```
// lab5_code.c
// Anthony Nguyen
// 11.20.2021

#include <avr/io.h>
#include <util/delay.h>
#include <avr/interrupt.h>
// #include <utils/twi.h>
#include <string.h>
#include <stdlib.h>
#include "hd44780.h"
#include "lm73_functions.h"
#include "twi_master.h"
#include "uart_functions.h"

//  HARDWARE SETUP:
//  PORTA is connected to the segments of the LED display. and to the pushbutton
s.
//  PORTA.0 corresponds to segment a, PORTA.1 corresponds to segement b, etc.
//  PORTB bits 4-6 go to a,b,c inputs of the 74HC138.
//  PORTB bit 7 goes to the PWM transistor base.

// Bargraph board           Mega128 board
// --------------      ----------------------
//     reglck            PORTD bit 2 (ss_n)
//     srclk             PORTB bit 1 (sclk)
//     sdin              PORTB bit 2 (mosi)
//     oe_n              PORTB bit 7
//     gnd2                   ground
//     vdd2                     vcc
//     sd_out                no connect

// Encoder board            Mega128 board
// --------------      ----------------------
//   shift_ld_n             PORTE bit 6
//   clk_inh                PORTD bit 3 (ss)
//   sck                    PORTB bit 1 (sclk)
//   ser_in                 no connect
//   ser_out                PORTB bit 3 (miso)
//   vdd1                   vcc
//   gnd1                   ground

// Audio Amp                Mega128 board
// --------------      ----------------------
//  vol                     PORTE bit 3

// #define F_CPU 16000000 // cpu speed in hertz
#define TRUE 1
#define FALSE 0
#define MAX_BIT_DEBOUNCE 8 // numbers of bytes for the debounce

// segs to turn on for LED, negate everything
#define ZERO 0b00111111  // A, B, C, D, E, F
#define ONE 0b00000110   // B, C
#define TWO 0b01011011   // A, B, D, E, G
#define THREE 0b01001111 // A, B, C, D, G
#define FOUR 0b01100110  // B, C, F, G
#define FIVE 0b01101101  // A, C, D, F, G
#define SIX 0b01111101   // A, C, D, E, F, G
```

```
#define SEVEN 0b00000111 // A, B, C
#define EIGHT 0b01111111 // A, B, C, D, E, F, G
#define NINE 0b01100111  // A, B, C, F, G
#define BLANK 0x00
#define COLON 0b00000011 // A, B
// Port B decoder, remember to not all the digits
#define DIGIT1 0x40     //((1 << PB6) | (0 << PB5) | (0 << PB4))
#define DIGIT2 0x30     //((0 << PB6) | (1 << PB5) | (0 << PB4))
#define DIGIT3 0x10     //((0 << PB6) | (0 << PB5) | (1 << PB4))
#define DIGIT4 0x00     //((0 << PB6) | (0 << PB5) | (0 << PB4))
#define DIS_COLON 0x20  //((0 << PB6) | (1 << PB5) | (0 << PB4))
#define TRI_BUFFER 0x70 //((1 << PB6) | (1 << PB5) | (1 << PB4))

//*************** structs ***********************************
struct Clock
{
    uint8_t seconds;
    uint8_t minutes;
    uint8_t hours;
};

struct Alarm
{
    uint8_t seconds;
    uint8_t minutes;
    uint8_t hours;
};

struct LcdDisplay
{
    int8_t insideOutsideFlag; // 1 if outside, 0 if inside
    char *alarm;
    char outside_temperature[16];
    char *outside_temperature_F;
    char *outside_temperature_C;
    char inside_temperature[16];
    char *inside_temperature_C;
    char *inside_temperature_F;
};
// ***********************************************************
// lab 4 define
#define SNOOZE_TIMER 10

//holds data to be sent to the segments. logic zero turns segment on
uint8_t segment_data[5];

//decimal to 7-segment LED display encodings, logic "0" turns on segment
uint8_t dec_to_7seg[12];

// Decoder 3 to 8
uint8_t decoder[8];

// current number on the display
uint16_t current_num = 0;

// what value to display
static uint8_t barGraphDisplay = 0;

// determine if we are increment or decrement mode
```

```
static uint8_t data = 0;

// holding the ADC value
uint16_t adc_result; //holds adc result

// flags
static uint8_t colonDisplay = 0;     // blinking for colons
