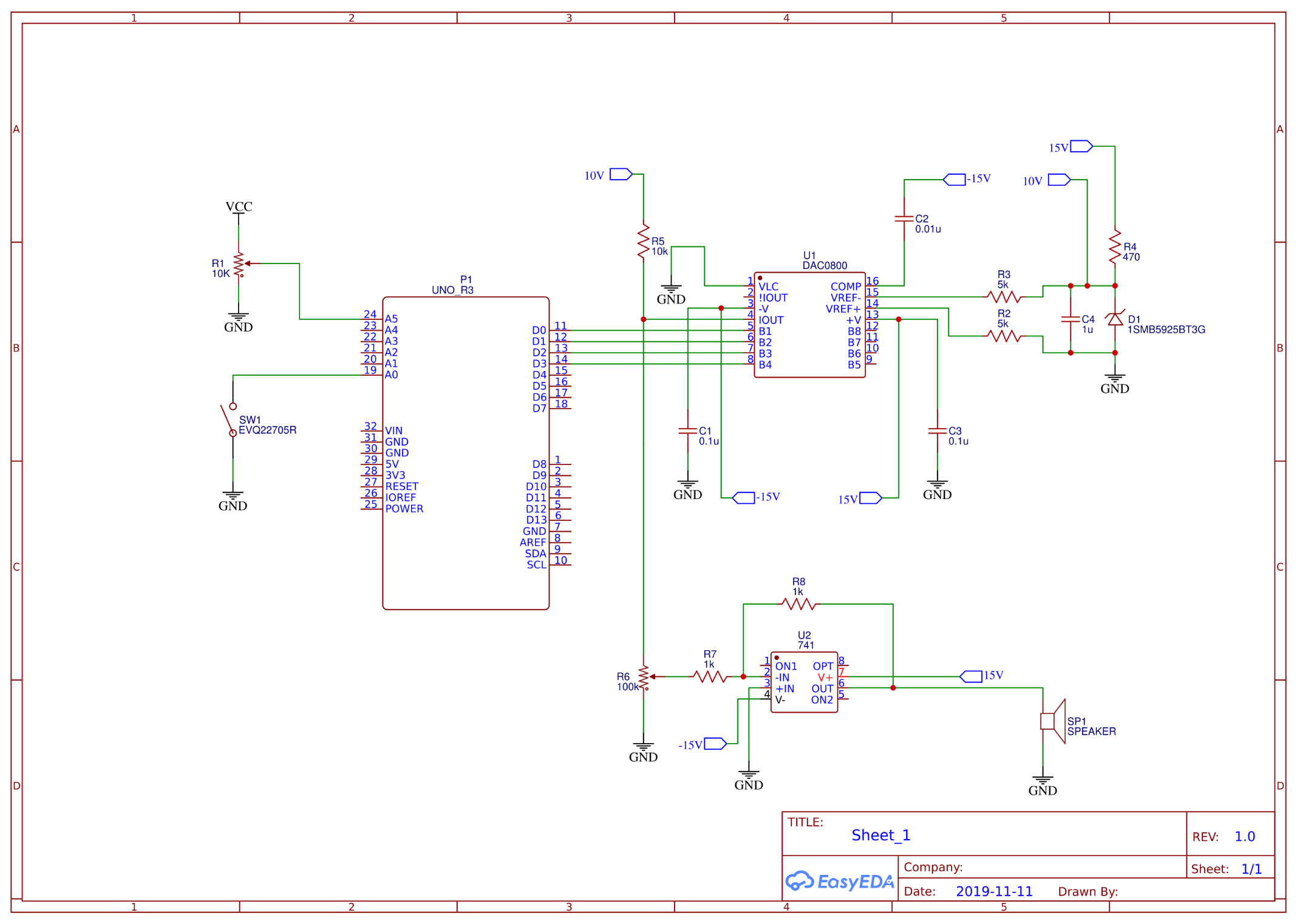
Javier Hernandez

Project proposal + submission – Arbitrary Waveform Generator

Data Conversion Fall 2019

Detailed circuit diagram

Bill of Materials

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  |  |  | | --- | --- | --- | --- | | ID | Name | Designator | Quantity | | 1 | 10K | R1 | 1 | | 2 | 100k | R6 | 1 | | 3 | 1SMB5925BT3G | D1 | 1 | | 4 | 0.1u | C1,C3 | 2 | | 5 | 0.01u | C2 | 1 | | 6 | 1u | C4 | 1 | | 7 | 741 OPAMP | U2 | 1 | | 8 | 5k | R2,R3 | 2 | | 9 | 470 | R4 | 1 | | 10 | 1k | R7,R8 | 2 | | 11 | 10k | R5 | 1 | | 12 | SPEAKER | SP1 | 1 | | 13 | DAC0800 | U1 | 1 | | 14 | UNO\_R3 | P1 | 1 | |  |  |  |  |

