```
/**********************************************************************
 * main.c
 *
 * Joshua Barksdale
 *
 * This program implements an alarm clock. It outputs to the 7-seg,
 * the bar graph, the LCD screen, and some speakers. It takes input from
 * the buttons, the encoders, and a photocell.
 *********************************************************************/

#include <avr/io.h>
#include <avr/interrupt.h>
#include <util/delay.h>
#include <string.h>
#include <stdlib.h>
#include "hd44780.h"

/* global vars */
//holds data to be sent to the segments. logic zero turns segment on
static uint8_t segment_data[5] = { 0xFF, 0xFF, 0xFF, 0xFF, 0xFF };
//decimal to 7-segment LED display encodings, logic "0" turns on segment
const uint8_t dec_to_7seg[12] = {
    0b11000000, //0
    0b11111001, //1
    0b10100100, //2
    0b10110000, //3
    0b10011001, //4
    0b10010010, //5
    0b10000010, //6
    0b11111000, //7
    0b10000000, //8
    0b10010000, //9
    0b11111111, //blank
    0b01111111  //D.P.
};
volatile uint8_t mode = 0x00;//stores which mode the counter is in
      // BIT  |     0      |    1     |   2   |    3    |
      // MODE | alm_active | snoozing | t_set | alm_set |
//time keeping:
static uint8_t hr     = 0x00;
static uint8_t min    = 0x00;
static uint8_t sec    = 0x00;
static uint8_t col    = 0x00; //keeps track of whether the colon is lit.
//alarm time
static uint8_t alm_hr     = 0x00;
static uint8_t alm_min    = 0x00;
static uint8_t alm_timr  = 0x00; //represents minutes left in alarm
static uint8_t snooze_timr = 0x00; //represents seconds left in snooze
//display control
static uint8_t which_digit = 0;//keeps track of which digit is lit.
static char    lcd_str[32];  //holds string to send to lcd
int16_t adc_result;     //holds adc result
/* function prototypes */
uint8_t debounce();
void segsum(int16_t sum);
int8_t read_encoder();
uint8_t read_buttons();
void init();
void update_globals(uint8_t buttons, int8_t encoders);
void rtc();
void spi_init(void);
void read_adc();

/* function definitions */
/**********************************************************************
*                            spi_init
*
* This function sets up SPI for communication with the LCD screen
*********************************************************************/
```

```
void spi_init(void){
    /* 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

    DDRB  |= 0x07;  //Turn on SS, MOSI, SCLK
    PORTB |= _BV(PB1);  //port B initalization for SPI, SS_n off
    //see: /$install_path/avr/include/avr/iom128.h for bit definitions

    //Master mode, Clock=clk/4, Cycle half phase, Low polarity, MSB first
    SPCR=(1<<SPE) | (1<<MSTR); //enable SPI, clk low initially,
                               // rising edge sample
    SPSR=(1<<SPI2X);           //SPI at 2x speed (8 MHz)
}


/**********************************************************************
*                     timer 0 interrupt service routine
*
* This ISR pulls values from the buttons and encoders.
*********************************************************************/
ISR(TIMER0_COMP_vect){
    rtc(); // keep real time clock ticking
    static uint8_t buttons = 0; //stores input from buttons
    static int8_t encoders = 0; //stores input from encoders
    /* read buttons */
    buttons  = read_buttons();
    /* read encoders */
    encoders = read_encoder();
    update_globals(buttons,encoders);
    read_adc();//reads adc and adjusts brightness accordingly
}

/**********************************************************************
*                     timer 1 interrupt service routine
*
* This ISR makes the alarm beep at the appropriate time
*********************************************************************/
ISR(TIMER1_COMPA_vect){
    if(alm_timr && !(snooze_timr)){
        static uint8_t beep = 0;
        static uint16_t count = 0;
        if(beep){ PORTE ^= (1<<5); }
        count++;
        if(count >= 2000) { count = 0; beep ^= 1; } //toggle beep every
        //half second
    }
}

/**********************************************************************
*                            read_adc
*
* This function reads the voltage of port F bit 0 and uses that value
* to set the output compare register of timer/counter 2. This
* implements auto-dimming of the 7-seg and bar graph.