static uint8_t timerFlag = 0;        // if timer is on, 10 seconds
static uint8_t alarmFlag = 0;        // indication on LED display
static uint8_t changeMinuteFlag = 0; // change the clock minutes
static uint8_t changeHourFlag = 0;   // change the clock hours
static uint8_t setAlarm = 0;         // setting the alarm to desire time
static uint8_t alarmInit = 0;        // alarm desire declared, know many times t
he button has been pressed
static uint8_t encoderUp = 0;        // toggle ecnoder up

// clock
static uint16_t timer = 0; // in seconds

// lab 2 functions
int8_t chk_buttons(int button); // check what button is being pressed
void segsum(uint16_t sum);
void setDigit();
void clearDecoder();
void set_dec_to_7seg();
void set_decoder();

// lab 3 functions
void barGraph();
uint8_t encoderRead(uint8_t data, uint8_t knob);
void spi_init(void);
void tcnt0_init(void);
ISR(TIMER0_OVF_vect);

// lab 4 functions
void segclock();
void alarmDisplay();
void buttonPress(uint8_t);
void tcnt1_init(void); // frequency of notes
void tcnt2_init(void);
void tcnt3_init(void);
ISR(TIMER1_COMPA_vect); // ctc, notes
void setVolumeController();
void adc_init(void);
void adc_read(void);
void snoozekiller(void);

// ****************** lab 5 functions and variables **********
ISR(USART0_RX_vect);
void configDisplay();
char lcd_whole_string_array[32];
// uart functions
volatile uint8_t rcv_rdy;
char rx_char;

// lm73 functions
char lcd_string_array[16]; //holds a string to refresh the LCD
static char lcd_string_array_master[16];
char lcd_string_array_F[16]; //holds a string for F
```

```
char lcd_string_array_C[16]; //holds a string for C
uint8_t i;                   // general purpose index

extern uint8_t lm73_wr_buf[2];
extern uint8_t lm73_rd_buf[2];

// ************* new **************
static struct Clock clock;
static struct Alarm alarm;
static struct LcdDisplay lcdDisplay;
// volatile struct LcdDisplay lcdDisplay;
// ********************************

int main()
{
    DDRB = 0xF0;        //set port B bits 4-7 B as outputs
    DDRE |= 0b01001000; // set E6, E3 to output
    DDRD |= 0b00001100; // slave select pins
    DDRC |= 0xFF;       // set prot C to all outputs

    PORTB &= ~(1 << PORTB7);
    PORTC |= (1 << PORTC6) | (1 << PORTC7); // turn it on

    tcnt0_init(); //initalize counter timer zero
    tcnt1_init(); //initalize counter timer one
    tcnt2_init(); //initalize counter timer two
    tcnt3_init(); //initalize counter timer three
    // PORTE |= 1 << PORTE3;
    adc_init(); // initalize the ADC
    spi_init(); //initalize SPI port
    lcd_init(); // initalize the lcd display
    sei();      //enable interrupts before entering loop

    //************************ lab 5 init MASTER ***********************
    // DDRF |= 0x08; // lcd strobe bit
    init_twi();  // initalize twi
    uart_init(); // initalize UART
    // int16_t lm73_temp; // a place to assemble the temperature from the lm73
    DDRF |= 0x08; // lcd strobe bit
    // float lm73_temp_C, lm73_temp_F;
    //set LM73 mode for reading temperature by loading pointer register
    lm73_wr_buf[0] = LM73_PTR_TEMP;             //load lm73_wr_buf[0] with tempe
rature pointer address
    twi_start_wr(LM73_ADDRESS, lm73_wr_buf, 2); //start the TWI write process
    _delay_ms(2);                               //wait for the xfer to finish
    //*****************************************************************

    set_dec_to_7seg(); // set values for dec_to_7seg array
    set_decoder();     // set values for the decoder array
    timer = SNOOZE_TIMER;
    clear_display();
    // cursor_home();

    while (1)
    {

        // spi
        PORTD |= 1 << PORTD3;    // clock_inh = 1
        PORTE &= ~(1 << PORTE6); // load sh/ld
```

```c
        PORTE |= 1 << PORTE6;    // sh/ld