Summary

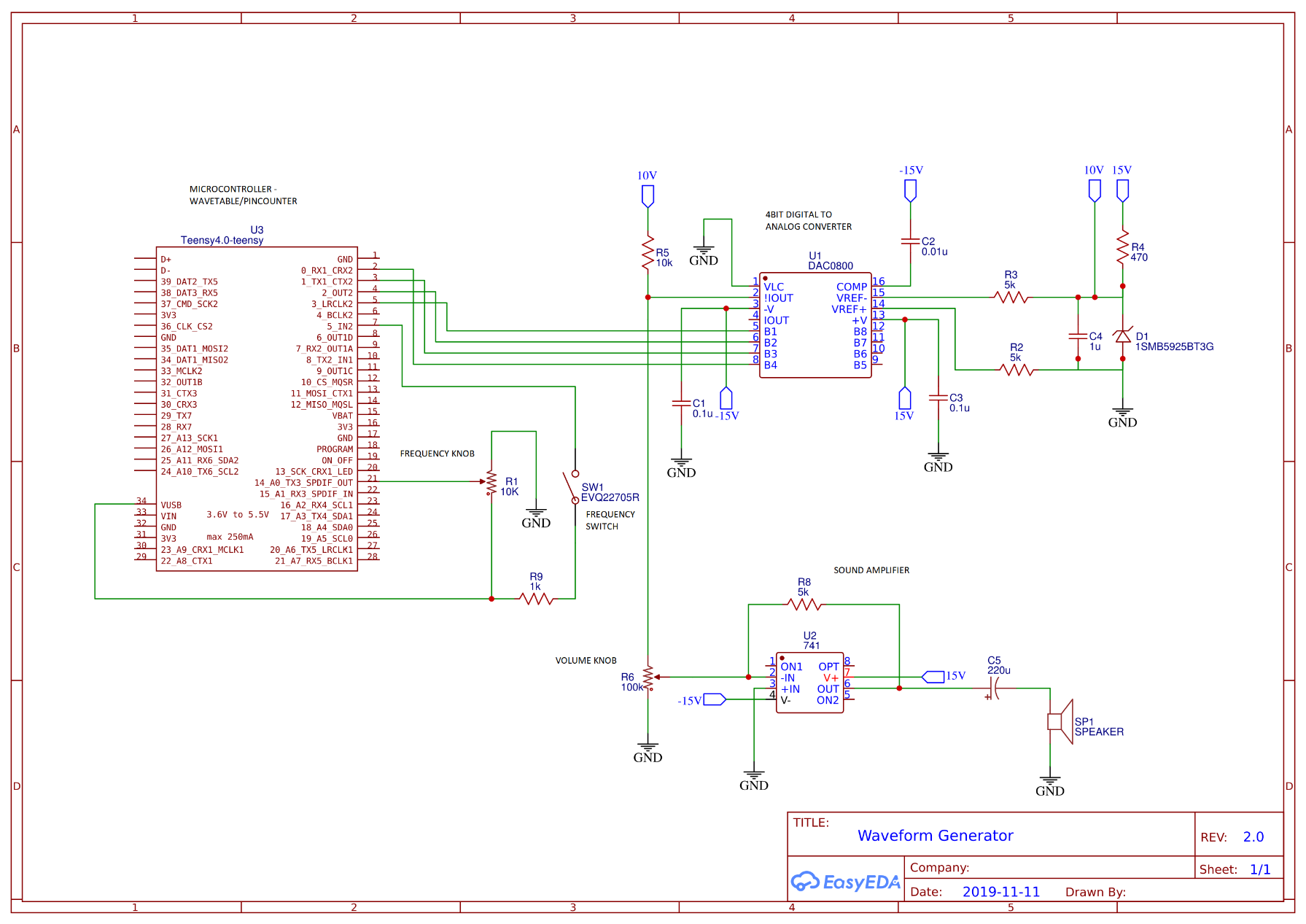
For this project, I will be using the Arduino Uno microcontroller along with the DAC0800 to create an arbitrary wave form generator. The project consists of four parts. A Wave table, position counter, DAC, speaker.

