```c
// lab4.c
// Victor Garcia Flores
// 11.23.2019

#define F_CPU 16000000 // cpu speed in hertz
#define TRUE 1
#define FALSE 0
#include <avr/io.h>
#include <math.h>
#include <util/delay.h>
#include <avr/interrupt.h>
#include <string.h>
#include "uart_functions.h"
#include "lm73_functions.h"
#include "twi_master.h"
#include "si4734.h"

//holds data to be sent to the segments. logic zero turns segment on
uint8_t segment_data[5]={0,0,0b11111100,0,0,0};

//decimal to 7-segment LED display encodings, logic "0" turns on segment
//Note: They are arranged so that the value of a possible integer matched with t
he position
uint8_t dec_to_7seg[12] = {0b11000000, 0b11111001,0b10100100,0b10110000,0b100110
01,0b10010010,0b10000010,0b11111000,0b10000000,0b10011000,0b01111111,0b11111111}
;

//Real-time clock counters
uint8_t seconds;
int8_t hours = 0;
int8_t minutes = 0;

//Colon Variable
uint8_t colon = 0x01;

//ADC Variables
uint16_t last_adcVal;

//General Encoder Variables
uint8_t raw_encoder = 0; //raw data from break out board
//Encoder #1
uint8_t prevL_Encoder=0;
uint8_t currL_Encoder=0;
//Encoder #2
uint8_t prevR_Encoder=0;
uint8_t currR_Encoder=0;

//volatile raw segment data
uint16_t volatile encoder_test;

//Global Button Variables
uint8_t ButtonState = 1; //increment and decrement value
uint8_t buttons[8] = {0}; // used to see which button was pressed

//startup flag
uint8_t start_flag=0; //used for encoder

//variable for current value
int16_t CurrCountVal = 0;

//Button Variables
uint8_t ChangeTime = 0;
uint8_t ChangeAlarmTime = 0;
uint8_t AlarmOnOff = 0;
uint8_t Snooze = 0;
uint8_t Volumeup = 0;
uint8_t Volumedown = 0;
uint8_t buttonsToggled = 0;
```

```c

//Alarm Managing
int8_t AlarmHrs = 12;
int8_t AlarmMins = 0;
uint8_t AlarmSounding = 0;
uint8_t SnoozeSecCounter = 0;

//Temperature Sensor Variables
char    lcd_string_array[16];  //holds a string to refresh the LCD
char     lcd_string_C[16];  //holds Celcius string
char     lcd_string_F[16];  //holds Farenheit string
char     lcd_draft[32] = {' '};  //holds final output string
char     lcd_output[32];  //holds final output string
extern uint8_t lm73_wr_buf[2];
extern uint8_t lm73_rd_buf[2];
uint16_t lm73_temp;  //a place to assemble the temperature from the lm73
uint16_t prev_lm73_temp; //store previous sensor value

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

  ADMUX |= (0<<ADLAR) | (1<<REFS0)|(1<<MUX2)|(1<<MUX1)|(1<<MUX0) ; //single-ende
d, input PORTF bit 7, right adjusted, 10 bits

  ADCSRA |= (1<<ADEN)|(1<<ADPS2)|(1<<ADPS1)|(1<<ADPS0);  //ADC enabled, don't st
art yet, single shot mode
                             //division factor is 128 (125khz)
}

void adc_read(){
  uint8_t adc_result;
  ADCSRA |= (1<<ADSC); //poke ADSC 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
  last_adcVal = div(adc_result, 205);
  OCR2 = adc_result;
}

/***********************************************************************/
//                              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){
  //DDRD |= (1<<PD4); //regclk
  DDRB   |= ((1<<PB0)|(1<<PB1)|(1<<PB2)| (0<<PB3)); //output mode for MOSI, SCLK
  SPCR   = (1<<SPE) | (1<<MSTR); //master mode, clk low on idle, leading edge sa
mple (p. 167)
  SPSR   = (1<<SPI2X); //choose double speed operation // double speed operation

  /* Run this code before attempting to write to the LCD.*/
  DDRF  |= 0x08;  //port F bit 3 is enable for LCD
  PORTF &= 0xF7;  //port F bit 3 is initially low

 }//spi_init

 /***********************************************************************/
 //
                               spi_read
 //Reads the SPI port.