*********************************************************************/
void read_adc(){
    ADCSRA |= (1<<ADSC);//poke ADSC and start conversion
    while(bit_is_clear(ADCSRA,ADIF));//wait for conversion to finish
    ADCSRA |= (1<<ADIF);//its done, clear flag by writing a one
    adc_result = ADC;//read the ADC output as 16 bits
    adc_result -= 700;//range is 700 (bright) to 960 (dark)
    if(adc_result < 0) {adc_result = 0;}      //bound acd
    if(adc_result > 0xF0) {adc_result = 0xF0;}//result
    OCR2 = adc_result;
}

/**********************************************************************
*                            update_globals
```

```c
* This function takes the most recent value pulled from the buttons and
* encoders and it takes appropriate action based on that input. Mostly
* it changes the value of global variables (hence the name).
***********************************************************************/
void update_globals(uint8_t buttons, int8_t encoders){
    switch(buttons){
        case (1<<0):// set alarm
            if(!(mode & (1<<2))){//if we're not in time set mode
                mode ^= (1<<3);
            }
            break;
        case (1<<1):// arm/disarm alarm
            mode ^= (1<<0);
            if(mode & (1<<0)){ //if alarm is armed
                strcpy(lcd_str, "ALARM ON        ");
            }
            else {
                strcpy(lcd_str, "ALARM OFF       ");
                alm_timr = 0;
                snooze_timr = 0;
                mode &= ~(1<<1);//turn off snooze indicator
            }
            break;
        case (1<<2):// snooze
            if(alm_timr){
                mode |= (1<<1);
                alm_timr = 5;
                snooze_timr = 10;
            }
            break;
        case (1<<3):// set time
            if(!(mode & (1<<3))){//if we're not in alarm set mode
                mode ^= (1<<2);
            }
            break;
    }

    switch(encoders){
        case(1)://hours--
            if(mode & (1<<2)){ //if time set mode
                if(hr == 0) hr = 23;
                else hr--;
            }
            else if(mode & (1<<3)){ //if alarm set mode
                if(alm_hr == 0) alm_hr = 23;
                else alm_hr--;
            }
        break;

        case(2)://hours++
            if(mode & (1<<2)){ //if time set mode
                if(hr == 23) hr = 0;
                else hr++;
            }
            else if(mode & (1<<3)){//if alarm set mode
                if(alm_hr == 23) alm_hr = 0;
                else alm_hr++;
            }
        break;

        case(3)://min--
            if(mode & (1<<2)){ //if time set mode
                if(min == 0) min = 59;
                else min--;
            }
            else if(mode & (1<<3)){ //if alarm set mode
                if(alm_min == 0) alm_min = 59;
                else alm_min--;
            }
```

```c
        break;

        case(4)://min++
            if(mode & (1<<2)){ //if time set mode
                if(min == 59) min = 0;
                else min++;
            }
            else if(mode & (1<<3)){ //if alarm set mode
                if(alm_min == 59) alm_min = 0;
                else alm_min++;
            }
        break;
    }
}

/***********************************************************************
*                          update_globals
* This function is called regularly by an ISR and keeps track of the
* time. It also keeps track of how long the alarm has been going off
* and/or how long the user has been snoozing.
***********************************************************************/
void rtc(){
    static uint16_t count = 0;
    if(! (mode & (1<<2))){//if we're not setting the time currently
        count++;
        if(count >= 511){
            sec++;
            count = 0;
            col ^= 1;
            if(snooze_timr) snooze_timr--;
        }
        if(sec >= 59)   {
            min++;
            sec = 0;
            if(alm_timr) alm_timr--;
        }
        if(min >= 59)   { hr++; min = 0;}
        if(hr >= 23)    { hr = 0; }
        if((mode & 1<<0) && //alarm is armed &&
                alm_hr == hr && //it's time to go off
                alm_min == min &&
                sec == 0){
            alm_timr = 5;//tell the alarm to go off for 5 min
        }
    }
}

/***********************************************************************
*                          read_buttons
* This function takes user input from the button board with a debounce
* and returns it as a uint8_t
***********************************************************************/
uint8_t read_buttons(){
    static uint8_t temp = 0;
    static uint8_t buttons = 0;
    temp = PORTA;//store the displayed digit for later
    //make PORTA an input port with pullups
    DDRA = 0x00; //input
    PORTA = 0xFF; //pullups
    //enable tristate buffer for pushbutton switches
    PORTB |= 0x70;
    //wait for those changes to take place
    asm("nop");
    asm("nop");
    buttons = debounce();//check buttons with debounce, store in temp
    /* put ports A and B back how they were */
    PORTB &= ((which_digit<<4) | 0x8F);//disable tristate buffer, put
                                       //correct digit on the display
    DDRA = 0xFF;  //set PA to output mode
    PORTA = temp; //restore the displayed digit
    return buttons;
}
/***********************************************************************
*                                init
* This function performs all the setup required to run the program. It
* sets port modes and innitial values, enables interrupts, sets up
* timer/counter 0, and sets up serial communication.
