORT

                  lcd_data:         DS.B  1     ; lower 4 bits = LCD data lines, bit 6
= RS, bit 5 = RW
                  lcd_char_data:    DS.B  1     ; used by lcd_char subroutine to store
a character
                  lcd_col_idx:      DS.B  1     ; index of the column of the LCD that
the cursor is currently in

                  lcd_addr:         DS.B  1     ; holds an address for lcd_goto_addr

                  str_length:       DS.B  1     ; holds the offset into a string for
lcd_str

; code section
MyCode:     SECTION

;**************************************************************
;* Subroutine Name: lcd_init
;* Description: Initilizes the LCD.
;*
;* Registers Modified: None
;* Entry Variables: None
;* Exit Variables: None
;**************************************************************
lcd_init:
                  ; preserve registers
                  PSHA

;*** init RS and RW pins as outputs
                  LDA         PTADD
                  ORA         #$03
                  STA         PTADD

;*** wait for 15 ms
                  ; load address of SUB_delay_cnt
                  LDHX #SUB_delay_cnt

                  ; configure loop delays: 0x001388 = 20 ms
                  LDA         #$00
                  STA         2,X
                  LDA         #$13
                  STA         1,X
                  LDA         #$88
                  STA         0,X

                  ; jump to the delay loop
                  JSR         SUB_delay


;*** Send Init Command

                  LDA         #$03
                  JSR         lcd_write

;*** Wait for 4.1 ms
                  ; load address of SUB_delay_cnt
                  LDHX #SUB_delay_cnt

                  ; configure loop delays: 0x001388 = 20 ms
                  LDA         #$00
                  STA         2,X
                  LDA         #$13
                  STA         1,X
                  LDA         #$88
                  STA         0,X

                  ; jump to the delay loop
                  JSR         SUB_delay


;*** Send Init command

                  LDA         #$03
                  JSR         lcd_write


;*** Wait for 100 us
                  ; load address of SUB_delay_cnt
                  LDHX #SUB_delay_cnt

                  ; configure loop delays: 0x001388 = 20 ms
                  LDA         #$00
                  STA         2,X
                  LDA         #$13
                  STA         1,X
                  LDA         #$88
                  STA         0,X

                  ; jump to the delay loop
                  JSR         SUB_delay


;*** Send Init command

                  LDA         #$03
                  JSR         lcd_write
```

```
;*** Send Function set command

                  LDA         #$02
                  JSR         lcd_write

                  LDA         #$02
                  JSR         lcd_write

                  LDA         #$08
                  JSR         lcd_write ; goes blank here


;*** Send display ctrl command

                  LDA         #$00
                  JSR         lcd_write

                  LDA         #$0F
                  JSR         lcd_write

;*** Send display clear command

                  LDA         #$00
                  JSR         lcd_write

;*** Wait for 5 ms
                  ; load address of SUB_delay_cnt
                  LDHX #SUB_delay_cnt

                  ; configure loop delays: 0x001388 = 20 ms
                  LDA         #$00
                  STA         2,X
                  LDA         #$13
                  STA         1,X
                  LDA         #$88
                  STA         0,X

                  ; jump to the delay loop
                  JSR         SUB_delay

;*** Send display clear command

                  LDA         #$01
                  JSR         lcd_write

;*** Wait for 5 ms
                  ; load address of SUB_delay_cnt
                  LDHX #SUB_delay_cnt

                  ; configure loop delays: 0x001388 = 20 ms
                  LDA         #$00
                  STA         2,X
                  LDA         #$13
                  STA         1,X
                  LDA         #$88
                  STA         0,X