 /***********************************************************************/
 uint8_t spi_read(void){
        SPDR = 0x00; //"dummy" write to SPDR
        while (bit_is_clear(SPSR,SPIF)){} //wait till 8 clock cycles are done
        return(SPDR); //return incoming data from SPDR
 }

/***********************************************************************/
//                              tcnt0_init
//Initalizes timer/counter0 (TCNT0). TCNT0 is running in async mode
//with external 32khz crystal.  Runs in normal mode with no prescaling.
//Interrupt occurs at overflow 0xFF. This is used to keep track of time.
//******************************************************************************
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
// Initializes the configuration for the sound pins. I have selected
// CTC mode, no pre-scalar, with a frequency of 2k Hz
//******************************************************************************
void tcnt1_init(void){
  DDRD |= (1<<PD5);
        TCCR1B |= (1<<WGM12)|(1<<CS10); //CTC at TOP
  //Initialize the tone to be off
  OCR1A = 3999;
  TIMSK |= (1<<OCIE1A); //set tcnt1 compare match
}

/***********************************************************************/
//                              tcnt2_init
//Initalizes timer/counter0 (TCNT2). This is used to drive the PWM pin for the
//7-segment display. This sets up our configuration for the dimming option.
//******************************************************************************
void tcnt2_init(void){
  TIMSK  |=  (1<<TOIE2); //enable TCNT2 overflow interrupt
  TCCR2  |=  (1<<CS20) | (0<<CS21)|(1<<WGM20)|(1<<WGM21)| (1<<COM20) | (1<<COM21
); //normal mode, no prescale
  OCR2 = 200;
}

/*************************************************************************/
//                           section_tester
//This is is used to test to see if we get to certain places in the code
//Whatever value is passed into this function will be presented onto the graph
/*************************************************************************/
void section_tester(uint8_t state){
  /* Start transmission */
  SPDR = state;
  while (bit_is_clear(SPSR,SPIF)){} //spin till SPI data has been sent

  PORTD |= (1<<PD4);        //send rising edge to regclk on HC595
  PORTD &= ~(1<<PD4);            //send falling edge to regclk on HC595
}

/*************************************************************************/
//                           LCDUpdater
//To help with speed, we only want to update the LCD display when a change takes
// place. In this lab all we are writing is ALARM
/*************************************************************************/
void LCDUpdater(){
  // clear_display();
  // cursor_home();
  // if(AlarmOnOff == 1){
  //   string2lcd("ALARM");
  // }
  //lcd_string_F[16];
  //handle whether it should say ALARM or be blank

  //temptrSens();
```

```
  if(AlarmOnOff == 1){
    //string2lcd("ALARM");
    lcd_draft[0] = 'A';
    lcd_draft[1] = 'L';
    lcd_draft[2] = 'A';
    lcd_draft[3] = 'R';
    lcd_draft[4] = 'M';
  }
  else{
    uint8_t indexLCD;
    for(indexLCD=0;indexLCD<6;indexLCD++){
      lcd_draft[indexLCD] = ' ';
    }
  }
  //empty the spots in between
  uint8_t indexLCD2;
  for(indexLCD2=6;indexLCD2<16;indexLCD2++){
    lcd_draft[indexLCD2] = ' ';
  }

  //handle Farenheit display
  for(int j=16;j<22;j++){
    lcd_draft[j] = lcd_string_F[j-16];
  }
  //empty the spots in between
    lcd_draft[22] = ' ';
  //handle Celcius display
  for(int k=23;k<29;k++){
    lcd_draft[k] = lcd_string_C[k-23];
  }
  //empty the spots in between
  uint8_t indexLCD3;
  for(indexLCD3=29;indexLCD3<=31;indexLCD3++){
    lcd_draft[indexLCD3] = ' ';
  }
}

/*************************************************************************/
//                           AlarmHandler
//This handles anything alarm related. This ranges from determining when to play
//our tone, to snoozing. In short, this functions checks our flags and acts
//acordingly.