***********************************************************************/
void init(){
    /* set up ports */
    DDRA  = 0xFF;    //start port A out as all outputs
    PORTA = 0xFF;    //start with the display off
    DDRB  = ~(1<<3); //MISO (PB3) is an input, everything else is output
    PORTB = (1<<3);  //set pullup resistor on MISO, drive all outputs low
    DDRD  = (1<<2);  //bit 2 is an output
    PORTD = 0;       //all outputs driven low
    DDRE  = (3<<6);  //bits 6 and 7 outputs
    PORTE = 0;       //all outputs driven low
     // alarm outputs //
    DDRE  |= (1<<5);
    DDRC  |= (1<<0);
    PORTC |= (1<<0);

    /* set up timer 0 and interrupt */
    ASSR  |= (1<<AS0); //run off external 32khz osc (TOSC)
    //enable interrupts for output compare match 0
    TIMSK |= (1<<OCIE0);
    TCCR0 |=  (1<<WGM01) | (1<<CS00);  //CTC mode, no prescale
    OCR0  =   63;   //interrupt every 1/(2^9) sec

    /* set up serial communication */
    spi_init();
    /* set up LCD screen */
    lcd_init();
    clear_display();
    cursor_home();
    strcpy(lcd_str, "             ");
    /* set up the ADC */
    DDRF  &= ~(_BV(DDF0)); //make port F bit 0 is ADC input
    PORTF &= ~(_BV(PF0));  //port F bit 0 pullups must be off
    ADMUX = 0b01000000;//single-ended, input PORTF0, R-adjusted, 10 bits
    ADCSRA = 0b10000111;//ADC enabled, don’t start yet, single shot mode
    /* set up timer 2 for being a PWM dimmer */
    // fast pwm, no prescale, non-inverting
    TCCR2 |= (1<<WGM20) | (1<<WGM21) | (1<<CS20) | (1<<COM21);
    //start with 100% duty cycle
    OCR2 = 0x00;
    /* set up timer 1 for being an alarm oscillator*/
    TCCR1A = 0;
    TCCR1B = (1<<WGM12) | (1<<CS12);//CTC mode, 256 prescale
    TIMSK |= (1<<OCIE1A);//interrupt enable on output comapre 1A
    OCR1A = 0x000F; //interrupt 4000 times per second
    TCNT1 = 0;//initialize the TC1 counter to 0
}
/***********************************************************************
*                              debounce
* Debounces 8 input buttons on port A simultaneously. Each bit in the
* return value corresponds to a button. Bit 0 -> far right button. Bit 7
* -> far left button. For a given button press, that button’s bit will
* cycle high only once. Debounce time is the external loop delay * 12.
***********************************************************************/
uint8_t debounce() {
    //holds present state
    static uint16_t b_state[8] = { 0,0,0,0,0,0,0,0 };
    uint8_t ret = 0;//bit 0 is button 0, bit 7 is button 7
    for(uint8_t i=0; i<8; i++){ //for each button
        //advance the state if the buttin is depressed
        b_state[i] = (b_state[i] << 1) | (!bit_is_clear(PINA,i)) | 0xE000;
```

```
        // if state == 12, set the bit representing that button in the
        // return value
        if (b_state[i] == 0xF000) ret |= (1<<i);
    }
    return ret;
}//debounce

/***********************************************************************
 * read_encoder()
 * This function is based on the work of John Main which he published
 * on his site best-microcontroller-projects.com in the article:
 *
 * Rotary Encoder : How to use the Keys KY-040 Encoder on the Arduino
 *
 * It stores each new reading from the encoder in an 8-bit state
 * register, shifting out old readings as it goes. It looks for
 * 2-reading patterns that it recognizes and when it finds them,
 * it returns a number indicating ccw or cw rotation.
