ECE167L

Xingyu Zhou

Max Dunne

2020/02/08

Lab 2 Report

Introduction:

In this lab we are introduced with 3 new components: the quadrature encoder, an ultrasonic sensor and a capacitive sensor. There are 3 parts in total and in each part, we are asked to do different things with the components that are provided. For part 1, we need to write codes such that the LED on the encoder will shift color as we turn it. For part 2, we need to detect the distance of the object using the ultrasonic sensor and use the distance to make sounds on the speaker. For the last part, we will be writing another piece of code to determine whether the capacitive touch sensor is being touched or not. There are a lot of diagrams involved in this lab and I'll provide all of them at the end.

Part 1: QEI

This part is fairly easy to implement. The first thing I did was to solder the encoder with the PCB board provided. Then I connected the test circuit provided in the datasheet (figure 1) to see how it works. Based on my observation on the oscilloscope, when I turned the knob clockwise, the waveform of channel A would rise before the waveform of channel B and vice versa if I turn

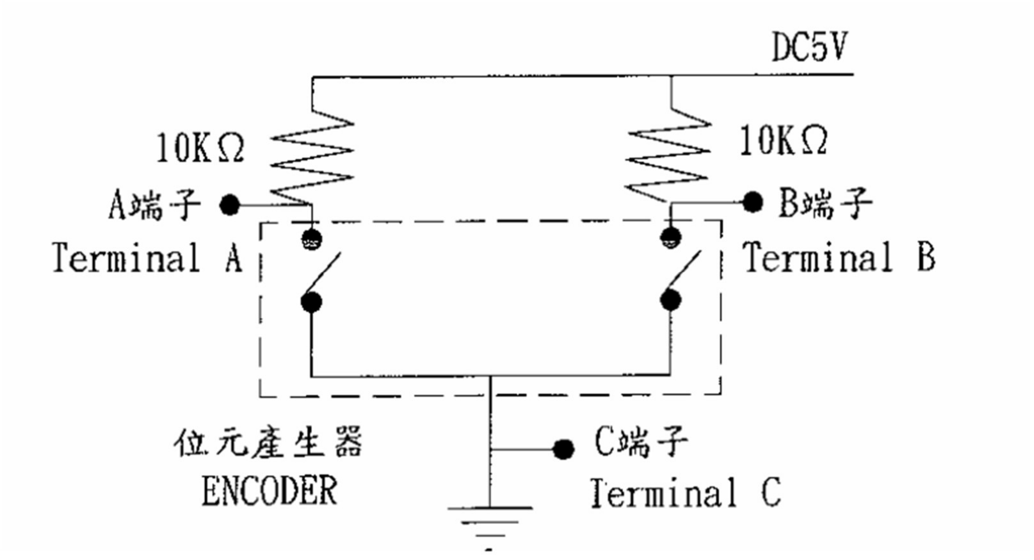


Figure 1: encoder test circuit

it counterclockwise. Since we were to implement a counter that could count or down based on the direction of the knob, this gave me some information on how to design the state machine. Figure 2 is the state machine that I draw for part 1. The way I implemented the state machine is as follows: first, I read the value of both the pins (A and B) in the initialization function, then I stored them as 4 states (10,01,00,11) based on the pin reading. Second, when the program enters the ISR, it will read the value of A and B again and store them as 4 new states. By now I will have 8 states in total—4 old states and 4 new states. I then compare them together following the logic provided in the state machine in figure 2. Once I have the encoder connected to uno32 (detailed diagram will be provided at the end of the report) and printed out the reading on the IO shield, I was able to get a stable reading from 0 to 96, which was what the lab manual specified.

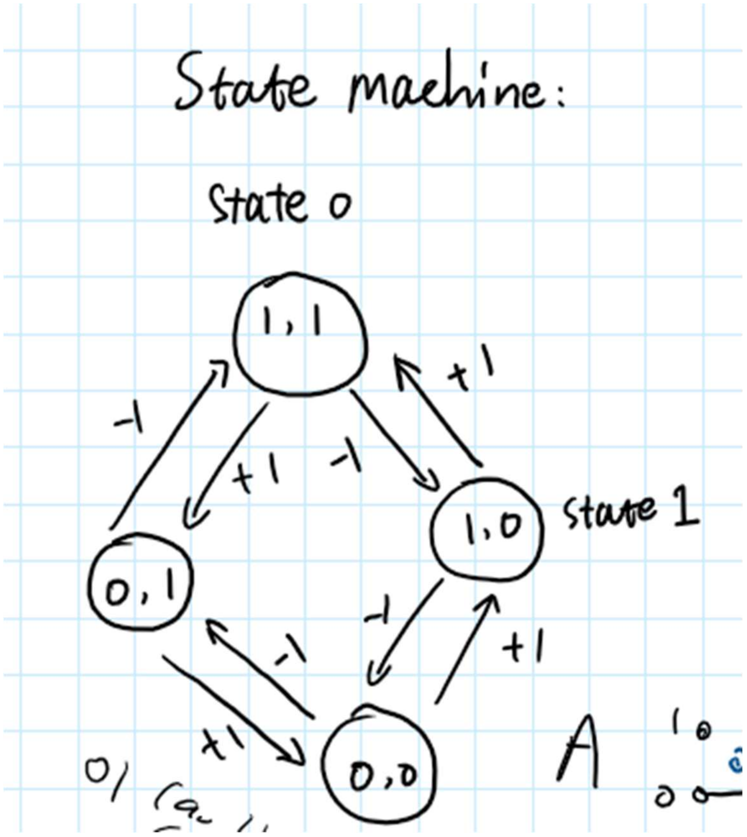


Figure 2: Encoder State Machine.

The last thing that we were asked to do in part 1 is the color wheel. Since the encoder is equipped with LED, we need to perform a color shift when the knob is being turned around. To do so, we need to map the color to the readings of the encoder. Luckily, thanks to our TAs' (or maybe some students') detailed color map on the white board, I was able to get this part done almost instantly. The color map is provided in figure 3 below. As you can see in figure 3, its divided into 6 sections, which for me is 16 ticks per section (96/6 = 16). For each section, what I need to do is to turn on/off the color based on the color map. We are using the PWM to control the brightness of each color so 1000(maximum frequency) means that color is completely turned off and 0 means it's completely turned on. One more thing to mention about the color map—the line going up means that color is being turned on and vice versa. Also, since my counter can only go up to 96, to make sure it can reach the maximum frequency, I scaled it up by multiplying the reading by 62.

