```
// lab3.c
// Cruz M. Solano-Nieblas
// 10.27.21

#define DEBUG

#define TRUE 1
#define FALSE 0
#define SEGNUMS 4
#define COLONPOS 2
#define BUTTONS 8
#include <avr/io.h>
#include <util/delay.h>
#include <avr/interrupt.h>

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

//decimal to 7-segment LED display encodings, logic "0" turns on segment
uint8_t dec_to_7seg[12] = {0xC0, 0xF9, 0xA4, 0xB0, 0x99, 0x92,
                           0x82, 0xF8, 0x80, 0x98, 0xFF, 0x07}; //0, 1, 2, 3, 4,
 5, 6, 7, 8, 9, (blank), (colon blank)

enum encoder_state{IDLE, STATE01, DETENT, STATE10};  // four states for the enco
der. STATE01 and STATE10 are in between IDLE and DETENT states

volatile uint8_t i; //general-purpose counter variable
volatile uint8_t mode; //user interface
volatile uint8_t sum; //this will be used to either increment by 0, 1, 2 or 4
volatile uint16_t display_count = 0; //display count
volatile uint8_t save_portA;
volatile uint8_t save_portB;

//encoder variables
volatile uint8_t encoder_data = 0xFF; //data being read from the encoder pins

//encoder 1
volatile enum encoder_state encoder1 = IDLE; //init encoder1 state
volatile int8_t encoder1_count = 0; //counter to track the encoder1 state machin
e
volatile uint8_t pinA1 = 1, pinB1 = 1, oldPinA1 = 1, oldPinB1 = 1; //hold pin va
lues for encoder1

//encoder 2
volatile enum encoder_state encoder2 = IDLE; //init encoder2 state
volatile int8_t encoder2_count = 0; //counter to track the encoder2 state machin
e
volatile uint8_t pinA2 = 1, pinB2 = 1, oldPinA2 = 1, oldPinB2 = 1; //hold pin va
lues for encoder2

//*************************************************************************
//                            spi_init
//                     Initializes spi operation
//

void spi_init(void){

  DDRB |= (1<<PB0 | 1<<PB1 | 1<<PB2); //output mode for SS, MOSI, SCLK
  SPCR |= (1<<SPE | 1<<MSTR); //master mode, clk low on idle, leading edge sampl
e
  SPSR |= 1<<SPI2X; //choose double speed operation

}//spi_init

//*************************************************************************
//                            spi_read
//          Reads data from MISO pin connected to the encoders

//
```



```
uint8_t spi_read(void){

        PORTE &= ~(1<<PE6); // parallel load encoder pins
        _delay_us(100); //need a delay for buffer to change states and PORTA to
read the buttons
        PORTE |= 1<<PE6; // disable parallel load to enable serial shifting
        _delay_us(100); //need a delay for buffer to change states and PORTA to
read the buttons

        SPDR = 0x00; // dummy transmission to start receive
        while (bit_is_clear(SPSR, SPIF)){} // spin until transmission is complet
e

        return SPDR;


}//spi_read

//*************************************************************************
//                            spi_write
//          Writes data to MOSI pin connected to the bar graph

//

void spi_write(uint8_t data){

        SPDR = data;
        while (bit_is_clear(SPSR, SPIF)){} // spin until transmission is complet
e
        PORTD |= 1<<PD2;
        PORTD &= ~(1<<PD2);

}//spi_write

//*************************************************************************
//                            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 states[8] = {0}; // an array to store the states of all buttons
on the button board
                             // states[0] corresponds to S1 on the board and states[7
] corresponds to S8
states[button] = (states[button]<<1 | (! bit_is_clear(PINA, button)) | 0xE000);
//first extract the bit that corresponds to the button
                                                                            //
then shift the state back to the 1's place
if (states[button] == 0xF000) {return TRUE;}
return FALSE;

}//chk_buttons

ISR(TIMER0_OVF_vect){

        save_portA = PORTA;
        save_portB = PORTB;

        //make PORTA an input port with pullups
        DDRA = 0x00; //inputs
        PORTA = 0xFF; //pullups enabled

        //enable tristate buffer for pushbutton switches
```

```
        PORTB |= 1<<PB4 | 1<<PB5 | 1<<PB6; //decoder outputs logic low DEC7 to a
ctive low tri state buffer

        _delay_us(0.1); //need a delay for buffer to change states and PORTA to
read the buttons
        //now check each button and increment the count as needed

        if (chk_buttons(0)){mode ^= 1<<2;} //toggle the bit on the bar graph tha
t corresponds to the button
        if (chk_buttons(1)){mode ^= 1<<5;} //toggle the bit on the bar graph tha
t corresponds to the button

        //disable tristate buffer for pushbutton switches
//      PORTB &= ~(1<<PB4); //decoder outputs logic high and disables tri state
buffer

        switch (mode){
                case 0x04:
                        sum = 2;
                        break;

                case 0x20:
                        sum = 4;
                        break;

                case 0x24:
                        sum = 0;
                        break;

                default:
                        sum = 1;

        }

        encoder_data = spi_read(); //read encoder pins from spi

        pinA1 = ((encoder_data & 0x01) == 0) ? 0 : 1; //sample pinA1
        pinB1 = ((encoder_data & 0x02) == 0) ? 0 : 1; //sample pinB1

        //encoder1 state machine
        switch (encoder1){
                case IDLE:
                      //check if encoder1 has gone through all states of the sta
te machine
                      if (encoder1_count == 3){
                              display_count += sum;
                      }
                      else if (encoder1_count == -3){
                              display_count -= sum;
                      }
                      encoder1_count = 0;
                      if ((pinA1 != oldPinA1) || (pinB1 != oldPinB1)){ //if move
ment detected
                              if ((pinA1 == 0) && (pinB1 == 1)){ //CW movement
                                      if (oldPinA1 == 1){
                                              encoder1 = STATE01;
                                              encoder1_count++;
                                      }
                              }
                              else if ((pinA1 == 1) && (pinB1 == 0)){ //CCW move
ment
                                      if (oldPinB1 == 1){
                                              encoder1 = STATE10;
                                              encoder1_count--;
                                      }
                              }
                      }
                      break;
```



```
                case STATE01:
                      if ((pinA1 == 0) && (pinB1 == 0)){ //CW movement
                              if (oldPinB1 == 1){
                                      encoder1 = DETENT;
                                      encoder1_count++;
                              }
                      }
                      else if ((pinA1 == 1) && (pinB1 == 1)){ //CCW movement
                              if (oldPinA1 == 0){
                                      encoder1 = IDLE;
                              }
                      }
                      break;

                case DETENT:
                      if ((pinA1 == 1) && (pinB1 == 0)){ //CW movement
                              if (oldPinA1 == 0){
                                      encoder1 = STATE10;
                                      encoder1_count++;
                              }
                      }
                      else if ((pinA1 == 0) && (pinB1 == 1)){ //CCW movement
                              if (oldPinB1 == 0){
                                      encoder1 = STATE01;
                                      encoder1_count--;
                              }
                      }
                      break;

                case STATE10:
                      if ((pinA1 == 1) && (pinB1 == 1)){ //CW movement
                              if (oldPinB1 == 0){
                                      encoder1 = IDLE;
                              }
                      }
                      else if ((pinA1 == 0) && (pinB1 == 0)){ //CCW movement
                              if (oldPinA1 == 1){
                                      encoder1 = DETENT;
                                      encoder1_count--;
                              }
                      }
                      break;

        }//end switch
        oldPinA1 = pinA1;
        oldPinB1 = pinB1;

        pinA2 = ((encoder_data & 0x04) == 0) ? 0 : 1; //sample pinA2
        pinB2 = ((encoder_data & 0x08) == 0) ? 0 : 1; //sample pinB2

        //encoder state machine
        switch (encoder2){
                case IDLE:
                      //check if encoder2 has gone through all states of the sta
te machine
                      if (encoder2_count == 3){
                              display_count += sum;
                      }
                      else if (encoder2_count == -3){
                              display_count -= sum;
                      }
                      encoder2_count = 0;
                      if ((pinA2 != oldPinA2) || (pinB2 != oldPinB2)){ //if move
ment detected
                              if ((pinA2 == 0) && (pinB2 == 1)){ //CW movement
                                      if (oldPinA2 == 1){
                                              encoder2 = STATE01;
                                              encoder2_count++;
                                      }
```

```
                                }
                                else if ((pinA2 == 1) && (pinB2 == 0)){ //CCW move
ment
                                        if (oldPinB2 == 1){
                                                encoder2 = STATE10;
                                                encoder2_count--;
                                        }
