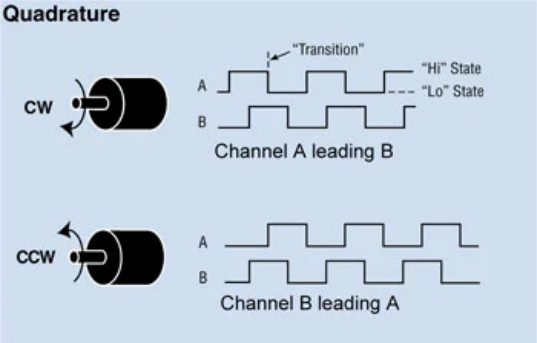
The rotary encoder has two output pins, they both produce a digital output when the encoder is turned. Depending on the direction that the encoder is turned then one channel will lead the other in outputting a response. From this the controller can determine which direction.



Piezo Speaker:

$$V_P = iZ$$
$$i = \frac{V_p}{Z}$$
$$i = \frac{3.3}{100}$$
$$i = 33mA$$

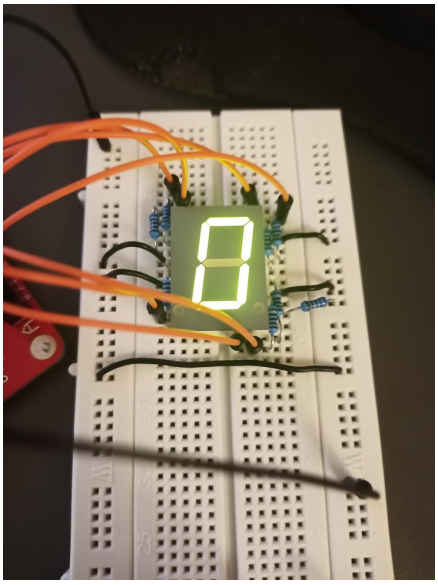
This is below the maximum pindrive of the Il Matto at 40mA. This means the piezo speaker can adequately drive the speaker.

11/11/20

https://en.wikipedia.org/wiki/Piezoelectric_speaker

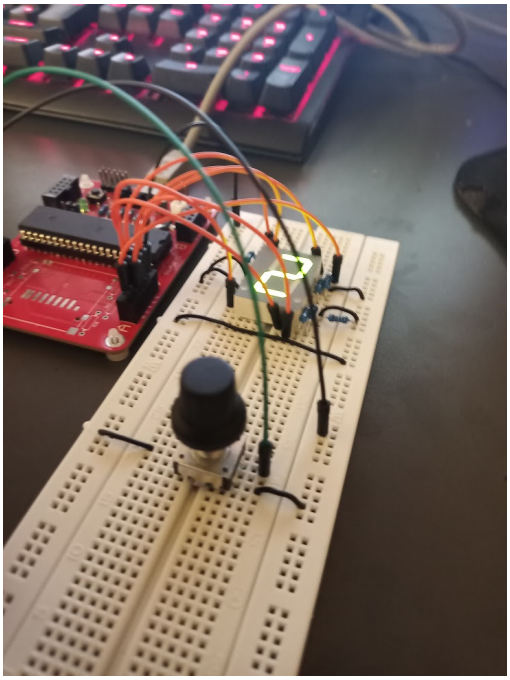
To display values on my seven segment display I wired up current limiting resistors and then connected them to the Il Matto using jumper leads. This allowed the Il Matto to drive the seven segment display.

```
#include <avr/io.h>
#include <util/delay.h>

/*
Pin     7 6 5 4 3 2 1 0
Segment a b c d e f g dp

0       1 1 1 1 1 1 0 0     0xFC
1       0 1 1 0 0 0 0 0     0x60
2       1 1 0 1 1 0 1 0     0xDA
3       1 1 1 1 0 0 1 0     0xF2
4       0 1 1 0 0 1 1 0     0x66
5       1 0 1 1 0 1 1 0     0xB6
6       1 0 1 1 1 1 1 0     0xBE
7       1 1 1 0 0 0 0 0     0xE0
8       1 1 1 1 1 1 1 0     0xFE
9       1 1 1 0 0 1 1 0     0xE6
*/

const uint8_t segments[10] = {0xFC, 0x60, 0xDA, 0xF2, 0x66, 0xB6, 0xBE, 0xE0, 0xFE, 0xE6};
uint8_t count = 0;

int main(void)
{
    DDRA = 0xFF; //Sets portA as outputs

    for (;;)
    {
        PORTA = segments[count % 10];
        count++;
        _delay_ms(1000); //Delay by 1 sec
    }
}
```



