```
// OregonState EECS
// Microcontroller System Design
// lab3_code.c
// Joshua Reed
// Oct. 20, 2015

#include <avr/io.h>
#include <avr/interrupt.h>
#include <avr/cpufunc.h>
#include <util/delay.h>

volatile uint8_t flag = 0;

//************************************************************************
*
//                                                          debounce_switch

// Check pushbuttons on PORTA.
// Returns which button was pressed 0-7.
//************************************************************************
*
uint8_t debounce_switch()
{
    uint8_t save_a = DDRA;

    // DDRA to input
    DDRA = 0x00;
    // Enable button board
    PORTB |= (7<<4);

    // Button press shift register
    static uint16_t SR[8] = {0,0,0,0,0,0,0,0};
    uint8_t i = 0;
    uint8_t ret_val = 8;

    for (i=0; i<8; i++)
    {
        // bit_is_clear() returns one when button depressed
        SR[i] = (SR[i] << 1) | bit_is_clear(PINA, i);
        if (SR[i] == 0x000F) { ret_val = i; }
    }


    DDRA = save_a;
    return ret_val;
}

//************************************************************************
*
//                                                          state_check
// Uses a for loop to return the bit number current state.
// If two state bits are set, this will return a 8 to represent a false.
//************************************************************************
*
uint8_t state_check(uint8_t state_reg)
{
    uint8_t i = 0;
    uint8_t state = 10;
    for(i=0; i<8; i++)
    {
        if ((state_reg>>i)&1)
        {
            if (state==10) { state = i; }
            else {state = 8;}
        }
    }
    return state;
}
```



```
//************************************************************************
*
//                                                              to_digs
// Returns an array pointer
// The array is a digit wise separation of numbers 0 to 3.
// For example, passing num as 1234 will result in digs[0] = 4, digs[1] = 2, etc
...
//************************************************************************
*
uint8_t * to_digs(uint16_t num)
{
    static uint8_t digs [4]; // Digit place holder
    static uint8_t sev_seg[11] = { 0b11000000,   // 0 // Seven segment decoder a
rray
                                   0b11111001,   // 1
                                   0b10100100,   // 2
                                   0b10110000,   // 3
                                   0b10011001,   // 4
                                   0b10010010,   // 5
                                   0b10000010,   // 6
                                   0b11111000,   // 7
                                   0b10000000,   // 8
                                   0b10010000,   // 9
                                   0b11111111 }; // off
    // Parse and decode digits
    digs[0] = sev_seg[num       % 10];
    digs[1] = sev_seg[(num/10)  % 10];
    digs[2] = sev_seg[(num/100) % 10];
    digs[3] = sev_seg[(num/1000)    ];
    return digs;
}

//************************************************************************
*
//                                                              set_disp
// Set which digit is displayed.
//************************************************************************
*
void set_disp(uint8_t disp)
{
    static uint8_t decode[5] = { 0,   // 000 disp 0
                                 1,   // 001 disp 1
                                 3,   // 011 colon
                                 4,   // 100 disp 2
                                 2 }; // 101 disp 3
            //if ( temp<8 )
            //{
            //    multiplier = temp;
            //    temp2 = state_check(mult_state);
            //    if ( (temp2 > 7) && ((mult_state>>temp)&1) ) { mult_state = (1
<<temp); }
            //    else {  }

            //}
    PORTB = (decode[disp]<<4) | (PORTB & 0b10001111);
}

//************************************************************************
*
//                                                              read_spi
// Reads encoder and writes to bar graph
// PIN E 7 is sh/ld which is active low to load
// PIN E 6 is clk_inhibit which inhibits the clock when held high
//************************************************************************
*
uint8_t spi_cycle(uint8_t write_val)
{
    PORTE = 0b01111111; // load data
    PORTE = 0b10000000; // set as shift reg and enable the clk
```

```
    SPDR = write_val; // Send data
    while (bit_is_clear(SPSR, SPIF)){} // SPI sreg, sflag
    PORTB |= 0b00000001; // Strobe high
    PORTB &= 0b11111110; // Strobe low
    return SPDR;
}

//***********************************************************************
//                                                            ISR
//
// Executes on TCNTR0 overflow.
