```
//Author: Jose Manuel Lopez Alcala
//ID:932503918
//ECE 473 Lab5
//Fall 2019
#include <avr/io.h>
#include <util/delay.h>
#include <avr/interrupt.h>
#include <math.h>
#include "hd44780.h"
#include <stdlib.h>
#include "twi_master.h"
#include "lm73_functions.h"
#include <string.h>
#include "uart_functions.h"

//Bit Macros
#define BIT0 0
#define BIT1 1
#define BIT2 2
#define BIT3 3
#define BIT4 4
#define BIT5 5
#define BIT6 6
#define BIT7 7

//Decoder Macros
#define SH_LD 6
#define CLK_INH 7

//Encoder 1 values
volatile uint8_t encoder1_past = 0;
volatile uint8_t encoder1_present =0;
//Encoder 2 value
volatile uint8_t encoder2_past = 0;
volatile uint8_t encoder2_present = 0;
//seconds counter variable
volatile uint8_t sec_count=0;
//minutes counter variable
volatile int8_t min_count=0;
//Military time hours counter variable
volatile int8_t hrs_mil=0;
//Standard time hours counter variabble
volatile int8_t hrs_std =12;
//Bool value for military time vs standard (1=standard/0=military)
volatile uint8_t mil_std = 0x00;
//Parsed time variables
volatile uint8_t min_ones =0;
volatile uint8_t min_tens =0;
volatile uint8_t hrs_ones =0;
volatile uint8_t hrs_tens =0;
//holds data to be sent to the segments. logic zero turns segment on
volatile uint8_t segment_data[5];
//hold adc result
volatile uint16_t adc_result;
//Mode Variables
volatile uint8_t mode =0;
volatile uint8_t inc_dec1 = 1;//for decoder 1
volatile uint8_t inc_dec2 = 1;// for decoder 2
//dummy counter variable(for loop) in update_mode()
volatile uint8_t n = 0;
//PORTC previous values
volatile uint8_t PORTC_previous = 0;
//PORTA previous value
volatile uint8_t PORTA_previous = 0;
//bit counter varibale for "cleaning" encoder data
volatile uint8_t bitn = 0;
//reset bool variable
volatile uint8_t reset =1;// true when first start up
//encoder value
```



```
volatile uint8_t encoder_val = 0;
//Alarm Variables
volatile int8_t a_hrs_mil = 0;//initialize to 7
volatile int8_t a_hrs_std = 0;//initialize to 7
volatile int8_t a_min_count = 1;//initialize to 0
volatile uint8_t a_min_ones =0;
volatile uint8_t a_min_tens =0;
volatile uint8_t a_hrs_ones =0;
volatile uint8_t a_hrs_tens =0;
//String for lcd
char lcd_string_on[16] = "ALARM";
//variable to hold snooze time in seconds
volatile uint16_t snooze_time =10;
//variable that indicates snoozing
uint8_t snoozing = 0;
//Alarm sound on variable
volatile uint8_t alarm_sound = 0;
//Update LCD flag
volatile uint8_t lcd_flag =0;
//OCR2 value holder
volatile int16_t ocr2_value = 0;
//Array to hold Temperature reading
volatile char lcd_str_temperature[16];
//Variable hold the Temperature reading
volatile uint16_t lm73_temp;
//flag that signals 1 second tick
uint8_t second_tick = 0;
//Array that hold the entire screen
char lcd_full_array[32]="                ";// emtpy screen
//Counter variable for update_lcd_temp()
uint8_t t_counter = 0;
//Variable to hold length of Temerature Reading
uint8_t length = 0;
//Character variable to hold incoming data
char rx_char;
//Array to hold in coming data
char rx_array[16];
//USART receive data complete flag
volatile uint8_t rcv_rdy=0;
//Char arry for debugging
char debug_array[16];


//*********************************************************************************
//                            debounce_switch
//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.