/*************************************************************************/
void AlarmHandler(){
  //If the alarm isn't be sounding
  if(AlarmSounding == 0){
    //but is enabled
    if(AlarmOnOff){
      //check to see if the alarm should be going off
      if((hours == AlarmHrs) && (minutes == AlarmMins)){
        AlarmSounding = 1;
      }
    }
  }
  //if alarm is off, make it so that no sound plays
  else if(AlarmOnOff == 0){
    AlarmSounding = 0;
  }
  //If snooze was turned on, change flag so that no sound plays
  if(Snooze){
    AlarmSounding = 0;
  }
  //if we have reached 10 sec of snooze, enable sound
  if(SnoozeSecCounter == 10){
    AlarmSounding = 1;
    Snooze = 0;
    SnoozeSecCounter = 0;
  }
  //If we should be playing a tone, enable the interrupt
  if(AlarmSounding){
    //Enable interrupt.
    TIMSK |= (1<<OCIE1A);

    //Set the value we calulated for the desired frequency
    OCR1A = 3999; //What makes the sound go off
  }
  //make sure sound is off
  else if(AlarmSounding == 0){
    //disable interrupt flag; used to help with speed. This way we aren’t always
    //interrupting
    TIMSK &= ~(1<<OCIE1A);
    //Reset value to zero. It’s a safety net so we don’t hear anything
    OCR1A = 0;
  }
}

//******************************************************************************
//                            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.
//******************************************************************************
uint8_t chk_buttons(uint8_t button) {
        static uint16_t state[8] = {0}; //We do what we did in lab 1, but this t
ime as an array so we can address the other buttons
        state[button] = ((state[button]<<1) | (!bit_is_clear(PINA,button)) | 0xE
000);
        if(state[button] == 0xFF00) return 1;
        return 0;
}

/*************************************************************************/
//                           Read_ButtonsV2()
// Button 7: Change time
// Button 6: Change Alarm Time
// Button 5: Enable/disable alarm clock
// Button 4: Snooze
/*************************************************************************/
void Read_ButtonsV2(){
        int BttnNum = 0;
        //Let’s read button data
  DDRA = 0x00; //sets as input
  PORTA = 0xFF; //pulls up the resistors
  PORTB |= ((1<<PB4)|(1<<PB5)|(1<<PB6)); //Select bits for the buttons

  for(BttnNum = 0; BttnNum <= 7; BttnNum++){
    if(chk_buttons(BttnNum)){ //If we read button input
      if(BttnNum == 7){ //7th button is pressed
        buttons[7] = 1; //button array
        ChangeTime ^= 1;

        //Clear other condition involving time
        ChangeAlarmTime = 0;
      }
      else if(BttnNum == 6){ //6th button is pressed
        buttons[6] = 1; //button array
        ChangeAlarmTime ^= 1;

        //Clear other condition involving time
        ChangeTime = 0;
      }
      else if(BttnNum == 5){ //5th button is pressed
        buttons[5] = 1; //button array
        AlarmOnOff ^= 1;
        buttonsToggled = 1;
```

```
      }
      else if(BttnNum == 4){ //4th button is pressed
        buttons[4] = 1; //button array
        Snooze ^= 1;
      }
      else if(BttnNum == 3){ //3rd button is pressed
        buttons[3] = 1; //button array
        //Volumeup ^= 1;
      }
      else if(BttnNum == 2){ //2nd button is pressed
        buttons[2] = 1; //button array
        //Volumedown ^= 1;

      }
    }
  }
  //reset button state
  int i;
  for (i=0;i<=8;i++){
    buttons[i] = 0;
  }
}

/*************************************************************************/
//                           CLKBounds()
//Used to bound block limits. When we edit time with encoders, we want to make
//sure that they don’t go over 59 minutes, and that it stays bounded to 24 hrs
/*************************************************************************/
void CLKBounds(){
  //If minutes is set to be 60+
  if(minutes>59){
    minutes = 0;
    hours++;
    if(hours > 23){
      hours == 0;
    }
  }
  //If hours is set to be 24+
  if(hours > 23){
    hours = 0;
  }

  //If we decrease past 0 hrs
  if(hours<0){
    hours = 23; //loop back to 23
  }

  //If we decrement minutes past 0 mins
  if(minutes < 0){
    minutes = 59; //warp back to 59
    hours--; //decrement down by one hour

    //If hours is < 0
    if(hours<0){
      hours = 23; //Go back to 23
    }
  }
}

/*************************************************************************/
//                           AlarmBounds()
//This performs the same exact task as CLKBounds(), but for the alarm clock.