        PORTD &= ~(1 << PORTD3); // clock_inh

        SPDR = 0; // writing a random value

        while (bit_is_clear(SPSR, SPIF));
        data = SPDR; // read data
        barGraph();
        // end of spi

        segclock(); // set each digit for the clock
        setDigit(); // setting the digit on display

        adc_read(); // read the ADC value
        // max value for adc is 1024
        if (adc_result < 100)
        {
            OCR2 = 50; //255/2;
        }
        else
        {
            OCR2 = (255 * -5 / (adc_result)) + 80; // best result
        }
        // check if the alarm matches the actually clock
        if ((alarmInit > 1) && (alarm.minutes == clock.minutes) && (alarm.hours
== clock.hours))
        {
            // timerFlag = 1; // make the timer go off
            // OCR3A = 0x1000;

            alarmFlag = 1;
        }
        // clear_display();
        alarmDisplay(); // display "ALARM" on the LCD display
        configDisplay();

    } //while
    return 0;
} //main

/***********************************************************************/
//                              spi_init
//Initalizes the SPI port on the mega128. Does not do any further
//external device specific initalizations.  Sets up SPI to be:
//master mode, clock=clk/2, cycle half phase, low polarity, MSB first
//interrupts disabled, poll SPIF bit in SPSR to check xmit completion
/***********************************************************************/
void spi_init(void)
{
    DDRB |= 0x07;                         //Turn on SS, MOSI, SCLK
    SPCR |= (1 << SPE) | (1 << MSTR); //enable SPI, master mode
    SPSR |= (1 << SPI2X);                 // double speed operation
} //spi_init

/***********************************************************************/
//                                tcnt0_init
//Initalizes timer/counter0 (TCNT0). TCNT0 is running in async mode
//with external 32khz crystal.  Runs in normal mode with no prescaling.
```

```c
//Interrupt occurs at overflow 0xFF.
//
void tcnt0_init(void)
{
    ASSR |= (1 << AS0);    //ext osc TOSC
    TIMSK |= (1 << TOIE0); //enable TCNT0 overflow interrupt
    TCCR0 |= (1 << CS00);  //normal mode, no prescale
}

/***********************************************************************/
//                                tcnt1_init
//Initalizes timer/counter1 (TCNT1). TCNT1 is running in async mode
//with interal 16Mhz crystal.  Runs in normal mode with no prescaling.
//Interrupt occurs at OCR1A value.
//
void tcnt1_init(void)
{
    TIMSK |= (1 << OCIE1A);               //enable TCNT1 ctc interrupt
    TCCR1A |= 0;                          // CTC on OC1A
    TCCR1B |= (1 << CS10) | (1 << WGM12); // no prescalar and ctc
    TCCR1C |= 0x0;
    OCR1A = 32000; // 440Hz, A4, change later for beaver fight song
}

/***********************************************************************/
//                                tnct2_init
// Initalizes timer/counter2 (TCNT2). TNCT2 is running a fast PWM mode.
// This will be on PORTB7. This timer to be used to set the brightness of
// the LED display
/***********************************************************************/
void tcnt2_init(void)
{
    //fast PWM, set on match, 64 prescaler
    // TCCR2 |= (1 << WGM21) | (1 << WGM20) | (1 << COM21) | (1 << COM20) | (1 <
< CS22);
    TCCR2 = 0b01111001; // removes the flickering
    OCR2 = 0xF0;        //clear at 0xF0 CLEAR AT BRIGHTNESS
}

/***********************************************************************/
//                                tnct3_init
// Initalizes timer/counter3 (TCNT3). TNCT3 is running a fast PWM mode,
// Uses OC3A which is on PE3. Clear at the bottom, inverting mode
// This sets the volume control for the speaker.