//
uint8_t debounce_switch(uint8_t pin) {
  static uint16_t state[8] = {0};
  state[pin] = (state[pin]<<1)|(!bit_is_clear(PINA,pin))|0xE000;
  if (state[pin]==0xF000) {
    return 1;
  }
  return 0;
}
/*************************************************************
Function: update_lcd_alarm()
Description:
  This function will update the LCD display based on whether
  or not the alarm is armed. EXCLUSIVELY FOR ALARM
Parameters: NONE
Return:void
*************************************************************/
void update_lcd_alarm(){
  if (((mode>>BIT3)&1)==1) {//Check Alarm EN Bit
```

```
    lcd_full_array[0]= 'A';
    lcd_full_array[1]= 'L';
    lcd_full_array[2]= 'A';
    lcd_full_array[3]= 'R';
    lcd_full_array[4]= 'M';
  }else{//clear Alarm message
    lcd_full_array[0]= ' ';
    lcd_full_array[1]= ' ';
    lcd_full_array[2]= ' ';
    lcd_full_array[3]= ' ';
    lcd_full_array[4]= ' ';
  }
  lcd_flag = 0;//reset flag for ALARM signaling

  // cursor_home();
  // if (((mode>>BIT3)&1)==1) {//Check Alarm EN bit
  //   string2lcd(lcd_string_on);//write alarm to the LCD
  // }else{
  //   string2lcd("     ");
  // }
  // lcd_flag = 0;//reset flag for ALARM signaling
}


//*********************************************************************
//                          init_tcnt0
//*********************************************************************
//initalize timer/counter zero to normal mode
void init_tcnt0(){
  ASSR  |= (1<<AS0);//run off external 32kHz osc
  TIMSK |= (1<<TOIE0);//overflow interrup
  TCCR0 |= (1<<CS00);//Normal mode, no prescale
}



//*********************************************************************
//                          init_tcnt2
//*********************************************************************
void init_tcnt2(){
  //Fast PWM, Non-inverting mode on OC2(PB7),CLKio/1024 prescale, uC clock
  TCCR2 |= (1<<WGM21)|(1<<WGM20)|(1<<COM21)|(1<<COM20)|(0<<CS20)|(0<<CS21)|(1<<C
S22);
  TIMSK |= (1<<TOIE2);//overflow interrupt
  OCR2  |= 127;//initalize to half of the total scale (0-255)

}

//*********************************************************************
//                          init_tcnt3
//*********************************************************************
void init_tcnt3(){
  //Non-inverting mode //FAST PWM, 8-bit, no prescale
  TCCR3A |= (1<<COM3A1)|(0<<COM3A0)|(0<<WGM31)|(1<<WGM30);
  TCCR3B |= (0<<WGM33)|(1<<WGM32)|(1<<CS30);
  ETIMSK |= (1<<TOIE3);//overflow interrup
  OCR3A =  127;//Vout = OCR3A*(0.0198)+0.022
}


//*********************************************************************
//                          init_tcnt1
//*********************************************************************
void init_tcnt1(){
  //CTC mode,no prescale, Normal port operation
  TCCR1B |= (1<<WGM12)|(1<<CS10);
  TIMSK  |= (1<<OCIE1A);// Output compare A match
  OCR1A   = 0x1F3F;//7999 for 1Khz wave
  //OCR1A   = 0xF9F;//3999 for 2Khz wave
```



```
  //OCR1A   = 0x3E7F;//15999 for 500Khz wave

}
/*************************************************************************
Function: read_LM73()
Description:
  This function will get a reading from the LM73 Temperature sensor and then
  update the array the holds the ASCII equivalent. This array is the one that
  is used to display the ASCII in the LCD display
Parameters: NONE
Return: VOID
*************************************************************************/
void read_LM73(){
  static uint8_t fc_toggle = 0;//1=Farenheit, 0=Celcius
  fc_toggle ^= (1<<BIT0);//Toggle the value
  lm73_temp = read_temperature();
  lm73_temp_convert(lcd_str_temperature,lm73_temp,fc_toggle);
}

// //*********************************************************************
// //                        ISR for timer counter one