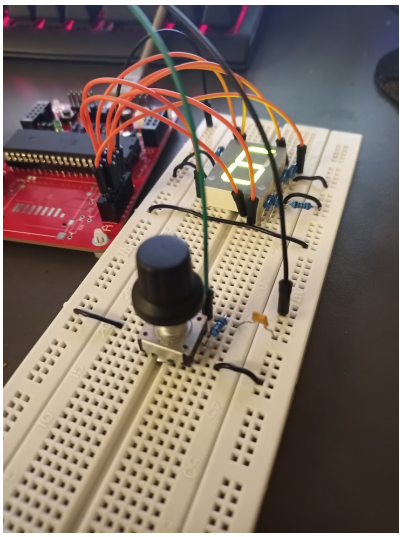
I then modified the code so that it would increment the number every time the rotary encoder was pressed. This worked for the most part but sometimes it would skip multiple places. This is likely due to switch bounce.

```c
#include <avr/io.h>
#include <util/delay.h>

/*
Pin     7 6 5 4 3 2 1 0
Segment a b c d e f g dp

0       1 1 1 1 1 1 0 0     0xFC
1       0 1 1 0 0 0 0 0     0x60
2       1 1 0 1 1 0 1 0     0xDA
3       1 1 1 1 0 0 1 0     0xF2
4       0 1 1 0 0 1 1 0     0x66
5       1 0 1 1 0 1 1 0     0xB6
6       1 0 1 1 1 1 1 0     0xBE
7       1 1 1 0 0 0 0 0     0xE0
8       1 1 1 1 1 1 1 0     0xFE
9       1 1 1 0 0 1 1 0     0xE6
*/

const uint8_t segments[10] = {0xFC, 0x60, 0xDA, 0xF2, 0x66, 0xB6, 0xBE, 0xE0, 0xFE, 0xE6};
uint8_t count = 0;

int main(void)
{
    DDRA = 0xFF;  //Sets portA as outputs
    DDRC = 0x00;  //Sets portC as inputs
    PORTC = 0xFF; //Enables inbuilt pull-up resistors

    PORTA = segments[0];
    for (;;)
    {
        while ((PINC & _BV(PC7)) != 0)
        {
            count++;
        }

        while ((PINC & _BV(PC7)) == 0)
        {
            PORTA = segments[count % 10];
        }
    }
}
```

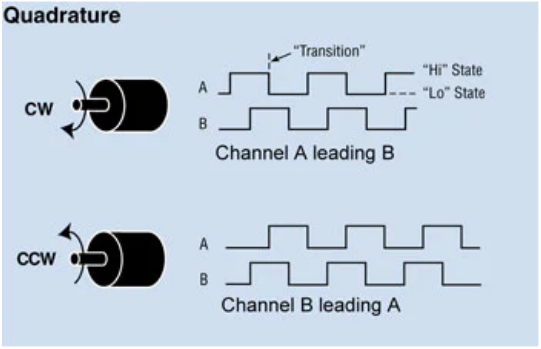


Due to the switch bounce I implemented one hardware solution, a low pass filter, and one software solution, disabling the pin. I found that this successfully debounced the pin and it would increment consistently.

```c
const uint8_t segments[10] = {0xFC, 0x60, 0xDA, 0xF2, 0x66, 0xB6, 0xBE, 0xE0, 0xFE, 0xE6};
uint8_t count = 0;

int main(void)
{
    DDRA = 0xFF;  //Sets portA as outputs
    DDRC = 0x00;  //Sets portC as inputs
    PORTC = 0xFF; //Enables inbuilt pull-up resistors

    PORTA = segments[0];
    for (;;)
    {
        while ((PINC & _BV(PC7)) != 0)
        {
            count++;
            _delay_ms(300);   //stop program for 300 ms to debounce switch
        }

        while ((PINC & _BV(PC7)) == 0)
        {
            PORTA = segments[count % 10];
        }

    }
}
```



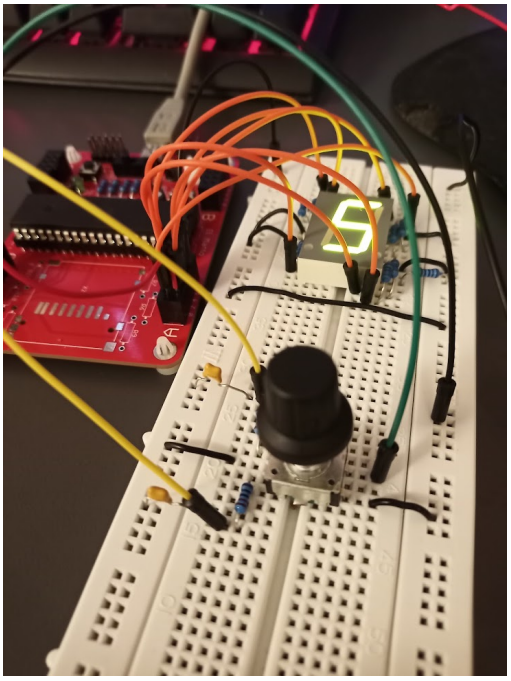
To use the rotary encoder to its full potential I used the I used the quadrature signal to increment and decrement the count when it detected a rotation.To



implement this I detected a rising edge trigger while another signal is true for each of the directions.

Reading of the rotary encoder was temperamental even after implementing debouncing techniques. My implementation works but not very successfully.