                  ; jump to the delay loop
                  JSR         SUB_delay


;*** Send entry mode command
```

```
                  LDA         #$00
                  JSR         lcd_write

                  LDA         #$06
                  JSR         lcd_write

;*** done ***

                  ; restore registers
                  PULA

                  ; return from subroutine lcd_init
                  RTS

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

```
;**************************************************************
;* Subroutine Name: lcd_write
;* Description: Sends data to the LCD.
;*
;* Registers Modified: Accu A
;* Entry Variables: Accu A
;* Exit Variables:
;**************************************************************
lcd_write:
                  ; preserve HX register
                  PSHH
                  PSHX

                  ; store param to var for latter
                  STA         lcd_data

                  ; clear RS and RW pins on PTAD
                  LDA   PTAD
                  AND         #$FC
                  STA         PTAD

                  ; put RS an RW on PTAD
                  LDA         lcd_data
                  NSA
                  AND   #$03
                  ORA         PTAD
                  STA         PTAD

                  ; prep bus data
                  LDA         lcd_data
                  AND         #$0F
                  STA         bus_data
                  ; prep bus addr
                  LDA         #$04
                  STA         bus_addr
                  ; write data to bus (and clock the addr)
                  JSR         bus_write


;*** Wait for 40 us
                  ; load address of SUB_delay_cnt
                  LDHX #SUB_delay_cnt

                  ; configure loop delays: 0x00000A = 40 us
                  LDA         #$00
                  STA         2,X
                  LDA         #$00
                  STA         1,X
                  LDA         #$0A
                  STA         0,X

                  ; jump to the delay loop
                  JSR         SUB_delay

                  ; restore HX register
                  PULX
                  PULH

                  ; done
                  RTS

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

```
;***************************************************************
;* Subroutine Name: lcd_char
;* Description: Writes a character to the LCD.
;*                      If lcd_col_idx is off of the first line, the
;*                      LCD will be cleared and the new char will be
;*                      written to the first column of row 0
;*
;* Registers Modified: Accu A
;* Entry Variables: Accu A
;* Exit Variables:
;***************************************************************
lcd_char:
                  ; preserve registers
                  PSHH
                  PSHX

                  ; save data
                  STA         lcd_char_data

                  ; write upper nibble
                  NSA
                  AND         #$0F
                  ORA         #$20
                  JSR         lcd_write

                  ; write lower nibble
                  LDA         lcd_char_data
                  AND         #$0F
                  ORA         #$20
                  JSR         lcd_write

;*** Wait for 1 ms ***
                  LDHX #SUB_delay_cnt

                  ; configure loop delays: 0x0000FA = 1 ms
                  LDA         #$00
                  STA         2,X
                  LDA         #$00
                  STA         1,X
                  LDA         #$FA
                  STA         0,X

                  ; jump to the delay loop
                  JSR         SUB_delay

                  ; done
                  PULX
                  PULH
                  RTS


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

```
;**************************************************************
;* Subroutine Name: lcd_str
;* Description: Writes a 0x00 terminated string of bytes to
;*                      the lcd, starting at the address in the HX
;*                      register. Does not keep track of location
;*                      on lcd.
;*
;* Registers Modified: Accu A, HX
;* Entry Variables: HX, A
;* Exit Variables:  none
;**************************************************************
lcd_str:
                  ; save str length
                  STA          str_length

lcd_str_loop:
                  ; get data
                  LDA          0,X

                  ; write data to lcd
                  JSR          lcd_char

                  ; increament lower byte X
                  PSHX
                  PULA
                  ADD          #01
                  PSHA
                  PULX

                  ; increment upper byte H
                  PSHH
                  PULA
                  ADC          #$00
                  PSHA
                  PULH

                  ; decrement str_length
                  LDA          str_length
                  DECA
                  STA          str_length

                  ; repeat if length != 0
                  BNE          lcd_str_loop

lcd_str_done:

                  RTS

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

```
;**************************************************************
;* Subroutine Name: lcd_num_to_char
;* Description: Takes a number in Accu A and converts it to the
;*                      ASCII representation of that number. Only works
;*                      for lower for bits of Accu A.
;*
;* Registers Modified: None.
;* Entry Variables: Accu A
;* Exit Variables: Accu A
;**************************************************************
lcd_num_to_char:
                  ; Add 0x30
                  ADD          #$30
                  RTS

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


;**************************************************************
;* Subroutine Name: lcd_clear
;* Description: Sends the clear command to the lcd and waits
;*                      for it to clear (20 ms).
;*
;* Registers Modified: A, HX
;* Entry Variables: None
;* Exit Variables: None
;**************************************************************
lcd_clear:

;*** Wait for 20 ms ***
                  LDHX #SUB_delay_cnt

                  ; configure loop delays: 0x001388 = 20 ms
                  LDA          #$00
                  STA          2,X
                  LDA          #$13
                  STA          1,X
                  LDA          #$88
                  STA          0,X