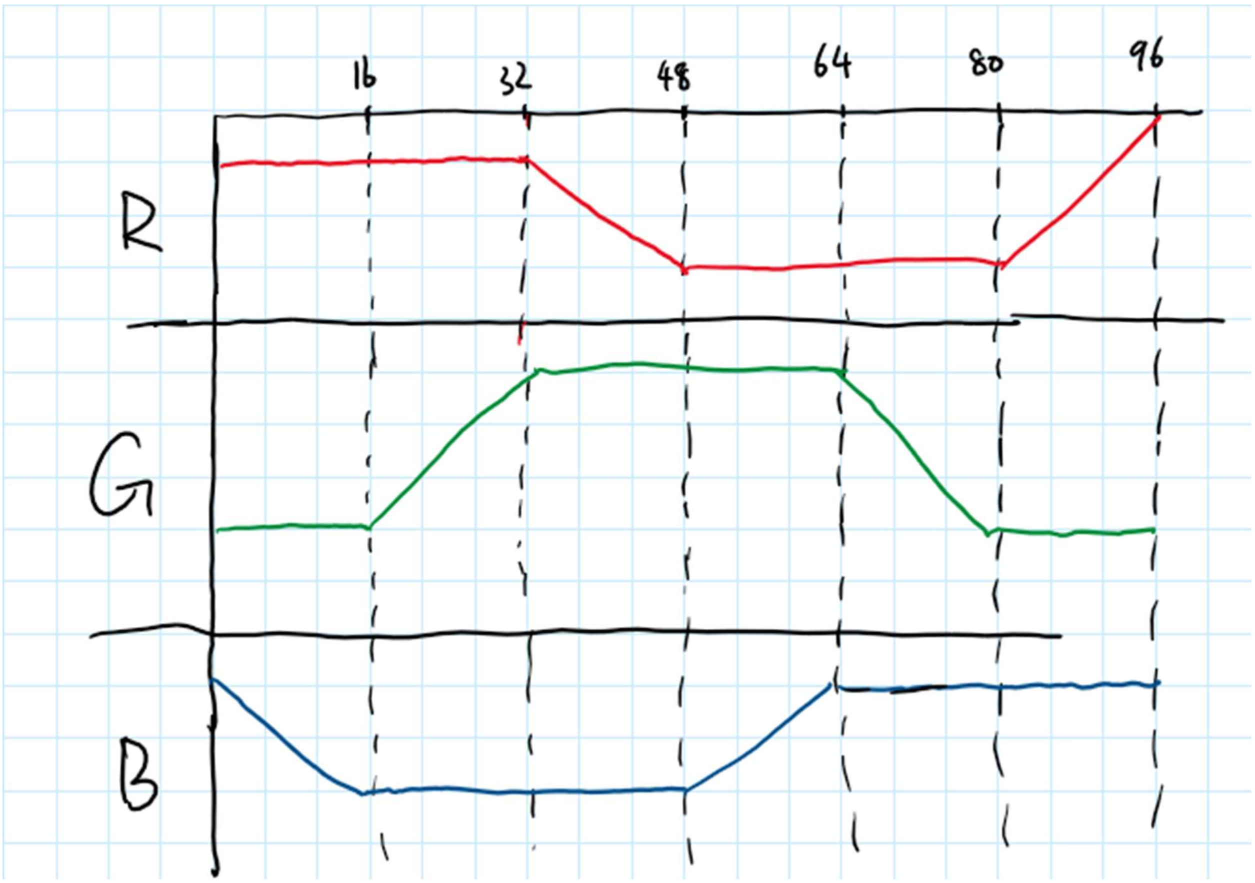


Figure 3: Color map used to do the color encoding

Result for part 1:

The encoder can count from 0 to exactly 96 when the knob is turned 360 degree. Also the color of the LED is changing based on the turning rate of the knob—a full color wheel is shown just like the one presented in the lab manual.

Part 2: Ping Sensor

The ping sensor is an ultrasonic sensor than can be used to detect objects and the distances between the sensor and the object. Again, detailed wiring diagram will be provided at the end of this report. The first thing that we need to do is to test if the ping sensor is working. According to the lab manual, we need to send a trigger signal for about 10 microseconds and then let it rest for 60 milliseconds and repeat the process. These times are set using a variable called PR4. To figure out what the value of PR4 should be, I connected the ultrasonic sensor to the oscilloscope and set the PR4 to be 10 in the timer ISR. By using the “width” function on the oscilloscope, I got a reading of 17microseconds of the pulse width, which meant that one tick in PR4 stands for 1.7 microseconds. Thus, I had my PR4 sent to 6 and 37500, which were 10 microseconds and 60 milliseconds respectively. Also, in the timer ISR, I had two if statements—their job were to detect the current reading of the PR4 and set the trigger to be high/low. The next step is to configure the Change Notice ISR. Each time a pulse sent, the Change Notice ISR will be triggered—this means that the ISR will be called twice—both on the rising and falling edge of the echo. What I did in this ISR is that I read the status of the pin#35 on the Uno32, this will give me either 0 (falling) or 1 (rising). Then I will compare it with the value previously read in the initialization function—if they are not the same, it means that an echo pulse has been sent out. I will then record the time in microseconds. If they are the same, it means that the pulse has been received, I will record another time in microseconds called finish time. Now I can get the time of travel, which is finish time – start time, and use it to calculate the distance. The distance is given by $\frac{microsecond}{58}$.

After making sure it works, we need to calibrate the ultrasonic sensor using a method called least squares. The instructions are pretty clear on the lab manual. I set up markings from 0cm to 100cm with 10 cm between each of them. Then I recorded the distance and time of flight and plotted the graph in figure 4.

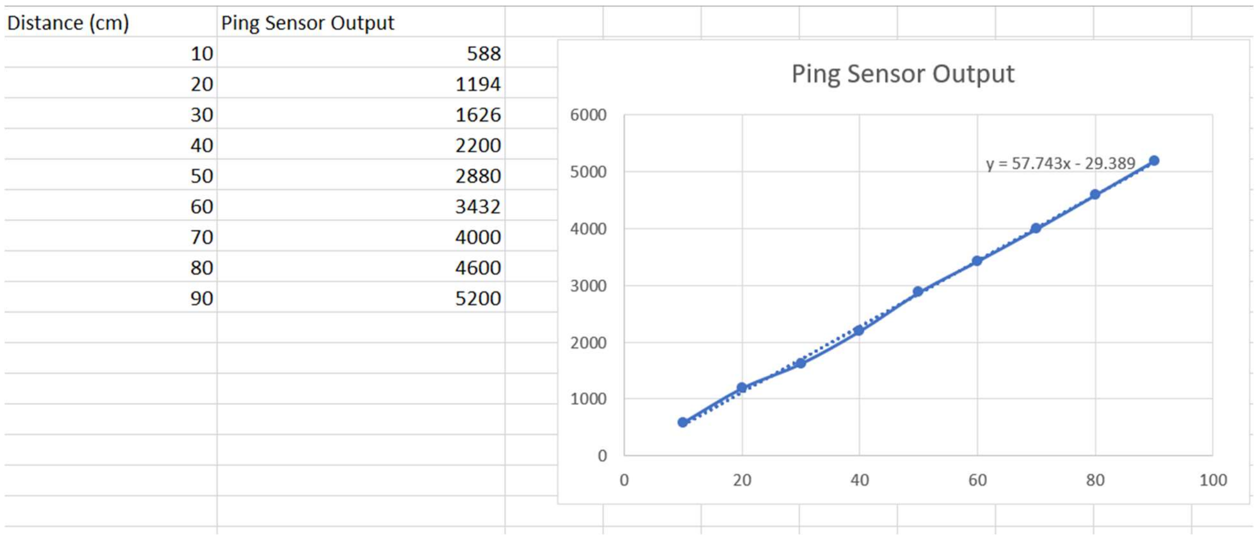

Distance (cm)	Ping Sensor Output
10	588
20	1194
30	1626
40	2200
50	2880
60	3432
70	4000
80	4600
90	5200

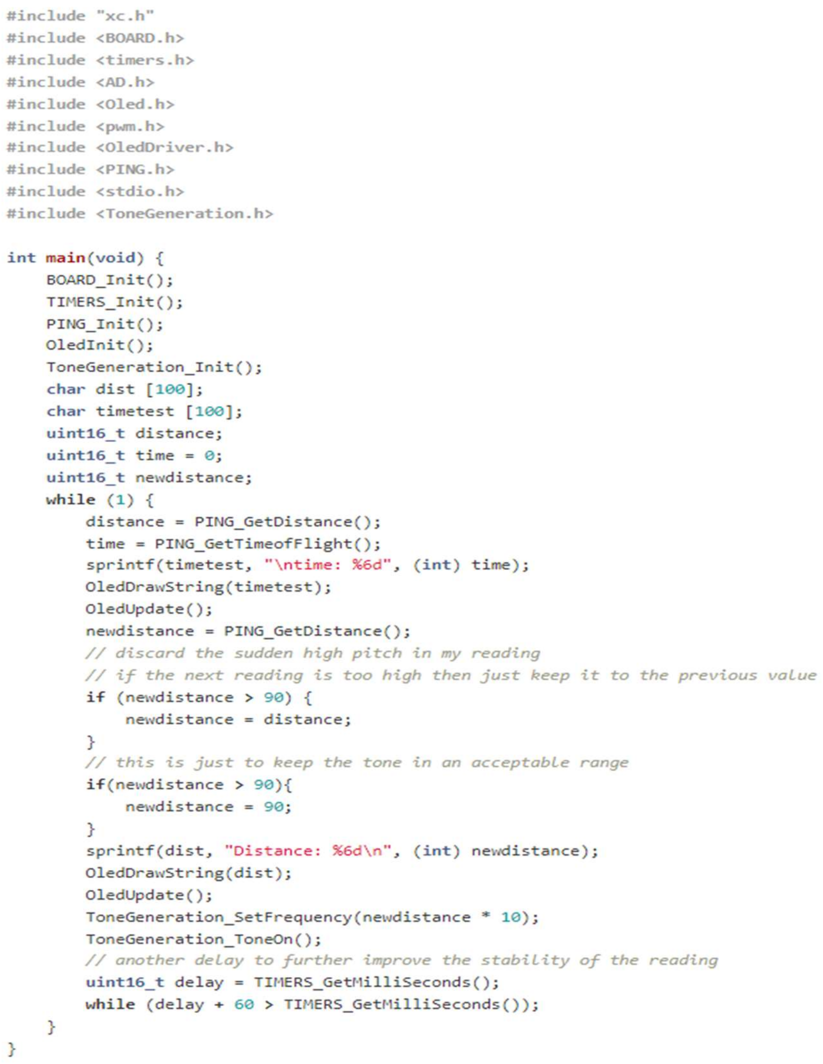
Figure 4: Sensor calibration data

By following the equation($\begin{matrix} m \\ b \end{matrix} = [X^T X]^{-1} X^T Y$) on the lab manual, I was able to get m = 57.74 and b = -29.38. This is close to what I have got using the linearization method in Excel. Lastly, instead of calculation the distance using the $\frac{microsecond}{58}$, I put the inverse function of what I got, which was $distance = \frac{time\ of\ flight + 29}{57}$. This equation will give me the measurement in centimeters.