//This way the alarm stays bounded
/*************************************************************************/
void AlarmBounds(){
  //If minutes is set to be 60+
  if(AlarmMins>59){
    AlarmMins = 0;
    AlarmHrs++;
```

```c
    if(AlarmHrs > 23){
      AlarmHrs == 0;
    }
  }
  //If hours is set to be 24+
  if(AlarmHrs > 23){
    AlarmHrs = 0;
  }

  if(AlarmHrs<0){
    AlarmHrs = 23;
  }
  if(AlarmMins < 0){
    AlarmMins = 59;
    AlarmHrs--;
    if(AlarmHrs<0){
      AlarmHrs = 23;
    }
  }
}

/*************************************************************************/
//                           bargraph_updater
//Used to update bargraph values with inc/decrement value
//The scalar inc/dec value will be displayed in binary
/*************************************************************************/
void bargraph_updater(){
  uint8_t output = 0; //what the bargraph will display
  if(ChangeTime == 1){ //when both buttons are pressed do nothing
    output = 0b00000001;
  }
  else if(ChangeAlarmTime == 1){ //increment/decrement by 1
    output = 0b00000010;
  }
  //commented out because the armed
  // else if(Set_Alarm == 1){//increment/decrement by 2
  //   output = 0b00000100;
  // }

  //Commented Out because Snooze should be on LCD display
  // else if(Snooze == 1){//increment/decrement by 4
  //   output = 0b00000100;
  // }]
  else if(Volumeup == 1){//increment/decrement by 4
    output = 0b00001000;
  }
  else if(Volumedown == 1){//increment/decrement by 4
    output = 0b00010000;
  }
  else{
    output = 0b00000000;
  }

  /* Start transmission */
  SPDR = output;
  while (bit_is_clear(SPSR,SPIF)){} //spin till SPI data has been sent

  PORTD |= (1<<PD4);        //send rising edge to regclk on HC595
  PORTD &= ~(1<<PD4);            //send falling edge to regclk on HC595
}

/*********************************************************************/
//
                                Encoder_Data
//Toggles SHIFT_LN_N on parallel shift register to get data into the flip flops
//Sets CLK_INH to low so we can read from QH.
//Remember: Most significant bit is at position H
/*********************************************************************/
void Encoder_Data(){
```

```c
      int i;
      //Remember: PE6-> SHIFT_LN_N and PE7-> CLK_INH
      //Toggle SH_LD to get their values into the flip flops
      PORTE ^= (1<<PE6);
      PORTE ^= (1<<PE6);

      //Output to through QH by changing CLK_INH
      PORTE ^= (1<<PE7);//CLK_INH
      raw_encoder = spi_read();

      //Stop the output
      PORTE ^= (1<<PE7);//CLK_INH

      //left Encoder
      currL_Encoder = raw_encoder;
      // get rid of LHS bits
      // what we want: 0bxx
for (i=7; i>1; i--) {
  currL_Encoder &= ~(1<<i);
}

      //Right encoder
      currR_Encoder = (raw_encoder>>2);
      //get rid of LHS bits
      // what we want (0bxx)
      for (i=7; i>1;i--) {
              currR_Encoder &= ~(1<<i);
      }

      //If it's a first time start-up
      if(start_flag == 0){
              prevL_Encoder = currL_Encoder;//set them equal
              prevR_Encoder = currR_Encoder;//set them equal
              start_flag = 1;
      }

// --------- LEFT ENCODER ----------//
if(currL_Encoder == 0b11 && prevL_Encoder == 0b01){
  if(ChangeTime){
    hours += 1;
    seconds = 0;
  }
  if(ChangeAlarmTime){AlarmHrs += 1;}
  prevL_Encoder = currL_Encoder;
}
else if(currL_Encoder == 0b11 && prevL_Encoder == 0b10){
  if(ChangeTime){
    hours -= 1;
    seconds = 0;
  }
  if(ChangeAlarmTime){AlarmHrs -= 1;}
  prevL_Encoder = currL_Encoder;
}
else{
  prevL_Encoder = currL_Encoder;
}
// --------- RIGHT ENCODER ----------//
if(currR_Encoder == 0b11 && prevR_Encoder == 0b01){
  if(ChangeTime){
    minutes += 1;
    seconds = 0;
  }