                  ; jump to the delay loop
                  JSR          SUB_delay

                  ; Send display clear command
                  LDA          #$00
                  JSR          lcd_write
                  LDA          #$01
                  JSR          lcd_write

;*** Wait for 20 ms ***
                  LDHX #SUB_delay_cnt

                  ; configure loop delays: 0x001388 = 20 ms
                  LDA          #$00
                  STA          2,X
                  LDA          #$13
                  STA          1,X
                  LDA          #$88
                  STA          0,X

                  ; jump to the delay loop
                  JSR          SUB_delay
                  ; done
                  RTS

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

;**************************************************************
;* Subroutine Name: lcd_goto_addr
;* Description: Commands the LCD to put the cursor at the
;*                      location given in Accu A.
;*
;* Registers Modified: A, HX
;* Entry Variables: None
;* Exit Variables: None
;**************************************************************
lcd_goto_addr:

                  ; store addr
                  STA         lcd_addr

                  ; write upper nibble
                  NSA
                  AND         #$0F
                  JSR         lcd_write

                  ; write lower nibble
                  LDA         lcd_addr
                  AND         #$0F
                  JSR         lcd_write

                  RTS

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


;**************************************************************
;* Subroutine Name: lcd_goto_row0
;* Description: Commands the LCD to put the cursor at colum 0
;*                      of row 0.
;*
;* Registers Modified: A, HX
;* Entry Variables: None
;* Exit Variables: None
;**************************************************************
lcd_goto_row0:

                  ; go back to first column and row of LCD
                  LDA         #$08
                  JSR         lcd_write
                  LDA         #$00
                  JSR         lcd_write

                  RTS

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

```
;**************************************************************
;* Subroutine Name: lcd_goto_row1
;* Description: Commands the LCD to put the cursor at colum 0
;*                      of row 1.
;*
;* Registers Modified: A, HX
;* Entry Variables: None
;* Exit Variables: None
;**************************************************************
lcd_goto_row1:

                  ; go back to first column and row of LCD
                  LDA         #$0C
                  JSR         lcd_write
                  LDA         #$00
                  JSR         lcd_write

                  RTS

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

```
;***************************************************************
;* File Name    :       led.asm
;* Author Names :       Matthew Handley
;* Date         :       2014-03-04
;* Description  :       Contains subroutines for controlling the
;*                            DFF-driven LEDs.
;*
;***************************************************************

; EQU statements


; Include derivative-specific definitions
            INCLUDE 'MC9S08QG8.inc'

; export symbols
            XDEF led_write, led_data

; import symbols
                  XREF bus_read, bus_write, bus_addr, bus_data


; variable/data section
MY_ZEROPAGE: SECTION  SHORT

                  led_data:         DS.B  1     ; 8 bit value for the 8 LEDs

; code section
MyCode:     SECTION
```

```
;**************************************************************
;* Subroutine Name: led_write
;* Description: Writes the 8 bits of led_data two the 8 LEDs
;*                      on the DFFs at address 0 and 1 on the bus
;*
;* Registers Modified: None
;* Entry Variables: led_data
;* Exit Variables: None
;**************************************************************
led_write:
                  ; preserve accumulator A
                  PSHA

;*** write lower nibble LEDs ***
                  ; set the address
            LDA   #$00
            STA         bus_addr

            ; set the data
            LDA   led_data
            AND         #$0F
            STA         bus_data

            ; write the data
            JSR         bus_write

;*** write upper nibble LEDs ***
                  ; set the address
            LDA   #$01
            STA         bus_addr

            ; set the data
            LDA   led_data
            NSA
            AND         #$0F
            STA         bus_data

            ; write the data
            JSR         bus_write

;*** done ***
                  ; restore accumulator A
                  PULA
                  RTS