// //*********************************************************************
ISR(TIMER1_COMPA_vect){
  static uint16_t count_isr1 =0;
  static uint16_t count_isr12=0;
  count_isr12++;
  count_isr1++;

  if (lcd_flag ==1 ) {//only update when bit toggled
    update_lcd_alarm();
  }

  if (((mode>>BIT3)&1) == 1) {// Check Alarm enabled bit
    if ((hrs_mil == a_hrs_mil)&&(min_count == a_min_count)) {//current time matc
hes alarm time
      alarm_sound = 1;//Alarm sound on
    }

    if ((((mode>>BIT4)&1)==1)&&(alarm_sound==1)) {//check Snooze enabled bit and
 alarm is on
      snoozing =1;//activate snoozing
      mode &= ~(1<<BIT4);//clear snooze bit
      count_isr1 = 0;//reset count ready for snoozing
    }

    if (snoozing == 1) {//system is snoozing
      alarm_sound = 0;//Alarm sound off
      if (count_isr1 == 2000) {//1 second tick
        snooze_time--;//decrement snooze time
        count_isr1 =0;//reset count
        if (snooze_time==0) {
          alarm_sound = 1;//Alarm sound ON
          snooze_time = 10;//reset snooze time
          snoozing    = 0;//reset snoozing
        }
      }
    }

    if (alarm_sound==1) {
      PORTD ^= (1<<BIT2);//toggle Pin 0 on Port d for alarm sound
    }
  }else{
    mode &= ~(1<<BIT4);// clear snooze if Alarm DISABLED
    alarm_sound=0;//alarm souund off
  }
}
```

```
/*************************************************************
Function: update_time()
Description:
  This function will update the time based on the current second
  count
Parameters: NONE
Return: void
*************************************************************/
void update_time() {
  if (sec_count == 59) {
    sec_count =0;//reset seconds
    if (min_count==59) {
      min_count=0;//reset mins
      if (hrs_mil==23) {
        hrs_mil =0;//reset military time
      }else{
        hrs_mil++;//increment military time
      }
      if (hrs_std == 13) {
        hrs_std =1;//reset standard time
      }else{
        hrs_std++;//increment standard time
      }
    }else{
      min_count++;//increment min count
    }
  }else{
    sec_count++;//increment seconds
  }
}

/***************Function: decoder()*************************
Description:
  Takes in an 8 bit decimal and returns the binary equivalent
  that will be used to display on the seven segement.
  Format:(DP)GFEDCBA
Parameters:
  - uint8_t decimal: This is the decimal that needs to be converted
Precondition: NONE
Return: uint_t binary equivalent
*********************************************************************/
uint8_t decoder(uint8_t decimal)
{
  if (decimal == 0) {//decimal: 0
    return 0b11000000;
  }else if (decimal == 1) {//decimal: 1
    return 0b11111001;
  }else if (decimal == 2) {//decimal: 2
    return 0b10100100;
  }else if (decimal == 3) {//decimal: 3
    return 0b10110000;
  }else if (decimal == 4) {//deciaml: 4
    return 0b10011001;
  }else if (decimal == 5) {//decimal: 5
    return 0b10010010;
  }else if (decimal == 6) {//deciaml: 6
    return 0b10000010;
  }else if (decimal == 7) {//decimal: 7
    return 0b11111000;
  }else if (decimal == 8) {//decimal: 8
    return 0b10000000;
  }else if (decimal == 9) {//decimal: 9
    return 0b10010000;
  }
  //should never reach this return
  return 0b11111111;// All segments OFF
}

/*************************************************************
```



```
Function: adc_init()
Description:
  This function will setup the Analog-to-digital conveter
Parameters: NONE
Return: void
*************************************************************/
void adc_init(){
  DDRF  &= ~(_BV(DDF7));//make port F bit 7 is ADC input
  PORTF &= ~(_BV(PF7));// port F bit 7 pullups must be off
  ADMUX |= (1<<REFS0)|(1<<MUX2)|(1<<MUX1)|(1<<MUX0) ;//single-ended, input PORTF