  if(ChangeAlarmTime){AlarmMins += 1;}
  prevR_Encoder = currR_Encoder;
}
else if(currR_Encoder == 0b11 && prevR_Encoder == 0b10){
  if(ChangeTime){
    minutes -= 1;
    seconds = 0;
    }
    if(ChangeAlarmTime){AlarmMins -= 1;}
    prevR_Encoder = currR_Encoder;
  }
  else{
    prevR_Encoder = currR_Encoder;
  }
  //Make sure the alarm time and clock time are bounded to military time
  CLKBounds();
  AlarmBounds();
}

/*********************************************************************/
//
        segMapper(uint8_t val)
//This is used to map our desired digit to the binary value that displays it on
//the 7-segment display
/*********************************************************************/
uint8_t segMapper(uint8_t val){
  uint8_t mapped_val;
  mapped_val = dec_to_7seg[val];
  return mapped_val;
}

/*********************************************************************/
//                                                                       TimedigP
arser(uint8_t hrs, uint8_t mins)
//This is used to parse hours and minutes into BSD and store it in the segment
//data array, which will then be used to output on the 7-seg
/*********************************************************************/
void TimedigParser(uint8_t hrs, uint8_t mins){
  uint8_t mins_OnesVal;
  uint8_t mins_TensVal;
  uint8_t hrs_OnesVal;
  uint8_t hrs_TensVal;

  //minutes
  mins_OnesVal = mins % 10;
  segment_data[0] = segMapper(mins_OnesVal);

  mins_TensVal = (mins/10) % 10;
  segment_data[1] = segMapper(mins_TensVal);

  //hours
  hrs_OnesVal = hrs % 10;
  segment_data[3] = segMapper(hrs_OnesVal);

  hrs_TensVal = (hrs/10) % 10;
  segment_data[4] = segMapper(hrs_TensVal);
}
void AlarmSetLED(){
  //DDRA = 0xFF;
  asm volatile("nop");
  asm volatile("nop");

  //if the alarm is on, set LED
  if(AlarmOnOff){

  }
  //if the alarm is off, dim LED off
  else{

  }
}
/*********************************************************************/
//                                                                       TimedigP
arser(uint8_t hrs, uint8_t mins)
//The sole purpose is so that it allows us to edit the right digit based on
//which select value is passed
```

```
/*********************************************************************/
void SevnSgDisp(uint8_t select){
  DDRA = 0xFF;
  //AlarmSetLED();
  //Adjust the select bits
  if(select == 0){ //first digit
    PORTB = 0x00;
  }
  else if(select == 1){ //second digit
    PORTB = 0x10;
  }
  else if(select == 2){ //colon
    PORTB = 0x20;
  }
  else if(select == 3){ //third digit
    PORTB = 0x30;
  }
  else if(select == 4){ //4th dig
    PORTB = 0x40;
  }

  //Send values to display
  PORTA = segment_data[select];
}

//*******************************************************************************
*****
// Function Name:void AllSegments_BitClearer
// This function is put to clear previous digit values on the seven segment disp
lay.
// Goal: The goal is to avoid ghosting and help set un-used segments to zero.
//*******************************************************************************
void AllSegments_BitClearer(){
        DDRA = 0xFF;
  asm volatile("nop");
  asm volatile("nop");

        //Ones
        PORTB = 0x00;
        PORTA = 0b11111111;
        _delay_ms(1);

        //Tens
        PORTB = 0x10;
        PORTA = 0b11111111;
        _delay_ms(1);

        //Hundreds
        PORTB = 0x30;
        PORTA = 0b11111111;
        _delay_ms(1);

        //Thousands
        PORTB = 0x40;
        PORTA = 0b11111111;
        _delay_ms(1);
}


//*******************************************************************************
*****
//This ISR is used to keep track of secends that passed. Within this function,
//we also read out encoders, implement our dimming function, and read from our
//adc.