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

```
;**************************************************************
;* File Name    :       rtc_driver.asm
;* Author Names :       Matthew Handley
;* Date         :       2014-03-27
;* Description  :       Contains subroutines talking to a DS1337
;*                            Real Time Clock, using i2c_driver.asm
;*
;**************************************************************

; EQU statements

RTC_ADDR_W        EQU $D0     ; Slave address to write to RTC
RTC_ADDR_R        EQU $D1     ; Slave address to read from RTC
RTC_REG_SEC       EQU   $00         ; register address of the seconds register

; Include derivative-specific definitions
            INCLUDE 'MC9S08QG8.inc'

; export symbols
            XDEF rtc_init, rtc_set_time, rtc_get_time, rtc_display_data,
rtc_prompt_time
            XDEF Sec, Min, Hour, Date, Month, Year

; import symbols
                  XREF i2c_init, i2c_start, i2c_stop, i2c_tx_byte, i2c_rx_byte

                  XREF lcd_init, lcd_write, lcd_char, lcd_str, lcd_num_to_char,
lcd_clear, lcd_goto_addr, lcd_goto_row0, lcd_goto_row1
            XREF lcd_data, lcd_char_data, lcd_col_idx

            XREF keypad_get_keypress


; variable/data section
MY_ZEROPAGE: SECTION  SHORT

                  Sec:                    DS.B  2
                  Min:                    DS.B  2
                  Hour:                   DS.B  2
                  Date:                   DS.B  2
                  Month:                        DS.B  2
                  Year:                   DS.B  2

                  Byte_counter:           DS.B  1

MY_CONST: SECTION
; Constant Values and Tables Section

                  str_date:               DC.B  "Date is "
                  str_date_length:  DC.B  8
                  str_time:               DC.B  "Time is "
                  str_time_length:  DC.B  8

                  str_prompt_row0:              DC.B  "Set   : MM/DD/YY"
                  str_prompt_row0_length:       DC.B  16
                  str_prompt_row1:              DC.B  "Clock : HH:MM:SS"
                  str_prompt_row1_length:       DC.B  16


; code section
MyCode:     SECTION
```

```
;*************************************************************
;* Subroutine Name: rtc_init
;* Description: Initilizes the RTC driver.
;*
;* Registers Modified: None
;* Entry Variables: None
;* Exit Variables: None
;*************************************************************
rtc_init:

                  ; load data into time vars
                  MOV         #$00, Sec+0
                  MOV         #$00, Sec+1

                  MOV         #$05, Min+0
                  MOV         #$03, Min+1

                  MOV         #$00, Hour+0
                  MOV         #$09, Hour+1

                  MOV         #$00, Date+0
                  MOV         #$01, Date+1

                  MOV         #$00, Month+0
                  MOV         #$04, Month+1

                  MOV         #$01, Year+0
                  MOV         #$04, Year+1


                  ; set the time
                  JSR         rtc_set_time

                  RTS

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

```
;**************************************************************
;* Subroutine Name: rtc_set_time
;* Description: Set the RTC with the current time in the Sec,
;*                      Min, etc var values
;*
;* Registers Modified: Accu A
;* Entry Variables: None
;* Exit Variables: None
;**************************************************************
rtc_set_time:

                  ; start condition
                  JSR         i2c_start

                  ; send rtc write addr
                  LDA         #RTC_ADDR_W
                  JSR   i2c_tx_byte

                  ; send register address
                  LDA         #RTC_REG_SEC
                  JSR   i2c_tx_byte

                  ; send seconds data
                  LDA         Sec+0
                  NSA
                  AND         #$70
                  ORA         Sec+1
                  JSR   i2c_tx_byte

                  ; send minutes data
                  LDA         Min+0
                  NSA
                  AND         #$70
                  ORA         Min+1
                  JSR   i2c_tx_byte

                  ; send hours data
                  LDA         Hour+0
                  NSA
                  AND         #$30
                  ORA         Hour+1
                  JSR   i2c_tx_byte

                  ; send day of week (not used)
                  LDA         #$01
                  JSR   i2c_tx_byte

                  ; send date data
                  LDA         Date+0
                  NSA
                  AND         #$30
                  ORA         Date+1
                  JSR   i2c_tx_byte

                  ; send month data
                  LDA         Month+0
                  NSA
                  AND         #$10
                  ORA         #$80        ; set century bit
                  ORA         Month+1
                  JSR   i2c_tx_byte
                  ; send year data
                  LDA         Year+0
                  NSA
                  AND         #$F0
                  ORA         Year+1
                  JSR   i2c_tx_byte