 bit 7, right adjusted, 10 bits
  ADCSRA|= (1<<ADEN)|(1<<ADPS2)|(1<<ADPS1)|(1<<ADPS0);//ADC enabled, division fa
ctor 128
}

/*************************************************************
Function: adc_read()
Description:
  This function will read the adc and return a 16 bit adc result
Parameters: NONE
Return:
  * 16 bit unsigned  adc result
*************************************************************/
uint16_t adc_read(){
  ADCSRA |= (1<<ADSC);//Start ADC
  while (bit_is_clear(ADCSRA,ADIF)){}//wait for ADC to finish
  ADCSRA |= (1<<ADIF);//clear flag by writing one
  return ADC;
}


/*************************************************************
Function: decode_time()
Description:
  Parases the value of time to the BCD segment code in the array
  segment_data for display. It also takes care of the PM/AM signaling
  by setting L3 LED on (bit 2 low ). Lastly, it also shows the
  "ALARM" enable on the DP of mins one segment
Parameters: NONE
Return: void
*************************************************************/
void decode_time()
{

  // Parse Clock time to segment array
  if (((mode>>BIT1)&1)==0) {
    min_ones = (min_count%10);//parsed ones place of minutes
    min_tens = (min_count/10)%10;//parsed tens place of minutes

    if (mil_std == 0) {//military time
      hrs_ones = (hrs_mil%10);//parse ones
      hrs_tens = (hrs_mil/10)%10;//parse tens
      segment_data[2] |= (1<<BIT2);//AM indicator
    }else{//standard time
      hrs_ones = (hrs_std%10);//parse ones
      hrs_tens = (hrs_std/10)%10;//parse tens
      if (hrs_mil>=12) {
        segment_data[2] &= ~(1<<BIT2);//PM indicator
      }else{
        segment_data[2] |= (1<<BIT2);//AM indicator
      }
    }

    //decoded numbers to array
    segment_data[0] = decoder(hrs_tens);//hours tens
    segment_data[1] = decoder(hrs_ones);//hours ones
    segment_data[3] = decoder(min_tens);//minutes tens
    segment_data[4] = decoder(min_ones);//minutes ones
  }else if (((mode&1)==0) && (((mode>>BIT1)&1)==1)) {//parse Alarm time to segem
```

```
t array
    a_min_ones = (a_min_count%10);//parsed ones place of minutes
    a_min_tens = (a_min_count/10)%10;//parsed tens place of minutes

    if (mil_std == 0) {//military time
      a_hrs_ones = (a_hrs_mil%10);//parse ones
      a_hrs_tens = (a_hrs_mil/10)%10;//parse tens
      segment_data[2] |= (1<<BIT2);//AM indicator
    }else{//standard time
      a_hrs_ones = (a_hrs_std%10);//parse ones
      a_hrs_tens = (a_hrs_std/10)%10;//parse tens
      if (a_hrs_mil>=12) {
        segment_data[2] &= ~(1<<BIT2);//PM indicator
      }else{
        segment_data[2] |= (1<<BIT2);//AM indicator
      }
    }

    //decoded numbers to array
    segment_data[0] = decoder(a_hrs_tens);//hours tens
    segment_data[1] = decoder(a_hrs_ones);//hours ones
    segment_data[3] = decoder(a_min_tens);//minutes tens
    segment_data[4] = decoder(a_min_ones);//minutes ones
  }
  if (((mode>>BIT3)&1)==1) {
    segment_data[4] &= ~(1<<BIT7);//ALARM EN indicator
  }else{
    segment_data[4] |= (1<<BIT7);//ALARM DISABLE
  }

}

// /*************************************************************
// Function: uart_tx_string()
// Description:
//   This function sends a string over UART
// Parameters:
//   - const char *str:
//     String that needs to be transmitted
// Return:VOID
// *************************************************************/
// void uart_tx_string(const char *str)
// {
//   uart_puts(str);//send string
//   uart_putc('\0');//send terminating character
// }

/*************************************************************
Function: update_lcd_temp()
Description:
  This function will update the LCD with the temperature readings
  from the remote and local sensors.