/***********************************************************************/
void tcnt3_init(void)
{
    //Fast PWM, set on compare match
    TCCR3A |= (1 << WGM31) | (1 << COM3A1) | (1 << COM3A0);
          // inverting mode
    TCCR3B |= /*(1 << ICES3) |*/ (1 << WGM33) | (1 << WGM32) | (1 << CS30) | (1
<< CS31); //No prescale
    TCCR3C = 0x00;
          //no force compare

    OCR3A = 0xFFFF; // initally no volume
    // OCR3A = 0x1000;
    ICR3 = 0xF000; // top value
}
```

```c
/***********************************************************************/
//                              adc_init
/***********************************************************************/
void adc_init(void)
{

    //Initalize ADC and its ports
    DDRF &= ~(_BV(DDF7)); //make port F bit 7 the ADC input
    PORTF &= ~(_BV(PF7)); //port F bit 7 pullups must be off

    ADMUX |= (0 << REFS1) | (1 << REFS0) | (0 << MUX4) | (0 << MUX3) | (1 << MUX
2) | (1 << MUX1) | (1 << MUX0); //single-ended input, PORTF bit 7, right adjuste
d, 10 bits
                                //reference is AVCC

    ADCSRA |= (1 << ADEN) | (1 << ADPS2) | (1 << ADPS1) | (1 << ADPS0) | (1 << A
DPS0); //ADC enabled, don’t start yet, single shot mode

       //division factor is 128 (125khz)
}

/***********************************************************************/
//                              adc_read()
/***********************************************************************/
void adc_read()
{
    ADCSRA |= (1 << ADSC); //poke the ADSC bit and start conversion
    while (bit_is_clear(ADCSRA, ADIF))
    {
    }                      //spin while interrupt flag not set
    ADCSRA |= 1 << ADIF; //its done, clear flag by writing a one
    adc_result = ADC;    //read the ADC output as 16 bits
}

/*******************************************************************************/
//                                set_dec_to_7seg
// setting the dec_to_7seg array for which segment to turn off in order to see
// the digit on the LED display.
/*******************************************************************************/
void set_dec_to_7seg()
{
    dec_to_7seg[0] = ~(ZERO);
    dec_to_7seg[1] = ~(ONE);
    dec_to_7seg[2] = ~(TWO);
    dec_to_7seg[3] = ~(THREE);
    dec_to_7seg[4] = ~(FOUR);
    dec_to_7seg[5] = ~(FIVE);
    dec_to_7seg[6] = ~(SIX);
    dec_to_7seg[7] = ~(SEVEN);
    dec_to_7seg[8] = ~(EIGHT);
    dec_to_7seg[9] = ~(NINE);
    dec_to_7seg[10] = ~(COLON);
    dec_to_7seg[11] = ~(BLANK);
}

/*******************************************************************************/
//                                set_decoder
```

```c
// This function sets the right value for decoder so that it display the right
// digit. The index value of the decoder represents the Yx output of the decoder
.
/*******************************************************************************/
void set_decoder()
{
    decoder[0] = DIGIT4;
    decoder[1] = DIGIT3;
    decoder[2] = DIS_COLON;
    decoder[3] = DIGIT2;
    decoder[4] = DIGIT1;
    decoder[7] = TRI_BUFFER;
}

//*********************************************************************************
//                              chk_buttons
//Checks the state of the button number passed to it. It shifts in ones till
//the button is pushed. Function returns a 1 only once per debounced button
//push so a debounce and toggle function can be implemented at the same time.
//Adapted to check all buttons from Ganssel’s "Guide to Debouncing"
//Expects active low pushbuttons on PINA port.  Debounce time is determined by
//external loop delay times 12.
//
int8_t chk_buttons(int button)
{

    static uint16_t state[MAX_BIT_DEBOUNCE]; //holds present state

    // bit_is_clear: test whether but but in IO register sfr is clear. This will
 return non zero
    // if the but is clear, and 0 if the bit is set
    // handling multiple inputs
    // https://www.avrfreaks.net/sites/default/files/debouncing.pdf’

    state[button] = (state[button] << 1) | (!bit_is_clear(PINA, button)) | 0xE00
0; // when the second button is pressed

    if (state[button] == 0xF000)
        return 1;

    return 0;
}

//*********************************************************************************
*****
//                                   segment_sum
//takes a 16-bit binary input value and places the appropriate equivalent 4 digi
t
//BCD segment code in the array segment_data for display.