Finally, we need to make sounds using the speaker using the distance measured. This is quite easy since instead of using a set frequency, I just put the measured distance in the ToneGeneration_SetFrequency(). I have my distance multiply by 10 because I want the frequency to go from 0 to 900, otherwise it would just go from 0 to 90, which is basically no change at all. Moreover, to eliminate the scratchiness of the sound, I put a small delay in my main code since I want the distance to be updated at a slower, consistent rate.

Result for part 2:

The measurements are somewhat accurate based on the size of the object being detected, and I was able to produce a smooth tone based on the distances of the detected object.



```c
#include "xc.h"
#include <BOARD.h>
#include <timers.h>
#include <AD.h>
#include <Oled.h>
#include <pwm.h>
#include <OledDriver.h>
#include <PING.h>
#include <stdio.h>
#include <ToneGeneration.h>

int main(void) {
    BOARD_Init();
    TIMERS_Init();
    PING_Init();
    OledInit();
    ToneGeneration_Init();
    char dist [100];
    char timetest [100];
    uint16_t distance;
    uint16_t time = 0;
    uint16_t newdistance;
    while (1) {
        distance = PING_GetDistance();
        time = PING_GetTimeofFlight();
        sprintf(timetest, "\ntime: %6d", (int) time);
        OledDrawString(timetest);
        OledUpdate();
        newdistance = PING_GetDistance();
        // discard the sudden high pitch in my reading
        // if the next reading is too high then just keep it to the previous value
        if (newdistance > 90) {
            newdistance = distance;
        }
        // this is just to keep the tone in an acceptable range
        if(newdistance > 90){
            newdistance = 90;
        }
        sprintf(dist, "Distance: %6d\n", (int) newdistance);
        OledDrawString(dist);
        OledUpdate();
        ToneGeneration_SetFrequency(newdistance * 10);
        ToneGeneration_ToneOn();
        // another delay to further improve the stability of the reading
        uint16_t delay = TIMERS_GetMilliSeconds();
        while (delay + 60 > TIMERS_GetMilliSeconds());
    }
}
```

Figure 5: Main code for ping sensor

Part 3: CAPTOUCH Sensor

This is the part which I spent most of my time on. In this part we are working with capacitive touch sensor. We will be using this sensor to build 3 circuits, and eventually write a piece of code that can detect if the sensor is being touched or not.

The first thing we need to do is to calculate the change in capacitance when the sensor is being touched. To do this, we will need to build a RC circuit with C being the touch sensor. Figure 6 is the circuit diagram. Here I chose my resistance to be 200kΩ and the power supply is running at 5V. By using the oscilloscope, I got that when the sensor is not touched, the rise time is 57 microseconds. When the sensor is touched, the rise time is 11.2 microseconds. This gives me a time difference of 45.8 microseconds. Then by using the formula that I found on Wikipedia $t_r = 2.2t$ with t_r being the time difference and t being the time constant, I was able to get the change in capacitance. Figure 7 shows the process of how I got the change in capacitance.

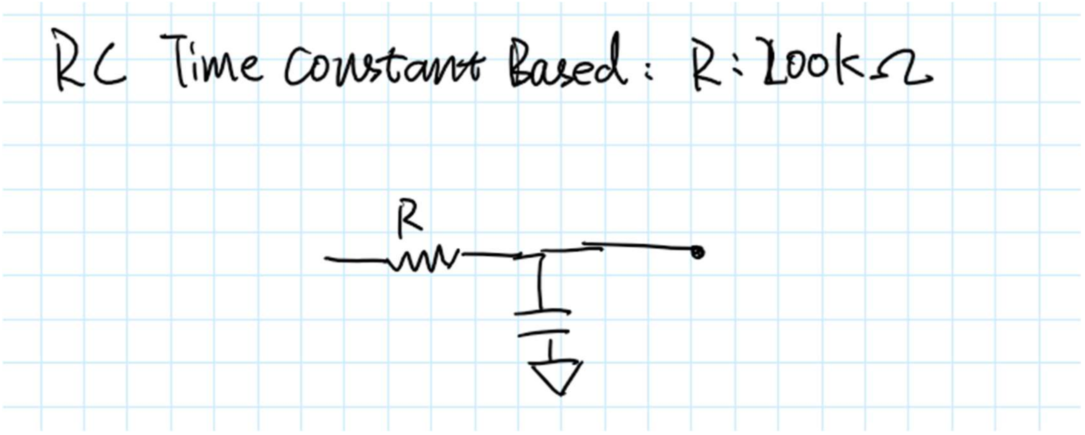


Figure 6: RC Time Constant based circuit

rise time (10% to 90%) $t_r \approx 2.2\tau \approx \frac{0.35}{f_c}$

Rise time $= 57 - 11.2 = 45.8\mu s = 0.0000458s.$

$2.2\tau = 0.0000458$

$\tau = 0.00002.$

$\tau = RC, \quad R = 200k\Omega.$

$\therefore C = \frac{\tau}{R} = \frac{0.00002}{2 \times 10^5} = \boxed{99.5pF}$ change in capacitance

Figure 7: Change in capacitance

The next thing we need to build is a capacitive bridge. To begin with, we will need to build a Wheatstone bridge but with the touch sensor being parallel with one of the capacitors. The circuit diagram is shown in figure 8. Noticed that I have only included the low-pass filter circuit because the signal of high-pass filter is completely unusable, and I'm not getting any response by touching the sensor. My cutoff frequency is 7989Hz according to the 200kΩ resistor and 100pF capacitor. After experimenting with it, I've notice that the low-pass filter setup, with the normal polarity gives the best signals. When I do touch the sensor, I can see that the signal labeled "yellow" will shrink compare to the original output signal "green". Figure 9 shows one of the waveforms that I got when the sensor is touched with the frequency set to 5000Hz.