 **********************************************************************/
int8_t read_encoder() {
    /* about rot_enc_table:
     * 0 means invalid input, 1 means valid input. You get the index
     * from the input by concatinating the previous and current states
     * of both switches in the encoder */
    static uint8_t rot_enc_table[16] = {0,1,1,0,1,0,0,1,1,0,0,1,0,1,1,0};
    static uint8_t input = 0; // stores the value read form SPDR
    static uint8_t state[2] = {0,0};//stores state of both encoders

    /* scan the register */
    PORTE &= ~(1<<6);//toggle shift/load low
    PORTE |= (1<<6);//toggle shift/load high
    SPDR = 0xFF;//start transmitting a dummy byte to get clock going
    while(bit_is_clear(SPSR, SPIF)){}//wait for the done receiving flag
    input = SPDR;//read value of SPDR

    /* update the state of encoder 1 */
    state[0] = (state[0] << 2);//make room for the latest data
    if (input & (1<<0)) state[0] |= 0x01;//if switch A open, set bit 0
    if (input & (1<<1)) state[0] |= 0x02;//if switch B open, set bit 1
    state[0] = (state[0] & 0x0F); //clear the 2 oldest states

    /* update the state of encoder 2 */
    state[1] = (state[1] << 2);//make room for the latest data
    if (input & (1<<2)) state[1] |= 0x01;//if switch A open, set bit 0
    if (input & (1<<3)) state[1] |= 0x02;//if switch B open, set bit 1
    state[1] = (state[1] & 0x0F); //clear the 2 oldest states

    /* check state 1 for ccw and cw rotation */
    if  (rot_enc_table[state[0]]) { //if  last 2 states indicate movement
        if (state[0] == 0x0B) return 1; //check for end of a ccw click
        if (state[0] == 0x07) return 2; //check for end of a cw click
    }

    /* check state 2 for ccw and cw rotation */
    if  (rot_enc_table[state[1]]) { //if last 2 states indicate movement
        if (state[1] == 0x0B) return 3; //check for end of a ccw click
        if (state[1] == 0x07) return 4; //check for end of a cw click
    }
    return 0;//report no turning
}//read_encoder

/***********************************************************************
*                                init
* This function figures out what to display on the 7-seg based on what
* time it is, when the alarm is set to go off, and what mode the clock
* is in.
***********************************************************************/
uint8_t seg_time(){
    if(mode & (1<<3)){ //if we’re setting the alarm
    /* colon */
    segment_data[2] &= ~(0x03);
    /* hours */
    segment_data[4] = dec_to_7seg[ alm_hr / 10 ];
    segment_data[3] = dec_to_7seg[ alm_hr % 10 ];
    /* minutes */
    segment_data[1] = dec_to_7seg[ alm_min / 10 ];
    segment_data[0] = dec_to_7seg[ alm_min % 10 ];
    }
    else{
    /* colon */
    if(col){ segment_data[2] &= ~(0x03); }
    else{ segment_data[2] |= 0x03; }
    /* hours */
    segment_data[4] = dec_to_7seg[ hr / 10 ];
    segment_data[3] = dec_to_7seg[ hr % 10 ];
    /* minutes */
    segment_data[1] = dec_to_7seg[ min / 10 ];
    segment_data[0] = dec_to_7seg[ min % 10 ];
    }
    return 0;
}

int main(){
    init(); //innitialize ports, timer/counter
    sei();  //enable global interrupts
    for(;;){
        seg_time();
        which_digit++;
        if(which_digit == 5){ which_digit = 0; }
        /* send 7 segment code to LED segments */
        PORTA = segment_data[which_digit];
        /* send PORTB the digit to display */
        PORTB |= (which_digit<<4);
        PORTB &= ((which_digit<<4) | 0x8F);

        /* send current mode to the bar graph */
        cli();
        SPDR = mode; // send mode to the bar graph
        while (bit_is_clear(SPSR, SPIF)){} //spin till SPI data is sent
        PORTD |= (1<<2);  //send rising edge to regclk on HC595
        PORTD &= ~(1<<2);  //send falling edge to regclk on HC595;
        /* update the lcd */
        refresh_lcd(lcd_str);
        sei();
        /* wait a bit before switching to the next digit */
        _delay_ms(2);
    }
}
```