//array is loaded at exit as:  |digit3|digit2|colon|digit1|digit0|
void segsum(uint16_t sum)
{
    // sum is the total count, place each digit into segment_data[5]
    // determine how many digits there are
    //break up decimal sum into 4 digit-segments
    //blank out leading zero digits
    //now move data to right place for misplaced colon position
    int i; //, leading_zero;
    segment_data[0] = sum % 10;
    segment_data[1] = (sum % 100) / 10;
    // segment_data[2] = 11; // doesn’t turn on the colon, blank
    segment_data[2] = (colonDisplay == 1) ? 10 : 11;
    segment_data[3] = (sum % 1000) / 100;
    segment_data[4] = sum / 1000;

    // remove the leading zeros
    // leading_zero = 1;
    for (i = 4; i > 0; i--)
    {
        if (i == 2)
            continue;
        if (segment_data[i] == 0)
            segment_data[i] = 11; // replace it with a blank

        else
            break;
    }

} //segment_sum

//*******************************************************************************
*****
//                                segclock
//takes two 8-bit binary values(hours and minutes) and places the appropriate
//equivalent 4 digit.
//BCD segment code in the array segment_data for display.
//array is loaded at exit as:  |digit3|digit2|colon|digit1|digit0|
void segclock()
{
    if (setAlarm == 0)
    {
        segment_data[0] = clock.minutes % 10;
        segment_data[1] = clock.minutes / 10;
        segment_data[2] = (colonDisplay == 1) ? 10 : 11;
        segment_data[3] = clock.hours % 10;
        segment_data[4] = clock.hours / 10;
    }
    if (setAlarm == 0x1)
    {
        segment_data[0] = alarm.minutes % 10;
        segment_data[1] = alarm.minutes / 10;
        segment_data[2] = (colonDisplay == 1) ? 10 : 11;
        segment_data[3] = alarm.hours % 10;
        segment_data[4] = alarm.hours / 10;
    }
}

/***************************************************************/
//                      setDigit function
// it will choose its given digit and set that number for it.
// The cases set the value on PORTA to the right segments and PORTB
// to decoder.
/***************************************************************/

void setDigit()
{
    DDRA = 0xFF; // setting PORT A as an output
```

```c
    int i;
    uint8_t dis;
    for (i = 0; i < 5; i++)
    {                           // looping through the segment data and assigning th
e port the right values.
        PORTB = decoder[i]; // enable the right digit to turn on
        dis = dec_to_7seg[segment_data[i]];
        if ((i == 4) && (setAlarm == 1))
        {

            dis &= ~(1 << 7);
        }
        PORTA = dis; // turn on the right segments
        _delay_ms(0.5);
    }
}

/******************************************************************************/
//                                      ISR
// Then fucntion will will called when there is an interrupt within the system
// and when the overflow flag for timer counter 0 it set.
// This fucntions checks the push buttons to see which buttons were pressed
// then set it in its correct mode.
// Afterwards checks the encoder to see where it is.