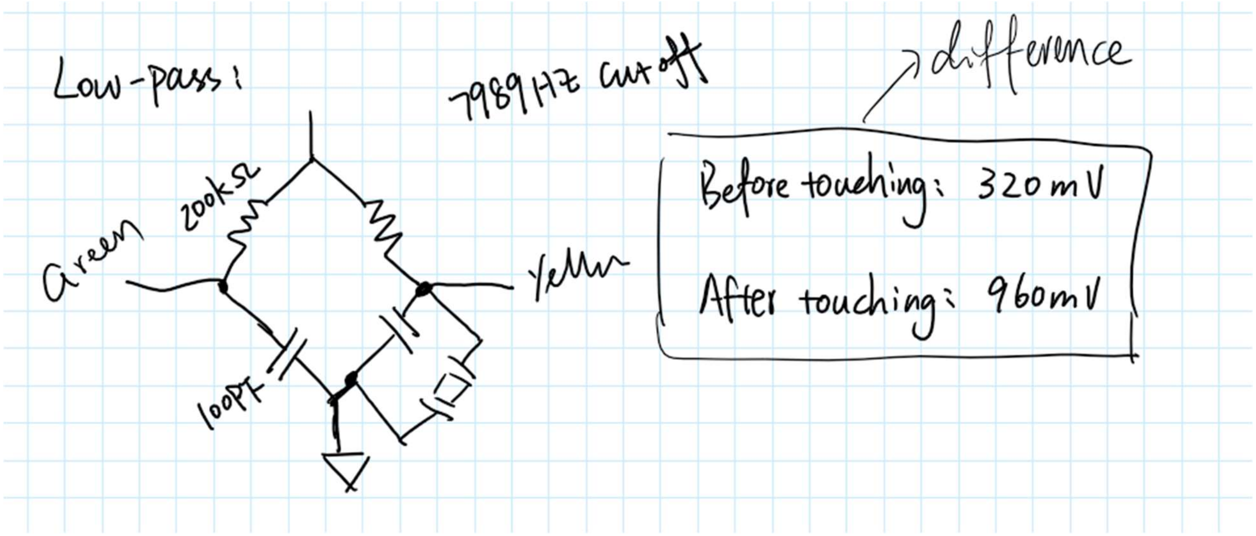
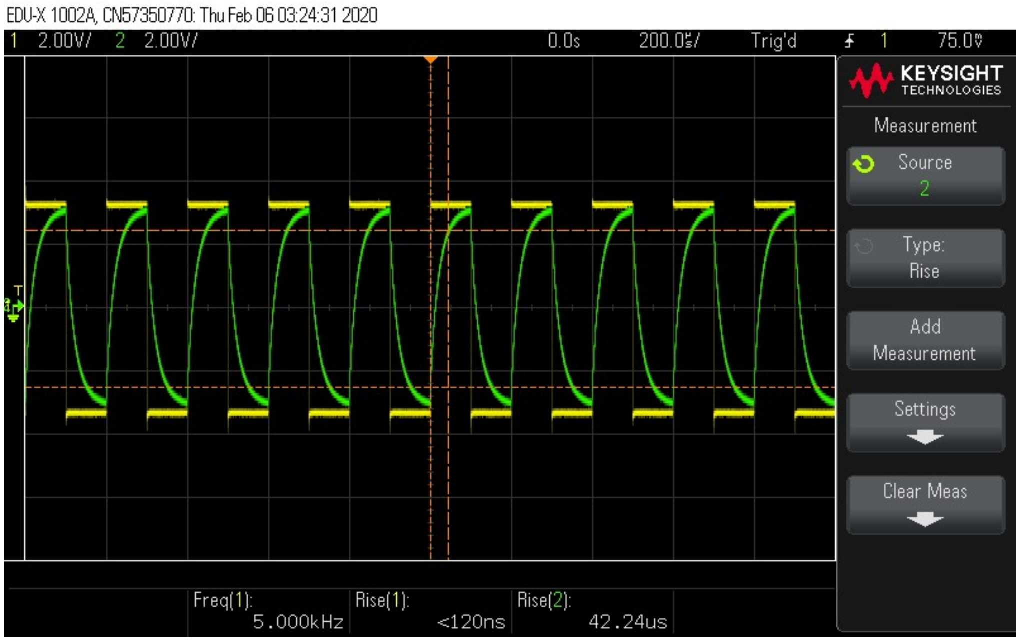


Figure 8: Low-pass filter with captouch sensor circuit



EDU-X 1002A, CN57350770: Thu Feb 06 03:24:31 2020

Freq(1): 5.000kHz
Rise(1): <120ns
Rise(2): 42.24us

Figure 9: RC-low pass filter with captouch sensor's waveform

Next, we need to implement a difference amp using the MCP6004. I tried to implement one, but the output didn't change too much compare to the output without the difference amp, so I decided not to use one. Figure 10 is the circuit diagram of how I connect the MCP6004 to my low-pass bridge.

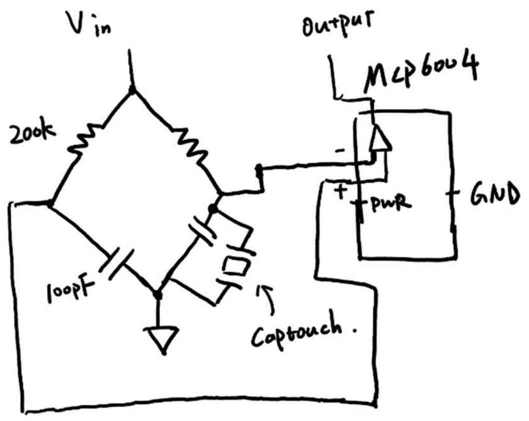
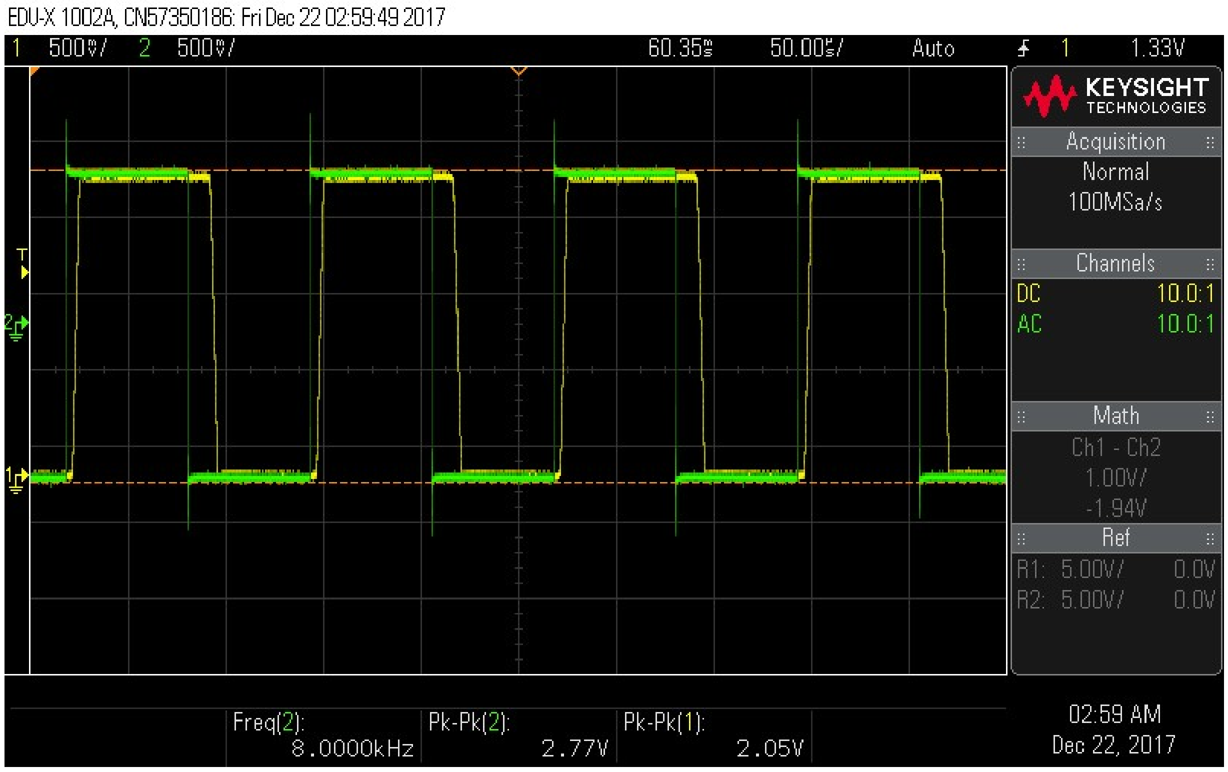


Figure 10: Captouch bridge with MCP6004 opamp

In figure 11&12 you can see that I'm getting a pretty stable signal using the oscilloscope with both touched and untouched condition.



```
EDU-X 1002A, CN57350186: Fri Dec 22 02:59:49 2017
1  500mV/  2  500mV/          60.35µs   50.00µs/   Auto   ƒ  1  1.33V

Acquisition
Normal
100MSa/s

Channels
DC    10.0:1
AC    10.0:1

Math
Ch1 - Ch2
1.00V/
-1.94V

Ref
R1:  5.00V/   0.0V
R2:  5.00V/   0.0V

Freq(2):          Pk-Pk(2):        Pk-Pk(1):           02:59 AM
     8.0000kHz          2.77V            2.05V       Dec 22, 2017
```