                  ; send stop condition
                  JSR         i2c_stop


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

```
;***************************************************************
;* Subroutine Name: rtc_get_time
;* Description: Get the RTC time and save to vars
;*
;* Registers Modified: Accu A
;* Entry Variables: None
;* Exit Variables: None
;***************************************************************
rtc_get_time:

                  ; start condition
                  JSR         i2c_start

                  ; send rtc write addr
                  LDA         #RTC_ADDR_W
                  JSR   i2c_tx_byte

                  ; send register address
                  LDA         #RTC_REG_SEC
                  JSR   i2c_tx_byte

                  ; stop condition
                  JSR         i2c_stop


                  ; set byte counter to 6
                  MOV         #$06, Byte_counter

                  ; start condition
                  JSR         i2c_start

                  ; send rtc read addr
                  LDA         #RTC_ADDR_R
                  JSR   i2c_tx_byte

                  ; read seconds data
                  LDA         #$01                ; ack the byte
                  JSR         i2c_rx_byte
                  STA         Sec+1
                  NSA
                  STA         Sec+0

                  ; read minutes data
                  LDA         #$01                ; ack the byte
                  JSR         i2c_rx_byte
                  STA         Min+1
                  NSA
                  STA         Min+0

                  ; read hours data
                  LDA         #$01                ; ack the byte
                  JSR         i2c_rx_byte
                  STA         Hour+1
                  NSA
                  STA         Hour+0

                  ; read day of week data
                  LDA         #$01                ; ack the byte
                  JSR         i2c_rx_byte
                  ; we don't care about this

                  ; read date data
                  LDA         #$01                  ; ack the byte
                  JSR         i2c_rx_byte
                  STA         Date+1
                  NSA
                  STA         Date+0

                  ; read month data
                  LDA         #$01                  ; ack the byte
                  JSR         i2c_rx_byte
                  STA         Month+1
                  NSA
                  STA         Month+0

                  ; read Year data
                  LDA         #$00                  ; nack the byte
                  JSR         i2c_rx_byte
                  STA         Year+1
                  NSA
                  STA         Year+0

                  ; stop condition
                  JSR         i2c_stop

                  ; mask off the recieved data
                  JSR         rtc_mask_data

                  RTS

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

```
;**************************************************************
;* Subroutine Name: rtc_display_data
;* Description: Takes the data in the Sec, Min, etc vars and
;*                      writes it to the lcd.
;*
;* Registers Modified: Accu A
;* Entry Variables: None
;* Exit Variables: None
;**************************************************************
rtc_display_data:

                  ; clear the lcd
                  JSR   lcd_clear

                  ; goto top row
                  JSR         lcd_goto_row0

                  ; write header
                  LDHX  #str_date
                  LDA         str_date_length
                  JSR         lcd_str

                  ; write month
                  LDA         Month+0
                  JSR         lcd_num_to_char
                  JSR         lcd_char

                  LDA         Month+1
                  JSR         lcd_num_to_char
                  JSR         lcd_char

                  ; write '/'
                  LDA         #'/'
                  JSR         lcd_char

                  ; write Date
                  LDA         Date+0
                  JSR         lcd_num_to_char
                  JSR         lcd_char

                  LDA         Date+1
                  JSR         lcd_num_to_char
                  JSR         lcd_char

                  ; write '/'
                  LDA         #'/'
                  JSR         lcd_char

                  ; write Year
                  LDA         Year+0
                  JSR         lcd_num_to_char
                  JSR         lcd_char

                  LDA         Year+1
                  JSR         lcd_num_to_char
                  JSR         lcd_char

                  ; goto second row on lcd
                  JSR         lcd_goto_row1

                  ; write header
                  LDHX  #str_time
                  LDA         str_time_length
                  JSR         lcd_str

                  ; write hour
                  LDA         Hour+0
                  JSR         lcd_num_to_char
                  JSR         lcd_char

                  LDA         Hour+1
                  JSR         lcd_num_to_char
                  JSR         lcd_char

                  ; write ':'
                  LDA         #':'
                  JSR         lcd_char

                  ; write minute
                  LDA         Min+0
                  JSR         lcd_num_to_char
                  JSR         lcd_char

                  LDA         Min+1
                  JSR         lcd_num_to_char
                  JSR         lcd_char

                  ; write ':'
                  LDA         #':'
                  JSR         lcd_char

                  ; write minute
                  LDA         Sec+0
                  JSR         lcd_num_to_char
                  JSR         lcd_char