/******************************************************************************/
ISR(TIMER0_OVF_vect)
{
    uint16_t i, j;
    static uint8_t count = 0, seconds;
    //insert loop demake lay for debounce

    // checking the push buttons
    // for loop for each phase of the digit
    PORTB |= TRI_BUFFER;

    for (i = 0; i < 12; i++)
    { // for the debounce

        //make PORTA an input port with pullups
        DDRA = 0x00;  // set port A as inputs
        PORTA = 0xFF; // set port A as pull ups

        // checking what button is being pressed
        for (j = 0; j < 8; j++)
        {
            if (chk_buttons(j))
                buttonPress(j);
        }
    }
    PORTB &= ~(TRI_BUFFER); // turn off the tri state buffer

    // reading each knob
    uint8_t enc1 = encoderRead(data, 0);
    uint8_t enc2 = encoderRead(data, 1);

    // each case of what the knob or buttons will be
    if (setAlarm == 0)
    {
        if ((encoderUp == 0) && (enc1 == 0 || enc2 == 0))
```

```c
        {
            // ccw
            //current_num -= 1;
            if (changeMinuteFlag == 1 && changeHourFlag == 0)
            {
                // change minutes
                clock.minutes--;
                if (clock.minutes == 255) // since its unsign 255 = -1
                    clock.minutes = 59;
            }
            if (changeHourFlag == 1 && changeMinuteFlag == 0)
            {
                clock.hours--;
                if (clock.hours == 255)
                    clock.hours = 23;
            }
        }

        if ((encoderUp == 1) && (enc1 == 0 || enc2 == 0))
        {
            // cw
            // current_num += 1;
            if (changeMinuteFlag == 1 && changeHourFlag == 0)
            {
                // change minutes
                clock.minutes++;
                if (clock.minutes % 60 == 0)
                    clock.minutes = 0;
            }
            else if (changeHourFlag == 1 && changeMinuteFlag == 0)
            {
                clock.hours++;
                if (clock.hours % 24 == 0)
                    clock.hours = 0;
            }
        }
    }

    // when the alarm flag is on set the the alarm desire time
    if (setAlarm == 0x1)
    {
        // have encoder 2 change the hours
        if ((encoderUp == 0) && (enc2 == 0))
        {
            alarm.hours--;
            if (alarm.hours == 255)
                alarm.hours = 23;
        }

        if ((encoderUp == 1) && (enc2 == 0))
        {
            alarm.hours++;
            if ((alarm.hours % 24 == 0) || (alarm.hours > 23))
                alarm.hours = 0;
        }

        // have encoder 1 change the minutes
        if ((encoderUp == 0) && enc1 == 0)
        {
```

```c
            // change minutes
            alarm.minutes--;
            if (alarm.minutes == 255 || (alarm.minutes >= 60 && alarm.minutes <=
 255)) // since its unsign 255 = -1
                alarm.minutes = 59;
        }

        if ((encoderUp == 1) && enc1 == 0)
        {
            // change minutes
            alarm.minutes++;
            if (alarm.minutes >= 60) // since its unsign 255 = -1
                alarm.minutes = 0;
        }
    }

    // add a counter to determine one second
    count++;
    if ((count % 128) == 0)
    {
        // 1 second has past
        // lab 5 temp sensor
        int16_t lm73_temp;
        float lm73_temp_C, lm73_temp_F;
        // clear_display();                                 //wipe the display
        twi_start_rd(LM73_ADDRESS, lm73_rd_buf, 2);     //read temperature data
from LM73 (2 bytes)
        _delay_ms(2);                                   //wait for it to finish
        lm73_temp = lm73_rd_buf[0];                     //save high temperature
byte into lm73_temp
        lm73_temp = lm73_temp << 8;                     //shift it into upper by
te
        lm73_temp |= lm73_rd_buf[1];                    //"OR" in the low temp b
yte to lm73_temp
        lm73_temp_C = lm73_temp / (float)256;           // how to find the temp
in C
        lm73_temp_F = (lm73_temp_C * 9 / 5) + 32;       // convert C to F
        dtostrf(lm73_temp_C, 0, 1, lcd_string_array_C); // converting float to s
tring
        dtostrf(lm73_temp_F, 0, 1, lcd_string_array_F); // converting float to s
tring
        strcpy(lcd_string_array, " ");                 // add C degrees
        strcat(lcd_string_array, lcd_string_array_C);   // add C degrees
        strcat(lcd_string_array, "C ");
        strcat(lcd_string_array, lcd_string_array_F);
        strcat(lcd_string_array, "F");
        clear_display();
        string2lcd(lcd_string_array);

        // set local temp in struct
        // strcpy(lcdDisplay.inside_temperature, lcd_string_array);


        // clear_display();                                //wipe the display
        uart_putc('\0');
        // ************** start rcv portion *********************
        if (rcv_rdy == 1)
        {
            clear_display();
            lcdDisplay.insideOutsideFlag = 1;
            // line2_col1();
            string2lcd(" ");
            string2lcd(lcd_string_array_master); // write out string if its read
y
            // fill_spaces();
            // lcdDisplay.