Parameters: NONE
Return:VOID
*************************************************************/
void update_lcd_temp(){
  //Remote Temperature Reading************************
  //USART ISR will put data in rx_array
  if (rcv_rdy==1) {
    rcv_rdy=0;//reset flag
    length = strlen(rx_array);//length of usart incoming buffer
    for (t_counter = 0; t_counter <= (length-1); t_counter++) {
      // clear_display();//for testing
      // cursor_home();//for testing
      // char2lcd(rx_array[t_counter]);// for testing
      // _delay_ms(1000);//for testing
      lcd_full_array[t_counter+24]=rx_array[t_counter];
    }
    lcd_full_array[length+24]=' ';//blank after number
```



```
  }
  //request data
  uart_putc('s');
  uart_putc('\0');

  //Local Temperature Reading************************
  read_LM73();
  length = strlen(lcd_str_temperature);
  for (t_counter = 0; t_counter <= (length-1); t_counter++) {
    lcd_full_array[t_counter+16]=lcd_str_temperature[t_counter];
  }
  lcd_full_array[length+16] = ' ';//makes a space afer writing numbers

}

// //*************************************************************************
// //                        ISR fo USART0
// //*************************************************************************
ISR(USART0_RX_vect){
static  uint8_t  i;
  rx_char = UDR0;              //get character
  rx_array[i++]=rx_char;  //store in array
 //if entire string has arrived, set flag, reset index
  if(rx_char == '\0'){
    rcv_rdy=1;
    i=0;
  }
}


//*************************************************************************
//                        ISR for timer counter zero
//*************************************************************************
ISR(TIMER0_OVF_vect)
{
  static uint8_t count_isr = 0;
  count_isr++;

  if (count_isr==250) {
    update_lcd_temp();//get reading from temp sensors
  }

  if ((count_isr%1)==0) {
    refresh_lcd(lcd_full_array);//update lcd
  }

  if ((count_isr % 128) == 0) {
    update_time();//update seconds, minutes and hours
    segment_data[2] ^= (1<<BIT0)|(1<<BIT1);//Toggle colon
    second_tick = 1;//set flag

  }

}

/*************************************************************
Function: update_inc_dec()
Description:
  This function will update the increase and decrese value according
  to the mode that has been selected by the user.
Parameters: NONE
Return:void
*************************************************************/
void update_inc_dec(){
  if (mode == 0) {// increas/decrease 0(default)
    inc_dec1 = 0;
    inc_dec2 = 0;
  }else if ( ((mode&1)==1) && (((mode>>1)&1)==0) ) {//set clock time
    inc_dec1 = 1;
```

```
    inc_dec2 = 1;
  }else if ( ((mode&1)==0) && (((mode>>1)&1)==1) ) {//set alarm time
    inc_dec1 = 1;
    inc_dec2 = 1;
  }else if ( ((mode&1)==1) && (((mode>>1)&1)==1) ) {//both modes are set
    inc_dec1 = 0;//do nothing
    inc_dec2 = 0;//do nothing
  }
}

/*************************************************************
Function: update_mode()
Description:
  This function will upate the mode of the system by reading the
  calling the debounce_switch() function and then toggling a bit
  in variable that will be used to determine what the user has selected.