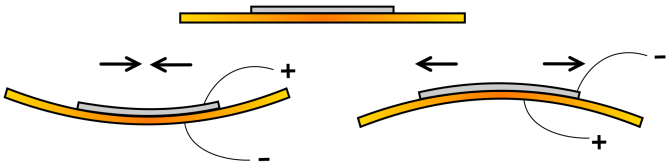
```c
int main(void)
{
    DDRA = 0xFF;  //Sets portA as outputs
    DDRC = 0x00;  //Sets portC as inputs
    PORTC = 0xFF; //Enables inbuilt pull-up resistors

    PORTA = segments[0];

    for (;;)
    {
        while ((PINC & _BV(PC0)) == 0)
        {
            nowA = (PINC & _BV(PC1)); // rising edge detection
            if (nowA != lastA)        //by ka7ehk on avrfreaks.net
            {
                if (nowA > 0)
                {
                    count++;
                    _delay_ms(300); //stop program for 10 ms to debounce switch
                }
            }
        }
        while ((PINC & _BV(PC1)) == 0)
        {
            nowB = (PINC & _BV(PC0)); // rising edge detction
            if (nowB != lastB)        //by ka7ehk on avrfreaks.net
            {
                if (nowB > 0)
                {
                    count--;
                    _delay_ms(300); //stop program for 10 ms to debounce switch
                }
            }
        }

        PORTA = segments[count % 10];
    }
}
```



After speaking to my instructor I was informed of a better method that would look at a change in both the A signal and the B signal and so would operate more reliably. I rewrote my code to implement this changed technique and found that it would work reliably.

```c
int main(void)
{
    uint8_t count = 0;
    uint8_t lastAB = 0x0, AB = 0x0;

    DDRA = 0xFF;  //Sets portA as outputs
    DDRC = 0x00;  //Sets portC as inputs
    PORTC = 0xFF; //Enables inbuilt pull-up resistors

    PORTA = segments[0];

    for (;;)
    {
        AB = rot_AB(PINC); // sample encoder

        if ((AB == 0x0 && lastAB == 0x2) || (AB == 0x3 && lastAB == 0x1)) // if CW
        {
            count--;
        }

        if ((AB == 0x3 && lastAB == 0x2) || (AB == 0x0 && lastAB == 0x1)) // if CCW
        {
            count++;
        }

        PORTA = segments[count % 10]; //display the  value
        _delay_ms(10); // debounce

        lastAB = AB; //update encoder position
    }
}
```

The piezo electric buzzer works by oscillating a crystal lattice. When a positive voltage it flexes in one direction and when a negative voltage is applied it flexes in the opposite direction. This means that to create a sound I have to alternate positive and negative voltages.



To test this I first created a simple program that creates a tone, and then implemented this into the button pressing program.

```c
int main(void)
{
    DDRA = 0xFF;  //Sets portA as outputs
    DDRC = 0x00;  //Sets portC as inputs
    PORTC = 0xFF; //Enables inbuilt pull-up resistors

    PORTA = segments[0];
    for (;;)
    {
     PORTA &= ~_BV(PA0); //Set bit n of port x low
     _delay_ms(1);
     PORTA |= _BV(PA0); //Set bit n of port x high
     _delay_ms(1);

    }
}
```

```c
int main(void)
{
    DDRA = 0xFF;  //Sets portA as outputs
    DDRC = 0x00;  //Sets portC as inputs
    PORTC = 0xFF; //Enables inbuilt pull-up resistors

    PORTA = segments[0];
    for (;;)
    {
        while ((PINC & _BV(PC7)) == 0)
        {
            count++;
            uint8_t i;
            for (i = 0; i < 150; i++)
            {
                PORTA &= ~_BV(PA0); //Set bit n of port x low
                _delay_ms(1);
                PORTA |= _BV(PA0); //Set bit n of port x high
                _delay_ms(1);
            }
        }

        while ((PINC & _BV(PC7)) != 0)
        {

            PORTA = segments[count % 10];
        }
    }
}
```

I then implemented it on my rotary encoder, to create a brief click when a value is updated.

```c
uint8_t buzz(uint8_t length)
{
    uint8_t i;
       for (i = 0; i < length; i++)
       {
           PORTA &= ~_BV(PA0); //Set bit n of port x low
           _delay_ms(0.5);
           PORTA |= _BV(PA0); //Set bit n of port x high
           _delay_ms(0.5);
       }
    return 0;
}

int main(void)
{
    uint8_t count = 0;
    uint8_t lastAB = 0x0, AB = 0x0;

    DDRA = 0xFF;  //Sets portA as outputs
    DDRC = 0x00;  //Sets portC as inputs
    PORTC = 0xFF; //Enables inbuilt pull-up resistors

    PORTA = segments[0];

    for (;;)
    {
        AB = rot_AB(PINC); // sample encoder

        if ((AB == 0x0 && lastAB == 0x2) || (AB == 0x3 && lastAB == 0x1)) // if CW
        {
            count--;
            buzz(30);
        }

        if ((AB == 0x3 && lastAB == 0x2) || (AB == 0x0 && lastAB == 0x1)) // if CCW
        {
            count++;
            buzz(30);
        }
```