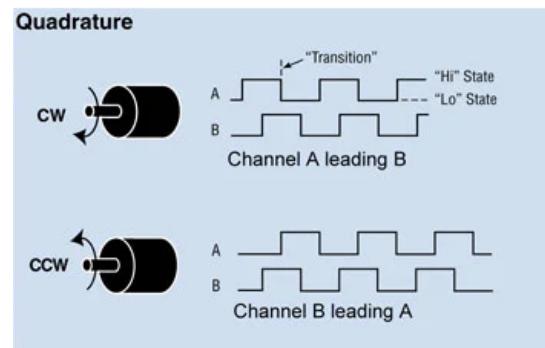


The rotary encoder has two output pins, they both produce a digital output when the encoder is turned. Depending on the direction that the encoder is turned then one channel will lead the other in outputting a response. From this the controller can determine which direction.



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 II Matto at 40mA. This means the piezo speaker can adequately drive the speaker.

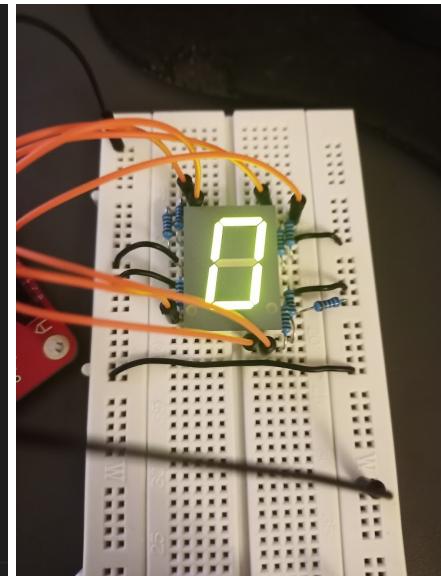
11/11/20

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 II Matto using jumper leads. This allowed the II Matto to drive the seven segment display.

```

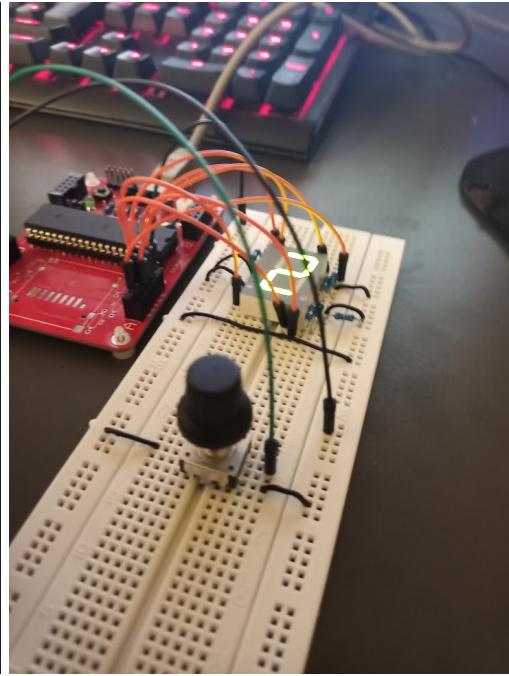
1 #include <avr/io.h>
2 #include <util/delay.h>
3
4 /*
5 Pin      7 6 5 4 3 2 1 0
6 Segment a b c d e f g dp
7
8 0      1 1 1 1 1 1 0 0    0xFC
9 1      0 1 1 0 0 0 0 0    0x00
10 2     1 1 0 1 1 0 1 0   0xDA
11 3     1 1 1 0 0 0 1 0   0xF2
12 4     0 1 1 0 0 1 1 0   0x66
13 5     1 0 1 1 0 1 1 0   0xB6
14 6     1 0 1 1 1 1 1 0   0xBE
15 7     1 1 1 0 0 0 0 0   0xE0
16 8     1 1 1 1 1 1 1 0   0xFE
17 9     1 1 1 0 0 1 1 0   0xE6
18 */
19
20 const uint8_t segments[10] = {0xFC, 0x00, 0xDA, 0xF2, 0x66, 0xB6, 0xBE, 0xE0, 0xFE, 0xE6};
21 uint8_t count = 0;
22
23 int main(void)
24 {
25     DDRD = 0xFF; //Sets portA as outputs
26
27     for (;;)
28     {
29         PORTA = segments[count % 10];
30         count++;
31         _delay_ms(1000); //Delay by 1 sec
32     }
33 }
```