// Checks push buttons.
// Checks the rotary encoders via SPI.
// Writes to the bar graph via SPI.
// Increments or decrements cnt variable.
//***********************************************************************
ISR(TIMER0_OVF_vect)
{
    flag = 1;
}

//*************************************************************************
*
//                                                            spi_setup
//
//*************************************************************************
*
void spi_setup()
{
    SPCR |= (1 << SPE) | (1 << MSTR); // SPI ctr reg -- SPI enable, MSTR
    SPSR |= (1 << SPI2X); // SPI status reg -- set clk/2
}

//*************************************************************************
*
//                                                            tcntr_setup
//
//*************************************************************************
*
void tcntr_setup()
{
    TIMSK |= (1 << TOIE0); // Enable interrupts
    TCCR0 |= (1 << CS02) | (1 << CS00); // Normal mode, prescale by 128
}

//*************************************************************************
*
//                                                            main
// Check active low switches on PORTB.
// If low for 4 passes of debounc_switch() increment counter.
// Display number on all four digits of the LED display board.
//*************************************************************************
*
int main()
{
    uint8_t state = 0;
    uint8_t multiplier = 1;
    uint8_t ec_state = 0;
    uint8_t mult_state = 0;
    uint8_t temp = 0;
    uint8_t temp2 = 0;
    uint8_t temp3 = 0;
    uint8_t ec1_curr = 0;
    uint8_t ec1_prev = 0;
    uint8_t ec2_curr = 0;
    uint8_t ec2_prev = 0;
    uint16_t cnt = 0;

    DDRB = 0b11110111; // Set to output except input on spi pin 3
```



```
    DDRE = 0xFF; // Set to output
    DDRA = 0xFF; // Set to output

    sei();
    tcntr_setup();
    spi_setup();

    while(1) // Loop forever
    {
        // Rotate state
        state++;
        state %= 4;

        // Display one digit per cycle
        set_disp(state);
        PORTA = to_digs(cnt)[state];
        _delay_us(15); // delay with the leds on
        PORTA = 0xFF; // turn leds off

        if (flag)
        {
            // Check buttons
            temp = debounce_switch();

            if ( temp<8 ) { mult_state ^= (1<<temp); }
            temp2 = state_check(mult_state);
            if (temp2 > 7) { multiplier = 0; }
            else { multiplier = temp2; }

            ec_state = (ec_state << 4); // state of encoders
            ec_state |= ~spi_cycle(mult_state);
            ec1_prev = (ec_state & 0b00110000)>>4;
            ec1_curr =  ec_state & 0b00000011;
            ec2_prev = (ec_state & 0b11000000)>>6;
            ec2_curr = (ec_state & 0b00001100)>>2 ;
            if (ec1_prev != ec1_curr)
            {
                if ( (ec1_prev == 0b11) && (ec1_curr == 0b01) )
                { // left turn
                    if (multiplier + cnt >= 1023) { cnt = 1; }
                    else { cnt += multiplier; }
                }
                else if ( (ec1_prev == 0b11) && (ec1_curr == 0b10) )
                { // right turn
                    if (multiplier > cnt) { cnt = 1023; }
                    else { cnt -= multiplier; }
                }
            }
            if (ec2_prev != ec2_curr)
            {
                if ( (ec2_prev == 0b11) && (ec2_curr == 0b01) )
                { // left turn
                    if (multiplier + cnt >= 1023) { cnt = 1; }
                    else { cnt += multiplier; }
                }
                else if ( (ec2_prev == 0b11) && (ec2_curr == 0b10) )
                { // right turn
                    if (multiplier > cnt) { cnt = 1023; }
                    else { cnt -= multiplier; }
                }
            }
        flag = 0;
        }
    }
}
```