  Register arrangment:
  BIT7(0)|BIT6(0)|BIT5(0)|BIT4(0)|BIT3(Alarm 0/1)|BIT2(Mil/STD)|BIT1(Alarm Time)
|BIT0(Clk Time)
Parameters: NONE
Return:void
*************************************************************/
void update_mode() {
  //Reconfigure the PORTS to read from the pushbuttons:
  DDRA = 0x00;//set PORT A to all inputs
  PORTA = 0xFF;//set PORTA all pullups
  PORTC = 0x70;// ENABLE TRI buffer

  for(n = 0; n<=4;n++){//only check button 0,1,2,3,4
    if (debounce_switch(n)) {
      if (n == 0) {//button 0
        mode ^= (1<<BIT0);//Set time (toggle bit 0)
      }else if (n==1) {//button 1
        mode ^= (1<<BIT1);//Set Alarm time (toggle bit 1)
      }else if (n==2) {//button 2
        mode ^= (1<<BIT2);//Toggle Military/Standard
      }else if (n==3) {
        mode ^= (1<<BIT3);//Alarm EN
        lcd_flag =1;//set flag to update lcd
      }else if (n==4) {
        mode |= (1<<BIT4);//Set snooze EN (DON'T Toggle)
      }
    }
  }
  update_inc_dec();//update the inc/dec varible
  PORTC = 0x60;//DISABLE Tri buffer
}

/*************************************************************
Function: spi_read()
Description:
  This function will read the SPI buffer and return the 8bit value
Parameters: NONE
Return:
  uint8_t value that contains the value of SPDR register
*************************************************************/
 uint8_t spi_read(void)
{
  SPDR = 0x00;//Send dummy data  to be able read register
  while (bit_is_clear(SPSR,SPIF)) {}//wait until it is done.
  return(SPDR);//return the value of the read data the came in
}

/*************************************************************
Function: read_encoders()
Description:
  This function will read the input of the encoder and increment
  or decrement the count  accordingly
Parameters: NONE
```



```
Return:void
*************************************************************/
void read_encoders(){
  PORTE ^= (1<<SH_LD);//toggle to low
  PORTE ^= (1<<SH_LD);//toggle back to high
  PORTE ^= (1<<CLK_INH);//toggle to low
  encoder_val= spi_read();
  PORTE ^= (1<<CLK_INH);//toggle to high

  //"clean" encoder1 value
  encoder1_present = encoder_val;
  for ( bitn = 7; bitn > 1; bitn--) {
    encoder1_present &= ~(1<<bitn);
  }
  //"clean" encoder2 value
  encoder2_present = (encoder_val>>2);//RS so that we get rid bit 0:1
  for ( bitn = 7; bitn > 1; bitn--) {
    encoder2_present &= ~(1<<bitn);
  }

  if (reset == 1) {
    encoder1_past = encoder1_present;//set them equal
    encoder2_past = encoder2_present;//set them equal
    reset = 0;//set to zero after first time
  }

  //Update for encoder1 (corse/ every locking position)
  //This updates HOURS
  switch (encoder1_present) {
    case 1://encoder value is 01
      if (encoder1_past == 0) {//past state 00
        encoder1_past = encoder1_present;//set current state as past state
      }else if (encoder1_past == 3 ) {//past state 11
        encoder1_past = encoder1_present;//set current state as past state
      }
      break;
    case 3://encoder value is 11
      if (encoder1_past == 1) {//past state 01
        if ( ((mode&1)==1) && (((mode>>1)&1)==0) ) {//Clock time
          hrs_mil += inc_dec2;//incremetn hours military
          hrs_std += inc_dec2;//increment hours standard
          if (hrs_mil == 24) {
            hrs_mil = 0;//reset military time hours
          }
          if (hrs_std == 13) {
            hrs_std = 1;//reset standard time hours
          }
        }else if (((mode&1)==0) && (((mode>>1)&1)==1) ) {//Alarm time
          a_hrs_mil += inc_dec2;//incremetn hours military
          a_hrs_std += inc_dec2;//increment hours standard
          if (a_hrs_mil == 24) {
            a_hrs_mil = 0;//reset military time hours
          }
          if (a_hrs_std == 13) {
            a_hrs_std = 1;//reset standard time hours
          }
        }
        encoder1_past = encoder1_present;
      }else if (encoder1_past == 2 ) {//past state 10