            rcv_rdy = 0;
            // cursor_home();
        }
        else
        {
            lcdDisplay.insideOutsideFlag = 0;
            clear_display();
            // line2_col1();
            string2lcd(lcd_string_array); //send the string to LCD (lcd_function
s)
        }
        // *************** end rcv portion **********************

        // timer, snooze, then alarm again
        if (timerFlag == 0x1)
        {
            // count down from snooze
            // display alarm
            //
            // alarmFlag = 1; // display alarm
            // timer on

            // OCR3A = 0xFFFF; // turn off volume
            // PORTC &= ~(1 << PORTC0);
            // PORTC &= ~(1 << PORTC1);
            // clear_display();
            alarmFlag = 0; // disable the display alarm
            timer--;
            if (timer == 0)
            {
                // timer goes off display alarm
                timerFlag = 0; // turn off timer
                alarmFlag = 1;
                OCR3A = 0x1000; // turn off volume
                PORTC |= (1 << PORTC0);
                PORTC |= (1 << PORTC1);
                // clear_display();
                // snoozekiller();
            }
            if (OCR3A != 0xffff)
            {
                OCR3A = 0xFFFF; // turn off volume
                PORTC &= ~(1 << PORTC0);
                PORTC &= ~(1 << PORTC1);
            }
        }

        // clock
        colonDisplay ^= 0x1; // blinking
        seconds++;

        if ((seconds % 60) == 0)
```

```
        {
            clock.minutes++;
            seconds = 0;
            if ((clock.minutes % 60) == 0)
            {
                clock.hours++;
                clock.minutes = 0;
                if (clock.hours % 24 == 0)
                    clock.hours = 0;
            }
        }
    }
}

/******************************************************************************/
//                                  encoderRead
// This function checks the state of the encoder so see what its behavior is.
// It will return -1 if there is no change within. It will return 1 if the syste
m is
// CW. It will return 0 if the system is CCW.
/******************************************************************************/
uint8_t encoderRead(uint8_t data, uint8_t knob)
{

    // check for encoder
    static uint8_t old_state[4] = {0xff, 0xff, 0xff, 0xff};
    uint8_t new_A = -1;
    uint8_t new_B = -1;
    static uint8_t count = 0;
    uint8_t return_val, a, b, a_index, b_index;

    a = (knob == 0) ? 1 : 4; // where the position of a is
    b = (knob == 0) ? 2 : 8; // where the position of b is

    a_index = (knob == 0) ? 0 : 2;
    b_index = (knob == 0) ? 1 : 3;

    new_A = (data & a) ? 1 : 0; // most LSB
    new_B = (data & b) ? 1 : 0; // 2nd LSB

    return_val = -1; // default return value, no change

    if ((new_A != old_state[a_index]) || (new_B != old_state[b_index]))
    { // if change occured
        if ((new_A == 0) && (new_B == 0))
        {
            if (old_state[a_index] == 1)
            {
                count++;
            }
            else
            {
                count--;
            }
        }
        else if ((new_A == 0) && (new_B == 1))
        {
            if (old_state[a_index] == 0)
            {
```

```
                count++;
            }
            else
            {
                count--;
            }
        }
        else if ((new_A == 1) && (new_B == 1))
        { // detent position
            if (old_state[a_index] == 0)
            { // one direction
                if (count == 3)
                {
                    return_val = 0;
                }
            }
            else
            { // or the other direction
                if (count == -3)
                {
                    return_val = 1;
                }
            }
            count = 0; // count is always reset in detent position
        }
        else if ((new_A == 1) && (new_B == 0))
        {
            if (old_state[a_index] == 1)
            {
                count++;
            }
            else
            {
                count--;
            }
        }

        old_state[a_index] = new_A; // save what are now old values
        old_state[b_index] = new_B;

    } // if changed occured
    // if return value is still -1 then nothing happen
    return (return_val); // return coder state
}

/******************************************************************************/
//                                  barGraph
// Set the mode on the bar graph.
/******************************************************************************/
void barGraph()
{

    SPDR = barGraphDisplay;
    while (bit_is_clear(SPSR, SPIF))
    {
    }                         //wait till data sent out (while loop)
    PORTD |= (1 << PORTD2); //HC595 output reg - rising edge...
    PORTD &= (0 << PORTD2); //and falling edge
}
```

```
/******************************************************************************/
//                                  alarmDisplay
// Display "ALARM" on the display if the alarm flag is on.