Figure 11: Captouch bridge with MCP6004 with captouch sensor touched

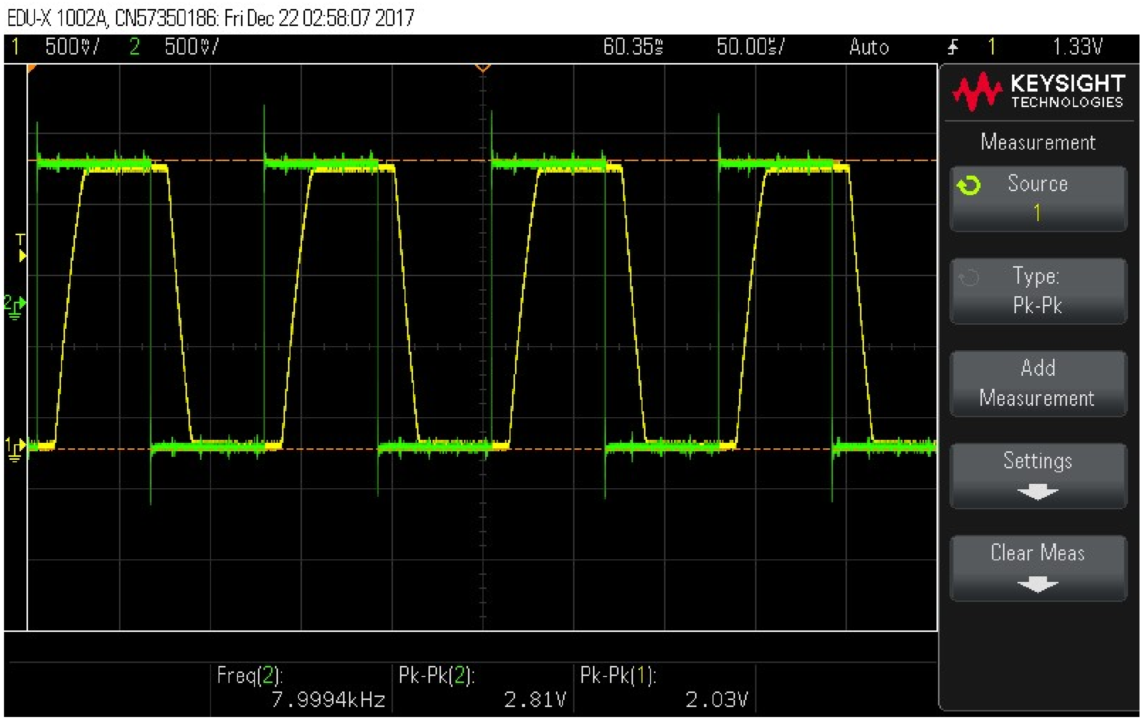
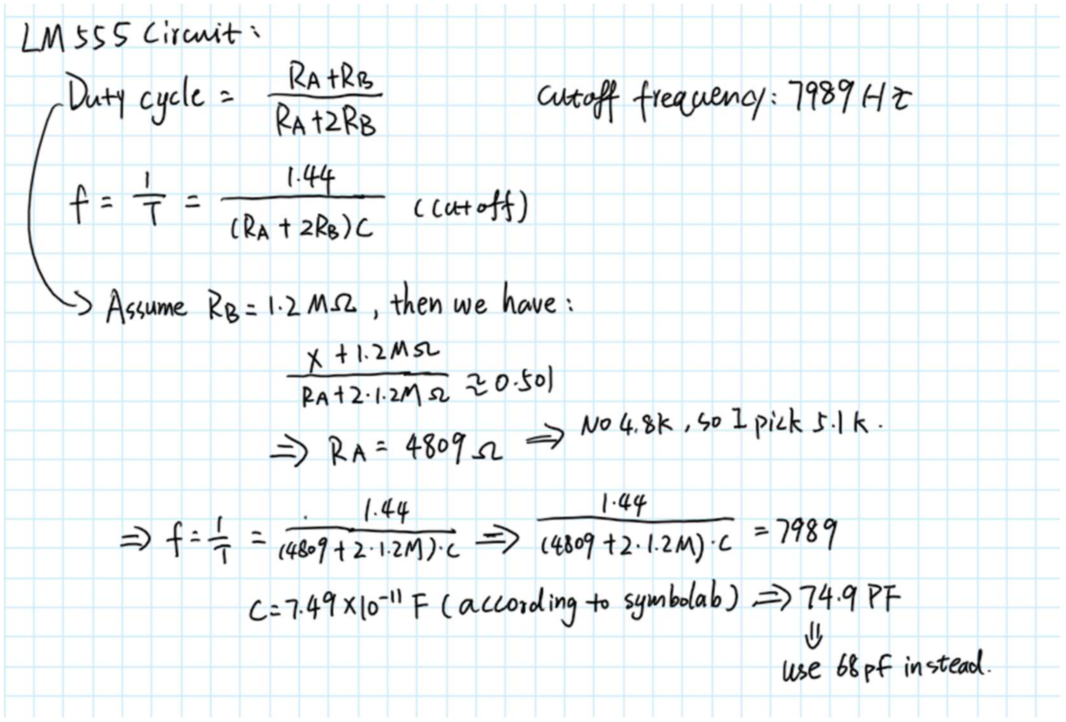


Figure 12: Captouch bridge with MCP6004 with captouch sensor untouched

Then we were introduced to LM555 timer, which based on my research, acts like a signal generator when powered up. Our job was to set up the LM555 in 50% duty cycle mode. Figure 13 shows the process of how I calculate the duty cycle and determine the resistor value.



LM555 Circuit:

Duty cycle = $\frac{R_A+R_B}{R_A+2R_B}$ cutoff frequency: 7989 Hz

$f = \frac{1}{T} = \frac{1.44}{(R_A+2R_B)C}$ (cutoff)

Assume $R_B = 1.2\,M\Omega$, then we have:

$\frac{x + 1.2M\Omega}{R_A + 2\cdot 1.2M\Omega} \approx 0.501$

$\Rightarrow R_A = 4809\,\Omega$ $\Rightarrow$ No 4.8k, so I pick 5.1k.

$\Rightarrow f = \frac{1}{T} = \frac{1.44}{(4809+2\cdot1.2M)\cdot C} \Rightarrow \frac{1.44}{(4809+2\cdot 1.2M)\cdot C} = 7989$

$C = 7.49\times10^{-11}$ F (according to symbolab) $\Rightarrow$ 74.9 PF

$\Downarrow$

use 68pF instead.

Figure 13: calculation process of determining the 50% duty cycle

The equation is found on the LM555 data sheet and I used a different circuit that I found online since the one provided in the datasheet doesn't work well.