        if (((mode&1)==1) && (((mode>>1)&1)==0) ) {//clock time
          hrs_mil -= inc_dec2;//decrement hours military
          hrs_std -= inc_dec2;//decrement hours standard
          if (hrs_mil < 0) {
            hrs_mil = 23;//reset military time hours
          }
          if (hrs_std <1) {
            hrs_std = 12;//reset standard time hours
          }
        }else if (((mode&1)==0) && (((mode>>1)&1)==1)) {//Alarm time
```

```
          a_hrs_mil -= inc_dec2;//decrement hours military
          a_hrs_std -= inc_dec2;//decrement hours standard
          if (a_hrs_mil < 0) {
            a_hrs_mil = 23;//reset military time hours
          }
          if (a_hrs_std <1) {
            a_hrs_std = 12;//reset standard time hours
          }
        }
        encoder1_past = encoder1_present;
      }
      break;
    case 2://encoder value 10
      if (encoder1_past == 3) {//past value 11
        encoder1_past = encoder1_present;
      }else if (encoder1_past == 0) {//past value 00
        encoder1_past = encoder1_present;
      }
      break;
    case 0://encoder value 00
      if (encoder1_past == 2) {//past value 10
        encoder1_past = encoder1_present;
      }else if (encoder1_past == 1) {//past value 01
        encoder1_past = encoder1_present;
      }
      break;
    default://nothing has changed
      hrs_mil +=0;//add nothing
      hrs_std +=0;//add nothing
      break;
  }

//Update for encoder2 (corse/ every locking position)
//This updates MINUTES
switch (encoder2_present) {
  case 1://current state is 01
    if (encoder2_past == 0) {//past state 00
      encoder2_past = encoder2_present;
    }else if (encoder2_past == 3 ) {//curent state 11
      encoder2_past = encoder2_present;
    }
    break;
  case 3://curent state 11
    if (encoder2_past == 1) {//past state 01(increse)
      if (((mode&1)==1) && (((mode>>1)&1)==0)) {//clock time
        min_count += inc_dec1;//increment minutes
        if (min_count == 60) {
          min_count = 0;//reset mins if over 59
        }
      }else if (((mode&1)==0) && (((mode>>1)&1)==1)) {//Alarm time
        a_min_count += inc_dec1;//increment minutes
        if (a_min_count == 60) {
          a_min_count = 0;//reset mins if over 59
        }
      }else if(((mode&1)==0)&& (((mode>>BIT1)&1)==0)){//volume control
        OCR3A++;//increase volume
      }
      encoder2_past = encoder2_present;
    }else if (encoder2_past == 2 ) {//past state 10(decrese)
      if (((mode&1)==1) && (((mode>>1)&1)==0)) {//clock time
        min_count -= inc_dec1;//decrement minutes
        if (min_count < 0) {
          min_count = 59;//Zero to 59 mins
        }
      }else if (((mode&1)==0) && (((mode>>1)&1)==1)) {// Alarm time
        a_min_count -= inc_dec1;//decrement minutes
        if (a_min_count < 0) {
          a_min_count = 59;//Zero to 59 mins
        }
```



```
      }else if(((mode&1)==0)&& (((mode>>BIT1)&1)==0)){//volume control
        OCR3A--;//decrease volume
      }
      encoder2_past = encoder2_present;
    }
    break;
  case 2://current state 10
    if (encoder2_past == 3) {//past state 11
      encoder2_past = encoder2_present;
    }else if (encoder2_past == 0) {//past state 00
      encoder2_past = encoder2_present;
    }
    break;
  case 0://curent state 00
    if (encoder2_past == 2) {//past state 10
      encoder2_past = encoder2_present;
    }else if (encoder2_past == 1) {//past state 01
      encoder2_past = encoder2_present;
    }
    break;
  default://nothing has changed
    min_count +=0;//add nothing
    break;
  }


}


//*********************************************************************
//                      ISR for timer counter TWO
//*********************************************************************
ISR(TIMER2_OVF_vect){
  //Store PORT values to be able to restore
  PORTA_previous =PINA;//save PORTA values
  PORTC_previous = PINC;//save PORTC values

  update_mode();//update Bar graph