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.

```

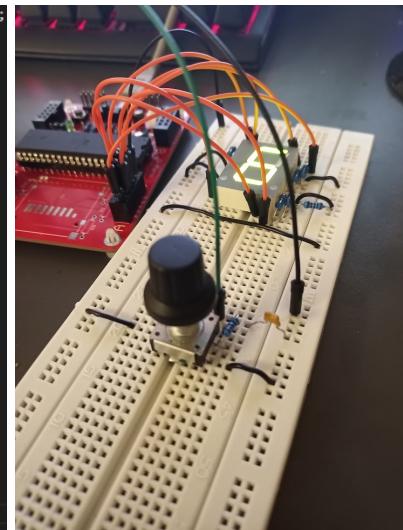
1  #include <avr/io.h>
2  #include <util/delay.h>
3
4  /*
5   Pin    7 6 5 4 3 2 1 0
6   Segment a b c d e f g dp
7
8   0     1 1 1 1 1 0 0 0xFC
9   1     0 1 1 0 0 0 0 0x60
10  2    1 1 0 1 1 0 1 0 0xDA
11  3    1 1 1 1 0 0 1 0 0xF2
12  4    0 1 1 0 0 1 1 0 0x66
13  5    1 0 1 1 0 1 1 0 0xB6
14  6    1 0 1 1 1 1 1 0 0xBE
15  7    1 1 1 0 0 0 0 0xE0
16  8    1 1 1 1 1 1 1 0 0xFE
17  9    1 1 1 0 0 1 1 0 0xE6
18 */
19
20 const uint8_t segments[10] = {0xFC, 0x60, 0xDA, 0xF2, 0x66, 0xB6, 0xBE, 0xE0, 0xFE, 0xE6};
21 uint8_t count = 0;
22
23 int main(void)
24 {
25     DDRA = 0xFF; //Sets portA as outputs
26     DDRC = 0x00; //Sets portC as inputs
27     PORTC = 0xFF; //Enables inbuilt pull-up resistors
28
29     PORTA = segments[0];
30     for (;;)
31     {
32         while ((PINC & _BV(PC7)) != 0)
33         {
34             count++;
35         }
36
37         while ((PINC & _BV(PC7)) == 0)
38         {
39             PORTA = segments[count % 10];
40         }
41     }
42 }
```



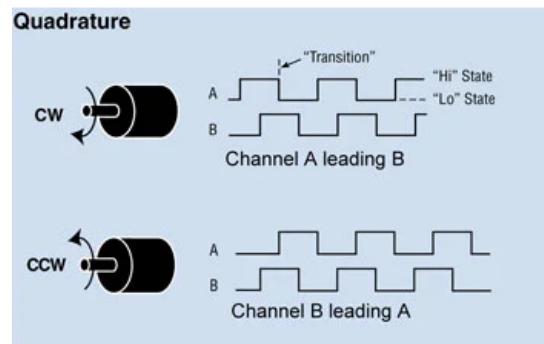
Due to the switch bounce I implemented one hardware solution, a low pass filter, and one software solution, disabling the pin. I found that this successfully debounced the pin and it would increment consistently.

```

20 const uint8_t segments[10] = {0xFC, 0x60, 0xDA, 0xF2, 0x66, 0xB6, 0xBE, 0xE0, 0xFE, 0xE6};
21 uint8_t count = 0;
22
23 int main(void)
24 {
25     DDRA = 0xFF; //Sets portA as outputs
26     DDRC = 0x00; //Sets portC as inputs
27     PORTC = 0xFF; //Enables inbuilt pull-up resistors
28
29     PORTA = segments[0];
30     for (;;)
31     {
32         while ((PINC & _BV(PC7)) != 0)
33         {
34             count++;
35             _delay_ms(300); //stop program for 300 ms to debounce switch
36         }
37
38         while ((PINC & _BV(PC7)) == 0)
39         {
40             PORTA = segments[count % 10];
41         }
42     }
43 }
```



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



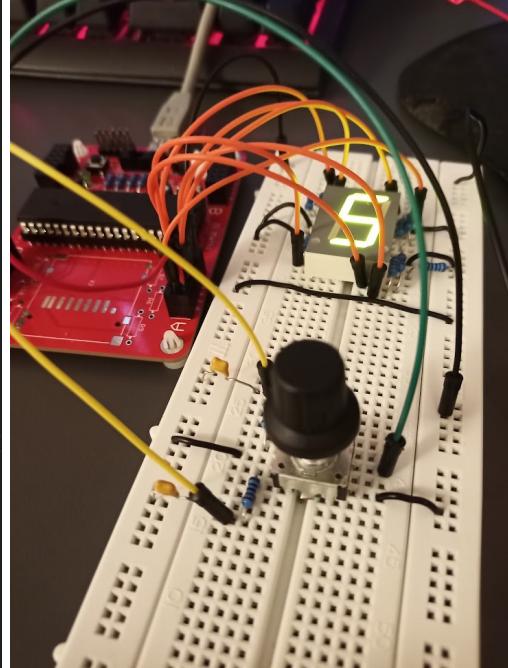
implement this I detected a rising edge trigger while another signal is true for each of the directions.