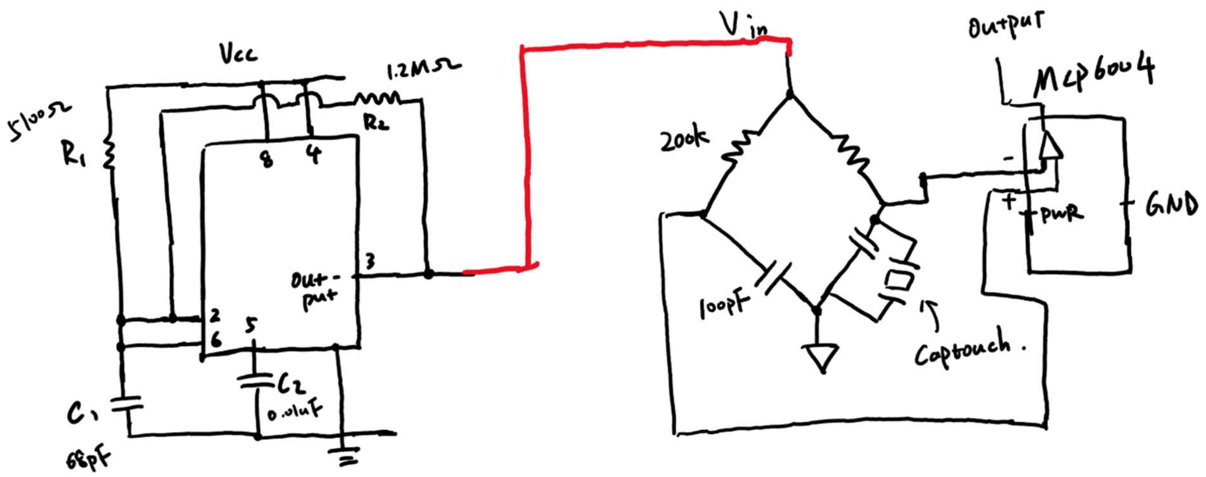
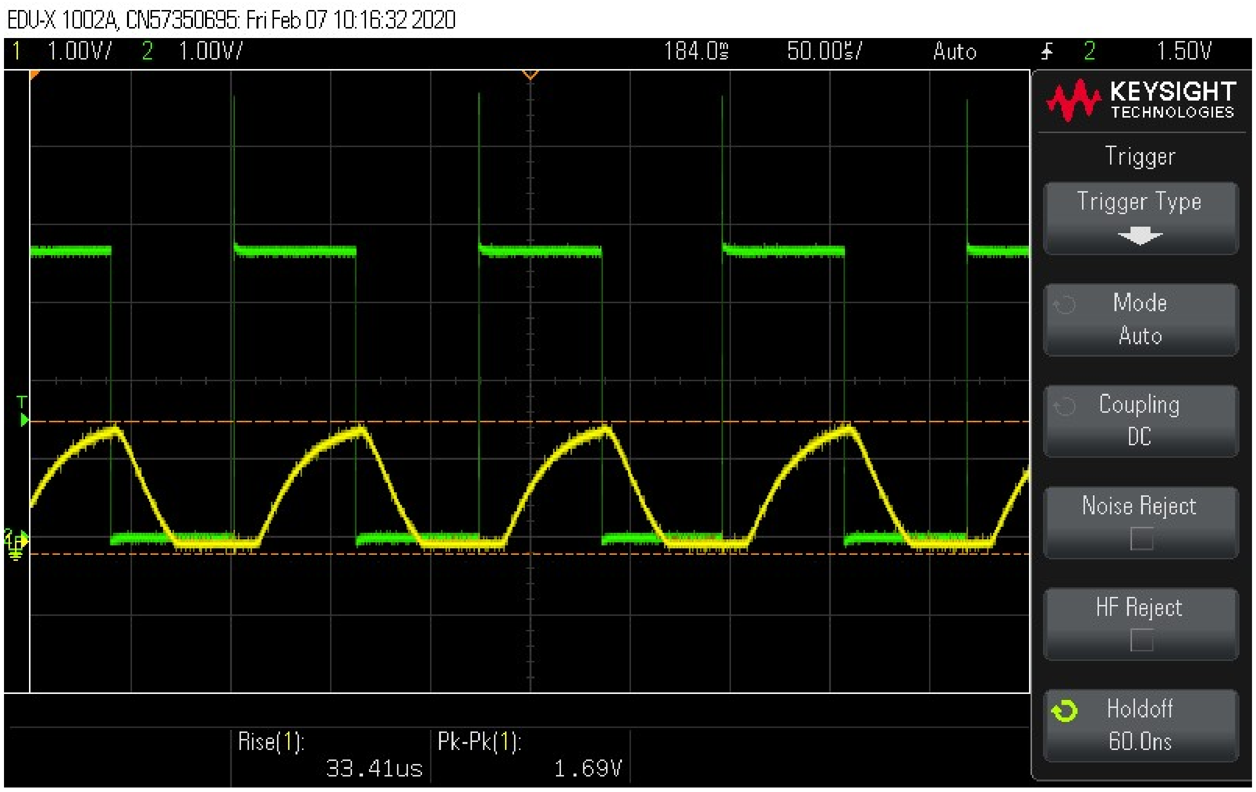


Figure 14: LM555 – MCP6004 set up

As you can see in figure 14, instead of using the function generator, I used LM555 instead. I managed to get a duty cycle of 49.8% and figure 15 shows what the waveform looks like from both the output of LM555 and the MCP6004.



EDU-X 1002A, CN57350695: Fri Feb 07 10:16:32 2020

1 1.00V/ 2 1.00V/ 184.0µs 50.00µs/ Auto 2 1.50V

Trigger: Trigger Type; Mode Auto; Coupling DC; Noise Reject; HF Reject; Holdoff 60.0ns

Rise(1): 33.41us Pk-Pk(1): 1.69V

Figure 15: LM555-MCP6004 setup with captouch sensor touched

Finally, for this combined circuit, we need to connect the output of the MCP6004 to one of the AD pins on Uno32 and write a code to detect if the sensor is being touched or not. This is quite simple to do—the output for MCP6004 is connected to AD_A3 on my Uno32. When I have the numbers printed out to the IO shield, there were a lot of noise—the readings were changing so quick that I couldn't find a proper threshold. I implemented a simple software filter to reduce the noise—each time the code will add 500 values together and output the average of those values. In this way, even though there might be some very large number jumps out in the middle, they will get averaged out. Next, I have another delay which will slow down the updating process of the reading and make the output more stable. The result is that the system can easily tell whether I have touched the sensor or no and print the result onto the IO shield.

The next part is the **Relaxation Oscillator**. In this part we will be building the relaxation oscillator using the LM555. Again, by following the astable mode diagram on the datasheet, I was able to build the circuit quickly. There is only a small change to the circuit, which was adding the capacitive touch sensor in parallel with the 22pF capacitor. Figure 16 is the diagram I draw for this relaxation oscillator along with the process of how I got the values for both resistors.

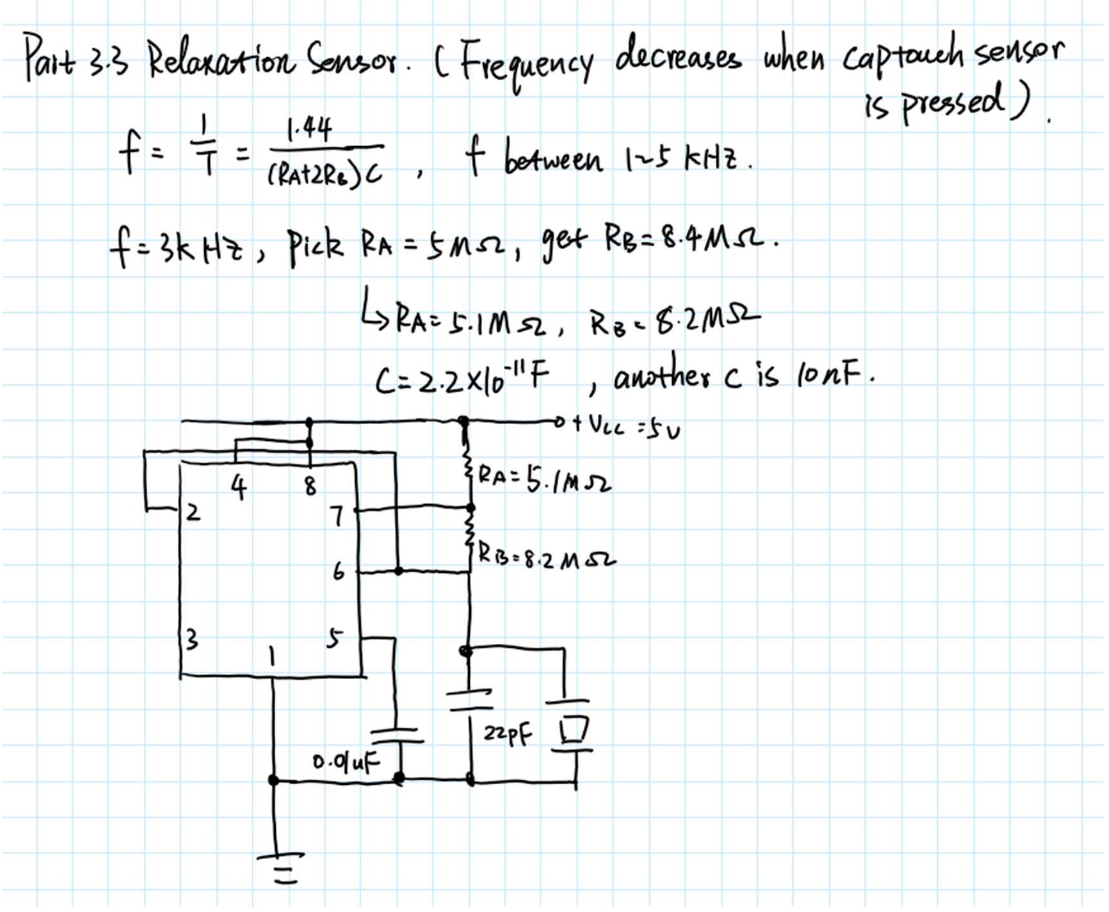
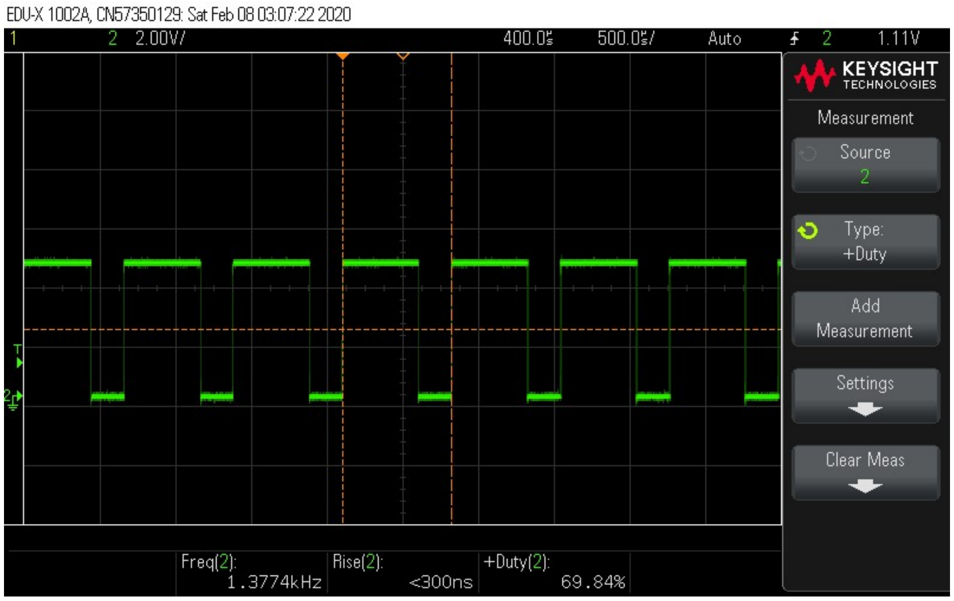