                  LDA         Sec+1
                  JSR         lcd_num_to_char
                  JSR         lcd_char

                  ; done
                  RTS


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

```
;**************************************************************
;* Subroutine Name: rtc_mask_data
;* Description: Takes the raw register values recieved in the
;*                      Sec, Min, etc vars and masks off the data we
;*                      want.
;*
;* Registers Modified: Accu A
;* Entry Variables: None
;* Exit Variables: None
;**************************************************************
rtc_mask_data:

                  ; Seconds
                  LDA         Sec+0
                  AND         #$07
                  STA         Sec+0

                  LDA         Sec+1
                  AND         #$0F
                  STA         Sec+1

                  ; Minutes
                  LDA         Min+0
                  AND         #$07
                  STA         Min+0

                  LDA         Min+1
                  AND         #$0F
                  STA         Min+1

                  ; Hours
                  LDA         Hour+0
                  AND         #$03
                  STA         Hour+0

                  LDA         Hour+1
                  AND         #$0F
                  STA         Hour+1

                  ; Date
                  LDA         Date+0
                  AND         #$03
                  STA         Date+0

                  LDA         Date+1
                  AND         #$0F
                  STA         Date+1

                  ; Month
                  LDA         Month+0
                  AND         #$01
                  STA         Month+0

                  LDA         Month+1
                  AND         #$0F
                  STA         Month+1

                  ; Year
                  LDA         Year+0
                  AND         #$0F
                  STA         Year+0
                  LDA         Year+1
                  AND         #$0F
                  STA         Year+1

                  ; done
                  RTS


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


;***************************************************************
;* Subroutine Name: rtc_prompt_time
;* Description: Prompts the user to enter a date and time on
;*                      the LCD with the keypad, and saves the
;*                      user-entered time into the Sec, Min, etc vars.
;*
;* Registers Modified: Accu A
;* Entry Variables: None
;* Exit Variables: None
;***************************************************************
rtc_prompt_time:

;*** write promt to display

                  ; clear the lcd
                  JSR   lcd_clear

                  ; goto top row
                  JSR         lcd_goto_row0

                  ; write header
                  LDHX  #str_prompt_row0
                  LDA         str_prompt_row0_length
                  JSR         lcd_str

                  ; goto bottom row
                  JSR         lcd_goto_row1

                  ; write header
                  LDHX  #str_prompt_row1
                  LDA         str_prompt_row1_length
                  JSR         lcd_str

                  ; goto MM address
                  LDA         #$88
                  JSR         lcd_goto_addr

                  ; prompt for Month+0
                  JSR         keypad_get_keypress
                  STA         Month+0

                  ; prompt for Month+1
                  JSR         keypad_get_keypress
                  STA         Month+1

                  ; goto DD address
                  LDA         #$8B
                  JSR         lcd_goto_addr

                  ; prompt for Date+0
                  JSR         keypad_get_keypress
                  STA         Date+0

                  ; prompt for Date+1
                  JSR         keypad_get_keypress
                  STA         Date+1

                  ; goto YY address
                  LDA         #$8E
                  JSR         lcd_goto_addr

                  ; prompt for Year+0
                  JSR         keypad_get_keypress
                  STA         Year+0

                  ; prompt for Year+1
                  JSR         keypad_get_keypress
                  STA         Year+1

                  ; goto HH address
                  LDA         #$C8
                  JSR         lcd_goto_addr

                  ; prompt for Hour+0
                  JSR         keypad_get_keypress
                  STA         Hour+0

                  ; prompt for Hour+1
                  JSR         keypad_get_keypress
                  STA         Hour+1

                  ; goto MM address
                  LDA         #$CB
                  JSR         lcd_goto_addr

                  ; prompt for Min+0
                  JSR         keypad_get_keypress
                  STA         Min+0

                  ; prompt for Min+1
                  JSR         keypad_get_keypress
                  STA         Min+1

                  ; goto SS address
                  LDA         #$CE
                  JSR         lcd_goto_addr

                  ; prompt for Sec+0
                  JSR         keypad_get_keypress
                  STA         Sec+0

                  ; prompt for Sec+1
                  JSR         keypad_get_keypress
                  STA         Sec+1

                  ; done
                  RTS


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