// Otherwise clear the screen.
/******************************************************************************/

void alarmDisplay()
{
    // char lcd_string_array_alarm[5] = "     ALARM      ";
    cursor_home();
    if (alarmFlag == 0x1)
    {
        // DDRE |= 1 << PORTE3; // turn off the port of the speaker
        OCR3A = 0x1000; // turn on the volume
        lcdDisplay.alarm = " ALARM"; // display alarm
    }
    else
    {
        lcdDisplay.alarm = "   "; // display blanks when the alarm isn't o
    }
}

/******************************************************************************/
//                                  buttonPress
// Different cases for each button pressed
/******************************************************************************/
void buttonPress(uint8_t button)
{

    switch (button)
    {
    case 0:
    {
        // snooze, turn off LCD display
        snoozekiller();
        return;
    }
    case 1:
    {
        // alarmFlag ^= 0x1; // show on the LED Display that you want to change
the alarm time
        setAlarm ^= 0x1; // toggle the alarm flag
        barGraphDisplay ^= 1 << 1;
        alarmInit++; // many times this button has been pressed
        return;
    }
    case 2:
    {
        // snooze button
        timer = SNOOZE_TIMER; // 10 seconds
        // sleep for 10 seconds then alarm again
        timerFlag = 1;
        barGraphDisplay ^= 1 << 2;
    }
    case 3:
    {
        encoderUp ^= 1; // toggle encoder rotating the other way
    }
    case 6:
    {
        // change minutes
        changeMinuteFlag ^= 1;
        barGraphDisplay ^= 1 << 6;
        return;
    }
    case 7:
    {
        // change hours
        changeHourFlag ^= 1;
        barGraphDisplay ^= 1 << 7;
        return;
    }
    default:
        break;
    }
}

/******************************************************************************/
/******************************************************************************/
ISR(TIMER1_COMPA_vect)
{
    PORTC ^= 1 << PORTC0; // turn on right speaker
    PORTC ^= 1 << PORTC1; // turn on left speaker
}

/******************************************************************************/
//                              snooze
// what happens when the snooze goes off, reset everything
/******************************************************************************/
void snoozekiller(void)
{
    timer = SNOOZE_TIMER;         // reset the timer to 10 seconds
    OCR3A = 0xFFFF;               // turn off volume
    timerFlag = 0;                // turn off the timer
    alarmFlag = 0;                // turn off the alarm
    barGraphDisplay &= ~(1 << 1); // turn off the timer modes
    barGraphDisplay &= ~(1 << 2); // turn off the timer modes
    PORTC &= ~(1 << PORTC0);
    PORTC &= ~(1 << PORTC1);
    alarm.minutes = 60;
    alarm.hours = 24;
    alarmInit = 0; // set it so that it so that there is not alarm set
    setAlarm = 0;
    // clear_display(); // when this is comment it out the temp changes numbers

    // turn off indication on LED display
}

//********************************************************
//                      ISR(USART0_RX_vect)
//********************************************************
ISR(USART0_RX_vect)
{
    static uint8_t j;
    rx_char = UDR0;                         //get character
    lcd_string_array_master[j++] = rx_char; //store in array
    // lcdDisplay.outside_temperature[j++] = rx_char;
```

```
    if (rx_char == '\0')
    {
        rcv_rdy = 1;
        j = 0;
    }
}


// ********************************************************
//                        configDisplay
// row 1: ALARM
// row 2: local_temp outside_temp
// ********************************************************
void configDisplay()
{
    char lcdrow1[16], lcdrow2[16];

    // row 1
    strcpy(lcdrow1, lcdDisplay.alarm);
    strcat(lcdrow1, '\0');

    // row 2
    if (lcdDisplay.insideOutsideFlag == 0)
        strcpy(lcdrow2, lcdDisplay.inside_temperature);
    else
        strcpy(lcdrow2, lcdDisplay.outside_temperature);
    strcat(lcdrow2, '\0');

    // display the info
    // clear_display(); // clear whatever was on the screen
    line2_col1();
    string2lcd(lcdrow1);
    // line2_col1(); // set cursor to second line
    // string2lcd(lcdrow2); // display either local or remote temp

}
```