Part 3.3 Relaxation Sensor. (Frequency decreases when captouch sensor is pressed).
$f = \frac{1}{T} = \frac{1.44}{(R_A+2R_B)C}$, f between 1~5 kHz.
f = 3kHz, Pick RA = 5MΩ, get RB = 8.4MΩ.
↳ RA = 5.1MΩ, RB = 8.2MΩ
$C = 2.2\times10^{-11}F$, another C is 10nF.
+Vcc = 5v
RA = 5.1MΩ
RB = 8.2MΩ
4
8
2
7
6
3
1
5
0.01uF
22pF

Figure 16： Relaxation Oscillator circuit diagram

The formula that I used to calculate the resistance is taken from the datasheet directly. After powering up with the 5V power supply, I got a frequency of 1.3k Hz when the capacitive sensor was not touched and around 300Hz when the sensor is touched. In figure 17 & 18 you will also notice a change in the length of the pulse when the sensor is being touched.

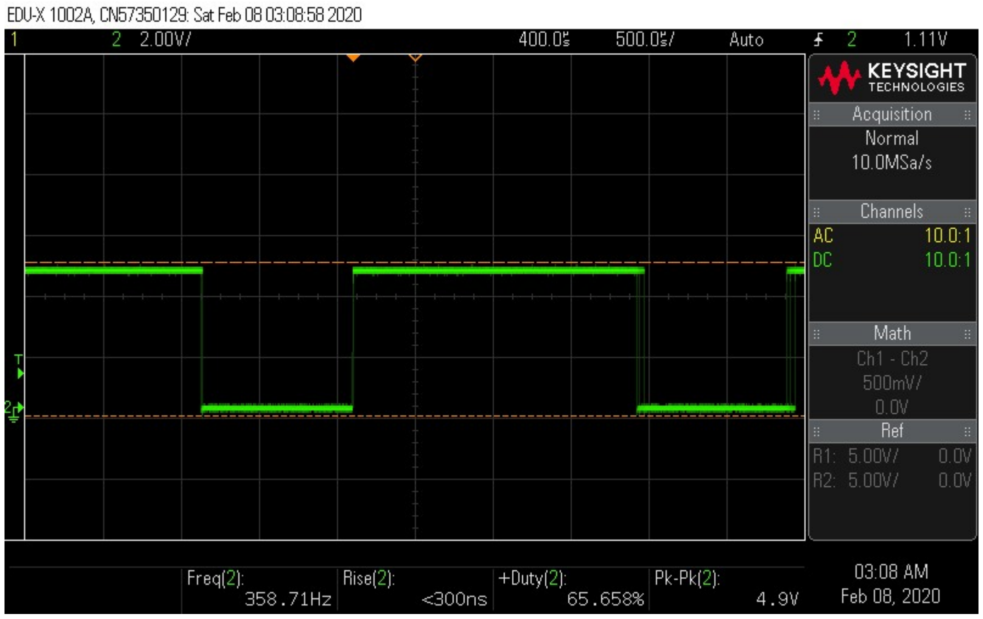

EDU-X 1002A, CN57350129: Sat Feb 08 03:07:22 2020

1 2 2.00V/ 400.0s 500.0s/ Auto 2 1.11V

Measurement
Source 2
Type: +Duty

Freq(2): 1.3774kHz
Rise(2): <300ns
+Duty(2): 69.84%

Figure 17: Relaxation Oscillator without touching


EDU-X 1002A, CN57350129: Sat Feb 08 03:08:58 2020

1 2 2.00V/ 400.0s 500.0s/ Auto 2 1.11V

Acquisition
Normal
10.0MSa/s

Channels
AC 10.0:1
DC 10.0:1

Math
Ch1 - Ch2
500mV/
0.0V

Ref
R1: 5.00V/ 0.0V
R2: 5.00V/ 0.0V

Freq(2): 358.71Hz
Rise(2): <300ns
+Duty(2): 65.658%
Pk-Pk(2): 4.9V

03:08 AM
Feb 08, 2020

Figure 18: Relaxation Oscillator with sensor touched

However, I was expecting an output frequency of 3000 Hz but I was only getting around 1.3kHz. I assume it should be a miscalculation in one of the resistor values but since I'm still within the frequency range (1k – 5k Hz), I decided to stick with the current setup.

Finally, the very last part of lab2 is to write another program that can detect whether the captouch sensor is being touched or not. This is similar to what I have done in the LM555-MCP6004 combination but this time I'll have to use the Input Capture ISR. To my own understanding, when the sensor is touched, the ISR will be triggered. There's a variable called IC4BUF that is provided which stores the timer value when the ISR is triggered. What I did is that I have two variables called last and current, and when ISR is triggered, the current timer reading will be stored in the "current". Then I would compare "current" with "last". When the ISR is triggered for the first time, the "last" variable will be 0, so I just simply updated it to be the same as "current". The next time ISR is triggered, there will be two different timer values. This is when I calculate the time difference. This time difference will then be used to determine whether the sensor is being touched or not—larger time difference means the sensor is being touched. When I was testing the program, I noticed that a very large reading (48xxxxx) would appear randomly and caused a lot of flickering for the determination process. The way I did to fix it is that I have another threshold, which is 10000. I would compare the difference between current time and the last time, and the time difference would be calculated only when (current time-last time) is less than the threshold.

<u>Result for part 3:</u>

The result of RC filter can be seen in figure 9, the waveform did change when I put my finger on the sensor. As for the LM555-MCP6004 combination circuit, I was able to observe a change in the waveform when the sensor is being touched and the duty cycle is almost 50% like I expected. For the LM555 relaxation oscillator, I did find myself having a different output frequency than expected but since it was still within the acceptable range, I let it go. The output is shown in figure 17&18. For the very last part, the captouch software interface, my program was able to identify if the sensor is being touched or not and print the result to the IO shield.

<u>Conclusion:</u>

This lab is way more time consuming than I expected it to be, but I did learn a lot, especially the various kinds of interrupts for getting the values from sensors. However, I'm still a little bit confused about the difference between the change notification and the input capture. Also, I spent most of my time in part 3, especially 3.2 (capacitive bridge and difference amp). I have found that he 50% duty cycle circuit provided in both LM555 datasheets is not working for me, and I'll have to find a difference circuit on the internet (after rebuilding several times). Other than that, this lab is fun since I get to work with different sensors.