  mil_std = (mode>>2) & 0x01;//Set to military or standard
  read_encoders();//read encoders

  //restore PORT A and C values
  DDRA = 0xFF;//outside of ISR, always output
  PORTA = PORTA_previous;//restore PORTA
  PORTC = PORTC_previous;//restore PORTC

}

//*********************************************************************
//                            spi_init
//*********************************************************************
void spi_init(void){
  //PORTB ouput: ss(pb0), MOSI(pb2), sclk(pb1)
  DDRB  |= (1<<BIT0)|(1<<BIT1)|(1<<BIT2);
  SPCR   = (1<<SPE) | (1<<MSTR) ; //master mode, clk low on idle, leading edge s
ample
  SPSR   = (1<<SPI2X); //choose double speed operation
 }

/*************************************************************
Function: display_mode()
Description:
  This function will display the mode on the bargraph using the
  SPI protocol
Parameters: NONE
Return:void
*************************************************************/
void display_mode(){
  SPDR = mode;//write value to register
```

```
  while(bit_is_clear(SPSR,SPIF)){}//wait until data is sent
  PORTB ^=(1<<BIT0);//send rising edge to regclk on HC595
  PORTB ^=(1<<BIT0);//send falling edge to reglck on HC595
}

  /*************************************************************
  Function: init_ports()
  Description:
    This function will set the correct bits used for the LCD
    display to work correctly
  Parameters: NONE
  Return:void
  *************************************************************/
  void lcd_ports(){
    DDRF  |= 0x08;  //port F bit 3 is enable for LCD
    PORTF &= 0xF7;  //port F bit 3 is initially low
  }

  // //***********************************************************************
  // //                        ISR for timer counter three
  // //***********************************************************************
  ISR(TIMER3_OVF_vect){
    static uint8_t count_isr3 = 0;
    count_isr3++;

    if ((count_isr3%60)==0) {
      adc_result = adc_read();//read adc
      ocr2_value = 0.457*adc_result-100;//calculate new ocr2
      //bound ocr2_value to 0-255
      if (ocr2_value<10) {
        ocr2_value = 10;
      }else if (ocr2_value>255) {
        ocr2_value = 255;//
      }
      OCR2 = ocr2_value;//set new OCR2
    }

  }



int main()
{
  init_tcnt0();//initalize timer/counter 0
  init_tcnt2();//initalize timer/counter 2
  init_tcnt1();//initialize timer/coutner 1
  init_tcnt3();//initialize timer/counter 3
  adc_init();//initalize AD
  lcd_ports();//initalize LCD ports
  uart_init();//initialize UART
  DDRB |= 0xC0;//OE_EN(pb6), PWM(pb7)
  spi_init();//initialize SPI
  lcd_init();
  init_twi();
  sei();//enable global interruts
  clear_display();
  DDRC = 0x70;//set PORTC(4(SEL0),5(SEL1),6(SEL2)) as output (never changes)
  DDRE = 0xC8;//set PORTE PIN 6,7,3 as ouputs(never changes)
  PORTE =0xC0;//set SH/LD and CLK_INH high (low enabled)
  DDRD = 0x4;//set pin 0 on PORTD as output
  uint8_t digit_sel = 0;//digit select
  segment_data[2] =0xFC;//initalize colon ON

  while (1) {
    _delay_ms(2);
    decode_time();//break up the time to 4, BCD digits in the array
    display_mode();//display mode on bargraph

    if (digit_sel>4) {//bound diigit select 0-4
```



```
      digit_sel =0;
    }
    DDRA = 0xFF;//make PORTA output for Seven Segment
    PORTA = segment_data[digit_sel];//send 7 segment code to LED segments
    //send PORTC the digit to display & power ON to seven seg(bit7=0)
    if (digit_sel == 0) {
      PORTC = 0b01000000;//Hours Tens digit
    }else if (digit_sel==1) {
      PORTC = 0b00110000;//Hours Ones  digit
    }else if (digit_sel==2) {
      PORTC = 0b00100000;//colon digit
    }else if (digit_sel==3) {
      PORTC = 0b00010000;//Minsutes tens digit
    }else if (digit_sel==4) {
      PORTC =0b00000000;//Minutes ones digit
    }
    digit_sel++;//update digit to display(increment)


  }//WHILE
  return 0;
}
```