                                }
                        }
                        break;

                case STATE01:
                        if ((pinA2 == 0) && (pinB2 == 0)){ //CW movement
                                if (oldPinB2 == 1){
                                        encoder2 = DETENT;
                                        encoder2_count++;
                                }
                        }
                        else if ((pinA2 == 1) && (pinB2 == 1)){ //CCW movement
                                if (oldPinA2 == 0){
                                        encoder2 = IDLE;
                                }
                        }
                        break;

                case DETENT:
                        if ((pinA2 == 1) && (pinB2 == 0)){ //CW movement
                                if (oldPinA2 == 0){
                                        encoder2 = STATE10;
                                        encoder2_count++;
                                }
                        }
                        else if ((pinA2 == 0) && (pinB2 == 1)){ //CCW movement
                                if (oldPinB2 == 0){
                                        encoder2 = STATE01;
                                        encoder2_count--;
                                }
                        }
                        break;

                case STATE10:
                        if ((pinA2 == 1) && (pinB2 == 1)){ //CW movement
                                if (oldPinB2 == 0){
                                        encoder2 = IDLE;
                                }
                        }
                        else if ((pinA2 == 0) && (pinB2 == 0)){ //CCW movement
                                if (oldPinA2 == 1){
                                        encoder2 = DETENT;
                                        encoder2_count--;
                                }
                        }
                        break;

        }//end switch
        oldPinA2 = pinA2;
        oldPinB2 = pinB2;

        DDRA = 0xFF; //make PORTA an output port

        //restore the states of PORTA and PORTB
        PORTA = save_portA;
        PORTB = save_portB;

}//end ISR

//*************************************************************************
//                                segment_sum
```



```
//takes a 16-bit binary input alue 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 segsum(uint16_t sum) {
  //int i; //for loop variable
  //determine how many digits there are
  int digits = 0; //stores the number of digits in sum
  uint8_t digit = 0; //stores a digit in sum
  uint16_t number = sum;
  if (number == 0){digits = 1;}
  else{
  while (number != 0) //divide number out until you get zero
  {
        number /= 10;
        digits++; // increase digits count after every loop iteration

  }

  }

  //break up decimal sum into 4 digit-segments
  for (i = 0; i < digits; i++)
  {
        digit = sum % 10; //extract least significant digit from the sum
        segment_data[i] = dec_to_7seg[digit]; //convert digit to BCD code and st
ore in segment_data array
        sum /= 10; //remove last digit;

  }

  //blank out leading zero digits
  if (digits < SEGNUMS) //if there are less digits than segment numbers
  {
        for (i = digits; i < SEGNUMS; i++)
        {
                segment_data[i] = dec_to_7seg[10]; //blank them out

        }

  }

  //now move data to right place for misplaced colon position
  for (i = SEGNUMS; i > COLONPOS; i--)
  {
        segment_data[i] = segment_data[i-1];

  }
  segment_data[COLONPOS] = dec_to_7seg[11];

}//segment_sum
//*************************************************************************
*****


//*************************************************************************
*****
uint8_t main()
{
//set port A at outputs
DDRA = 0xFF;

//set port B bits 4-7 as outputs
DDRB |= 1<<PB4 | 1<<PB5 | 1<<PB6 | 1<<PB7;
PORTB &= ~(0xF0); //init Port B

// bar graph and encoder init
DDRE |= 1<<PE6;
PORTE |= 1<<PE6;
```

```
spi_init();
DDRD |= 1<<PD2;

//timer counter 0 setup, running off i/o clock
TIMSK |= (1<<TOIE0);              //enable interrupts
TCCR0 |= (1<<CS02) | (1<<CS00);  //normal mode, prescale by 128

sei(); //enable global interrupt flag

while(1){

  spi_write(mode); //display the mode selected to the user

  //bound the count to 0 - 1023
  if (display_count < 0){display_count = 0;}
  else if (display_count > 1023){display_count = 0;}

  //break up the disp_value to 4, BCD digits in the array: call (segsum)
  segsum(display_count);

  //bound a counter (0-4) to keep track of digit to display
  PORTB &= ~(0xF0); //first digit
  for (i = 0; i < SEGNUMS+1; i++)
  {
        PORTA = segment_data[i]; //send 7 segment code to LED segments
        _delay_ms(2);

        //send PORTB the next digit to display
        PORTB += 0x10;

  }


  }//while
}//main
```