<u>Acknowledgements:</u>

- Figure 1, https://cdn.sparkfun.com/datasheets/Components/Switches/EC12PLRGBSDVBF-D-25K-24-24C-6108-6HSPEC.pdf

Wiring diagrams:

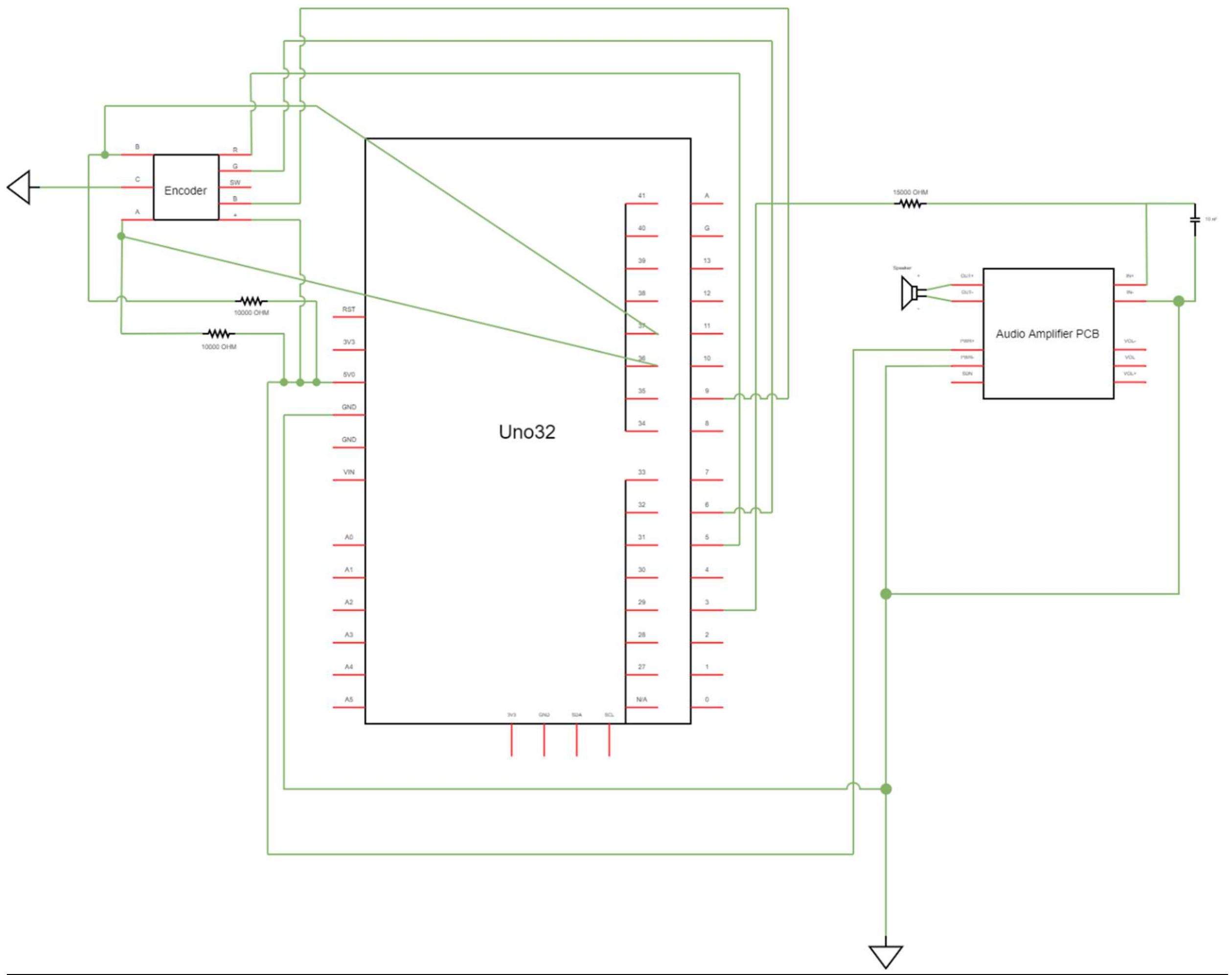


Figure 19: Part 1 Encoder wiring diagram

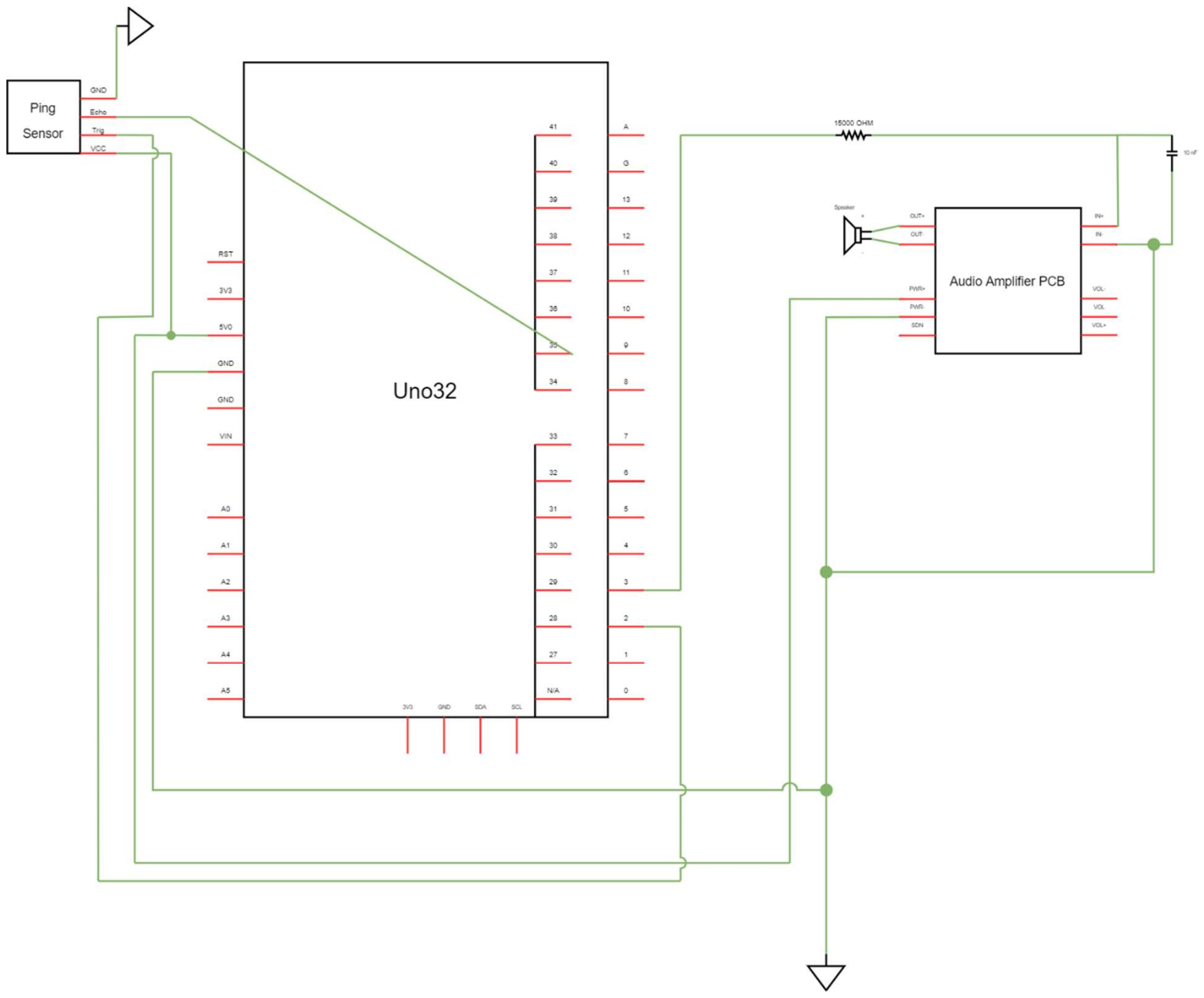


Figure 20: Part 2 Ping sensor wiring diagram