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

```

27 int main(void)
28 {
29     DDRA = 0xFF; //Sets portA as outputs
30     DDRC = 0x00; //Sets portC as inputs
31     PORTC = 0xFF; //Enables inbuilt pull-up resistors
32
33     PORTA = segments[0];
34
35     for (;;)
36     {
37         while ((PINC & _BV(PC0)) == 0)
38         {
39             nowA = (PINC & _BV(PC1)); // rising edge detection
40             if (nowA != lastA) //by ka7ehk on avrfreaks.net
41             {
42                 if (nowA > 0)
43                 {
44                     count++;
45                     _delay_ms(300); //stop program for 10 ms to debounce switch
46                 }
47             }
48         }
49         while ((PINC & _BV(PC1)) == 0)
50         {
51             nowB = (PINC & _BV(PC0)); // rising edge detection
52             if (nowB != lastB) //by ka7ehk on avrfreaks.net
53             {
54                 if (nowB > 0)
55                 {
56                     count--;
57                     _delay_ms(300); //stop program for 10 ms to debounce switch
58                 }
59             }
60         }
61         PORTA = segments[count % 10];
62     }
63 }

```



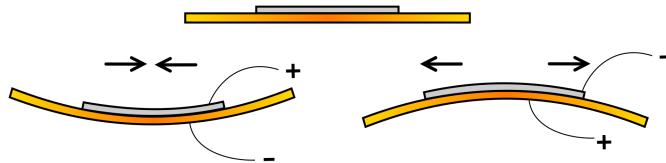
After speaking to my instructor I was informed of a better method that would look at a change in both the A signal and the B signal and so would operate more reliably. I rewrote my code to implement this changed technique and found that it would work reliably.

```

27 int main(void)
28 {
29     uint8_t count = 0;
30     uint8_t lastAB = 0x0, AB = 0x0;
31
32     DDRA = 0xFF; //Sets portA as outputs
33     DDRD = 0x00; //Sets portD as inputs
34     PORTD = 0xFF; //Enables inbuilt pull-up resistors
35
36     PORTA = segments[0];
37
38     for (;;)
39     {
40         AB = rot_AB(PINC); // sample encoder
41
42         if ((AB == 0x0 && lastAB == 0x2) || (AB == 0x3 && lastAB == 0x1)) // if CW
43         {
44             count--;
45         }
46
47         if ((AB == 0x3 && lastAB == 0x2) || (AB == 0x0 && lastAB == 0x1)) // if CCW
48         {
49             count++;
50         }
51
52         PORTA = segments[count % 10]; //display the value
53         _delay_ms(10); // debounce
54
55         lastAB = AB; //update encoder position
56     }
57 }

```

The piezo electric buzzer works by oscillating a crystal lattice. When a positive voltage is applied 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.

```

25 int main(void)
26 {
27     DDRA = 0xFF; //Sets portA as outputs
28     DDRC = 0x00; //Sets portC as inputs
29     PORTC = 0xFF; //Enables inbuilt pull-up resistors
30
31     PORTA = segments[0];
32     for (;;)
33     {
34         PORTA &= ~_BV(PA0); //Set bit n of port x low
35         _delay_ms(1);
36         PORTA |= _BV(PA0); //Set bit n of port x high
37         _delay_ms(1);
38     }
39 }
40 }
```

```

25 int main(void)
26 {
27     DDRA = 0xFF; //Sets portA as outputs
28     DDRC = 0x00; //Sets portC as inputs
29     PORTC = 0xFF; //Enables inbuilt pull-up resistors
30
31     PORTA = segments[0];
32     for (;;)
33     {
34         while ((PINC & _BV(PC7)) == 0)
35         {
36             count++;
37             uint8_t i;
38             for (i = 0; i < 150; i++)
39             {
40                 PORTA &= ~_BV(PA0); //Set bit n of port x low
41                 _delay_ms(1);
42                 PORTA |= _BV(PA0); //Set bit n of port x high
43                 _delay_ms(1);
44             }
45         }
46         while ((PINC & _BV(PC7)) != 0)
47         {
48             count++;
49         }
50     }
51     PORTA = segments[count % 10];
52 }
53 }
```

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

```

27     uint8_t buzz(uint8_t length)
28     {
29         uint8_t i;
30         for (i = 0; i < length; i++)
31         {
32             PORTA &= ~_BV(PA0); //Set bit n of port x low
33             _delay_ms(0.5);
34             PORTA |= _BV(PA0); //Set bit n of port x high
35             _delay_ms(0.5);
36         }
37     }
38 }
39
40 int main(void)
41 {
42     uint8_t count = 0;
43     uint8_t lastAB = 0x0, AB = 0x0;
44
45     DDRA = 0xFF; //Sets portA as outputs
46     DDRC = 0x00; //Sets portC as inputs
47     PORTC = 0xFF; //Enables inbuilt pull-up resistors
48
49     PORTA = segments[0];
50
51     for (;;)
52     {
53         AB = rot_AB(PINC); // sample encoder
54
55         if ((AB == 0x0 && lastAB == 0x2) || (AB == 0x3 && lastAB == 0x1)) // if CW
56         {
57             count--;
58             buzz(30);
59         }
60
61         if ((AB == 0x3 && lastAB == 0x2) || (AB == 0x0 && lastAB == 0x1)) // if CCW
62         {
63             count++;
64             buzz(30);
65         }
66     }
67 }
```