//*******************************************************************************
ISR(TIMER0_OVF_vect){
  static uint8_t OneSecTempCount=0;
  OneSecTempCount++;
  if((OneSecTempCount % 128) == 0){
    seconds ++;

    //colon handler
    colon ^= 0x01;
    if(colon == 0x01){
      segment_data[2] = 0b11111100;
    }
    else{
      segment_data[2] = 0b00000111;
    }

    //If seconds is 60
    if(seconds == 60){
      minutes++; //then increase minutes
      seconds = 0; //and reset seconds

      //check to see if minute is 60
      if(minutes == 60){
        hours++; // increment the hour
        minutes = 0; // reset minutes

        // check to see hours
        if(hours == 24){
            hours = 0; //then it’s back to start the day at 0 hours
        }
      }
    }
    //handle snooze count if enabled
    if(Snooze == 1){
      SnoozeSecCounter++;
    }
    //read temptr sensor data
    temptrSens();
    LCDUpdater();
  }
  //Handle LCD
  //LCDUpdater();
  // if(buttonsToggled){
  //   LCDUpdater();
  //   buttonsToggled = 0;
  // }
  //Used for brightness adjusting
  if((OneSecTempCount % 32) == 0){
    adc_read();
  }
  Encoder_Data();
  refresh_lcd(lcd_draft);
}
//******************************************************************************
*****
//This ISR is used to toggle the pin that will generate our tone
//******************************************************************************
ISR(TIMER1_COMPA_vect)
{
        PORTD ^= (1<<PD5);
}
ISR(TIMER2_OVF_vect){
}

void temptrSens(){
  prev_lm73_temp = lm73_temp;
  lm73_temp = get_rawData();
  //call function that perform the rest of the operations

  if(lm73_temp != prev_lm73_temp){
    //Display in Farenheit
    lm73_temp_convert(lcd_string_F,lm73_temp,1);
    // set_cursor(2,0);
```

```
    // string2lcd(lcd_string_F);

    //Display in Celcius
    lm73_temp_convert(lcd_string_C,lm73_temp,0);
    // set_cursor(2,7);
    // string2lcd(lcd_string_C);

    //cursor_home();//put the cursor back
  }
}

//**************************************************************************/
//
                        Main
//**************************************************************************/
int main(){
  DDRD |= (1<<PD4) | (1<<PD5);
  DDRB |= 0xF0; //set port bits 4-7 B as outputs
  DDRC |= (1<<PC0);
  DDRE = ((1<<PE7) | (1<<PE6)); //Outputs for CLK_INH and SHIFT_LN_N
  tcnt0_init();  //initalize counter timer zero
  tcnt1_init();
  tcnt2_init(); //Diming initializer
  spi_init();    //initalize SPI port
  adc_init(); // adc initializer
  lcd_init(); //lcd initializer
  PORTE = ((1<<PE7) | (1<<PE6)); //By default, disable CLK_INH (don’t want an ou
tput to QH yet) and SH/LD (active low)

  init_twi();
  sei();

  uint8_t digSel=0x00;


  //set LM73 mode for reading temperature by loading pointer register
  lm73_wr_buf[0] = 0;//load lm73_wr_buf[0] with temperature pointer address
  twi_start_wr(LM73_ADDRESS,lm73_wr_buf,1);//start the TWI write process
  //^the address value for "LM73_ADDRESS" is in lm73_functions.h
  _delay_ms(2); //wait for the xfer to finish

  clear_display();
  while(1){
    _delay_ms(1);
    Read_ButtonsV2();
    // ------- display on seven segment ------- //
    //If we aren’t changing alarm time, then display regular time
    if(ChangeAlarmTime != 1){
      TimedigParser(hours, minutes);
    }

    //If we are changing alarm time, show the alarm time on 7-seg
    else if(ChangeAlarmTime == 1){
      TimedigParser(AlarmHrs, AlarmMins);
    }
    if(digSel>4){
      digSel = 0;
    }
    SevnSgDisp(digSel);
    digSel++;
  // ------------------------------------------- //
  bargraph_updater();
  AlarmHandler();
  }
}
```