Four wave tables will be created using hardcoded HEX values and an extra optional wave table will be created to allow the user to enter a custom wave without reprogramming the microcontroller. On normal operation, the microcontroller will output a sawtooth wave and allows toggling other waves by pressing SW1, To allow the user to enter a custom wave an LCD to display current values of wave. A 4 bit encoder to allow the user to choose the desire value for the current entry. Two push button switch to scroll the values of the wave entry table and to enter the value of the 4 bit encoder into the current wave entry table and a toggle switch to enter the custom wave entry mode.

The position counter will read the entry table selected by SW1 and output a 4 bit value. R1 will be able to set the frequency of the wave from 20hz to 20Khz during the position counter procedure.

The DAC0800 will take the values from the microcontroller outputs and create an analog signal that will be used by the OP714 Amp in inverting mode. A possible second inverting amp might be needed to invert the signal again. The volume knob will allow the user to control the amplitude of the signal at the input of the amplifier.

Revised Circuit Schematic

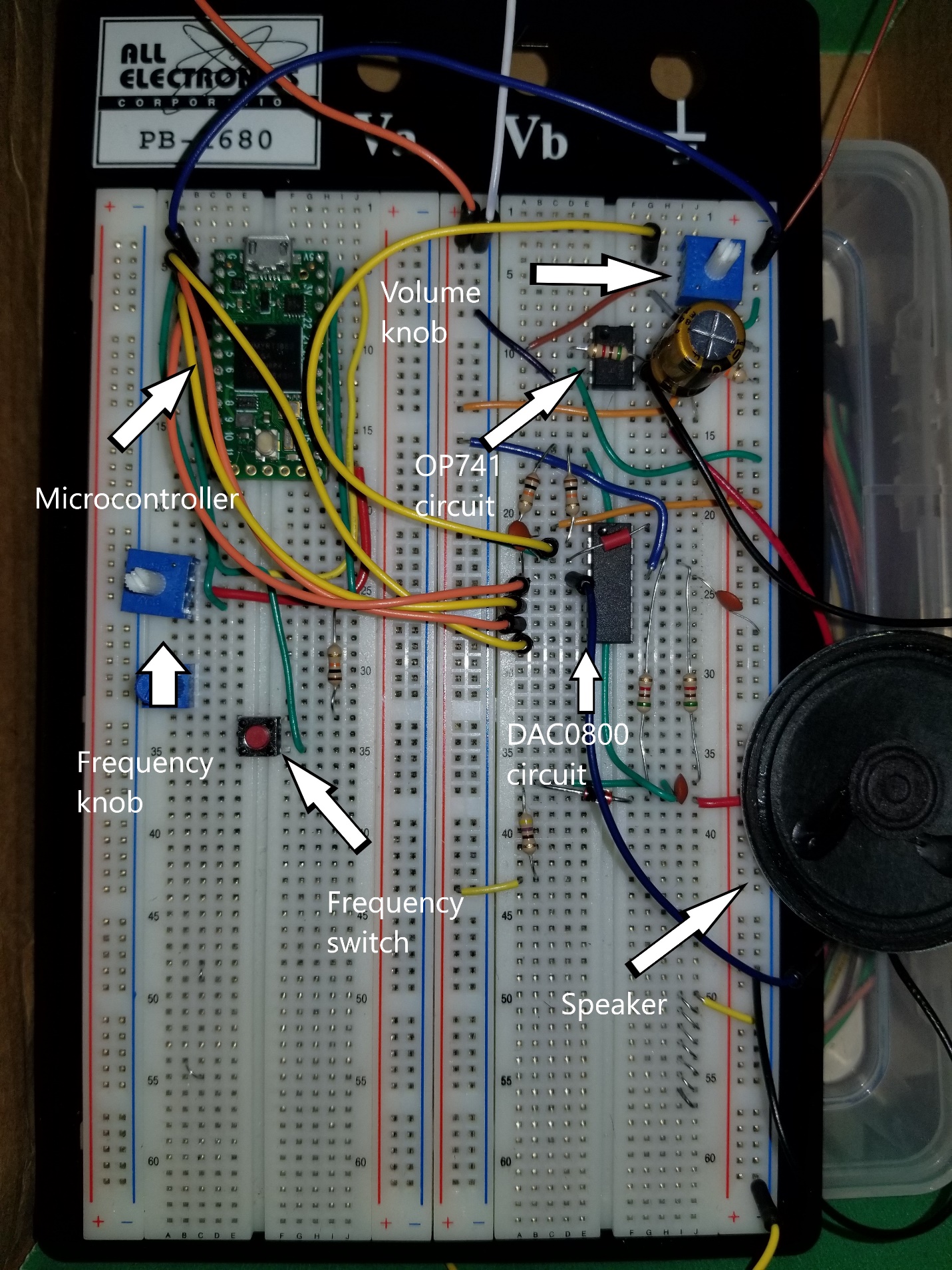


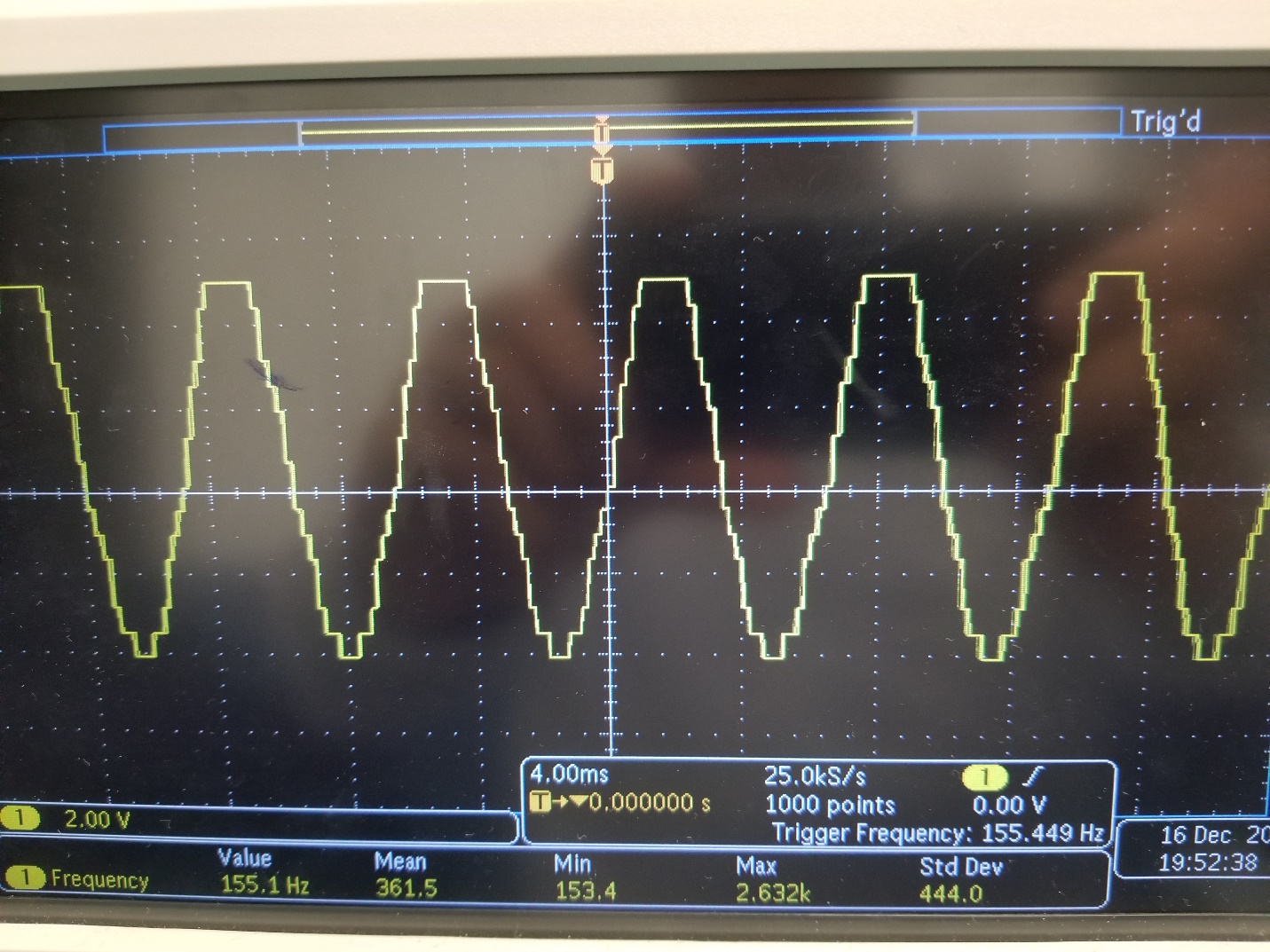
Revised Bill of Materials

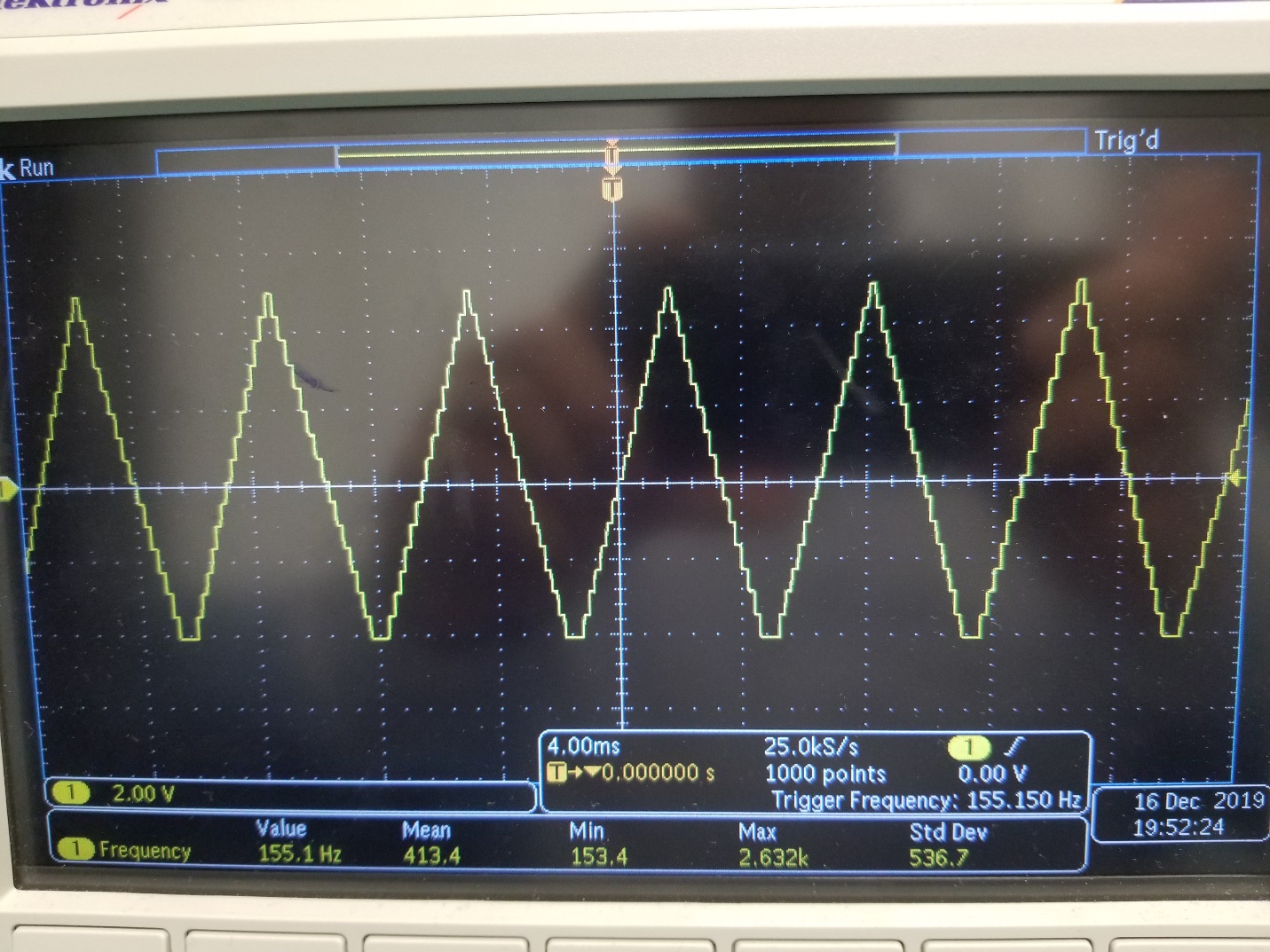


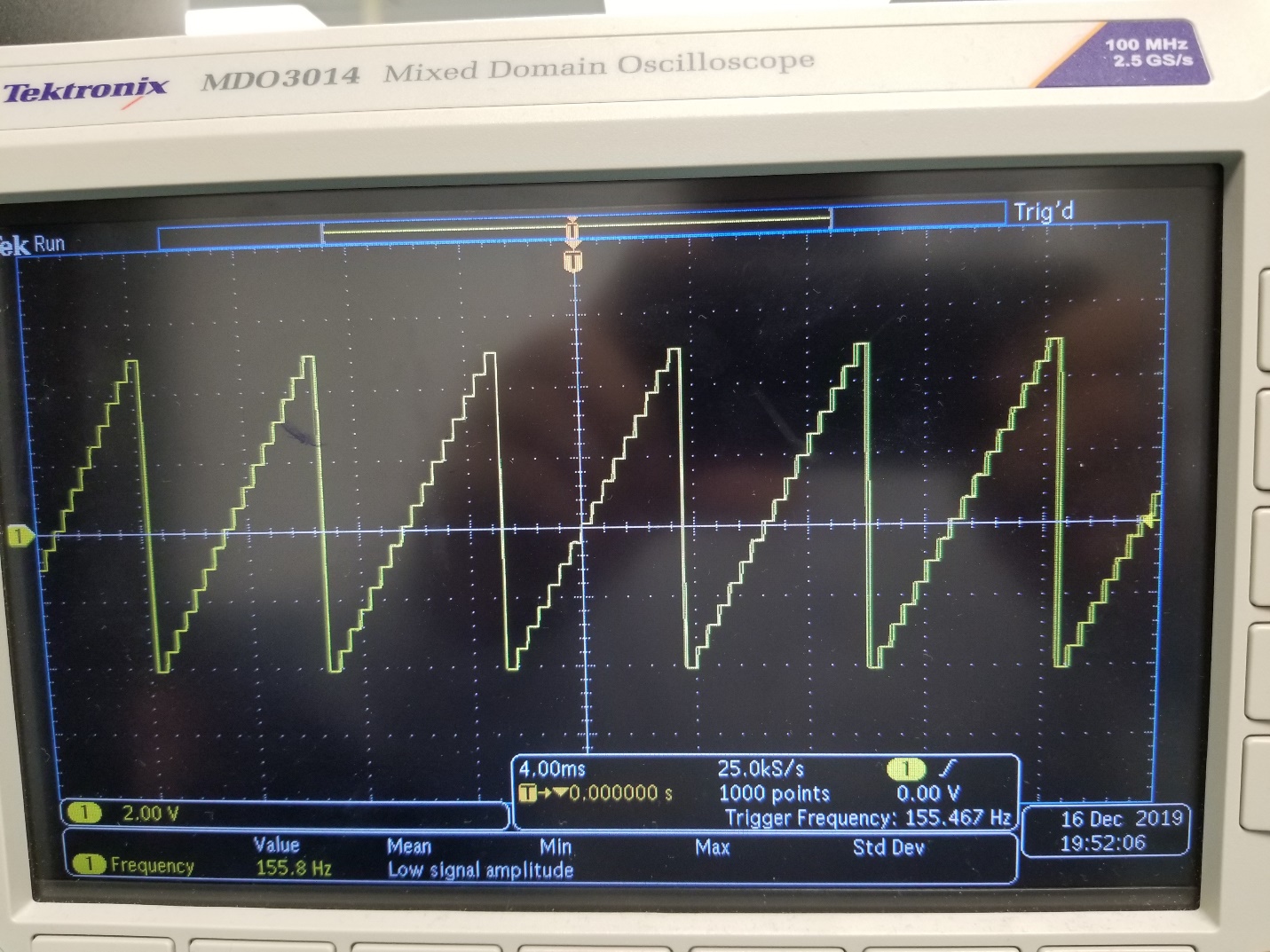
The microcontroller was changed from an Arduino to an Teensy 4.0 due to the Teensy having more processing power. The highest frequency achieved with the Arduino was of 1.5Khz while the Teensy can output a frequency of 19Khz.

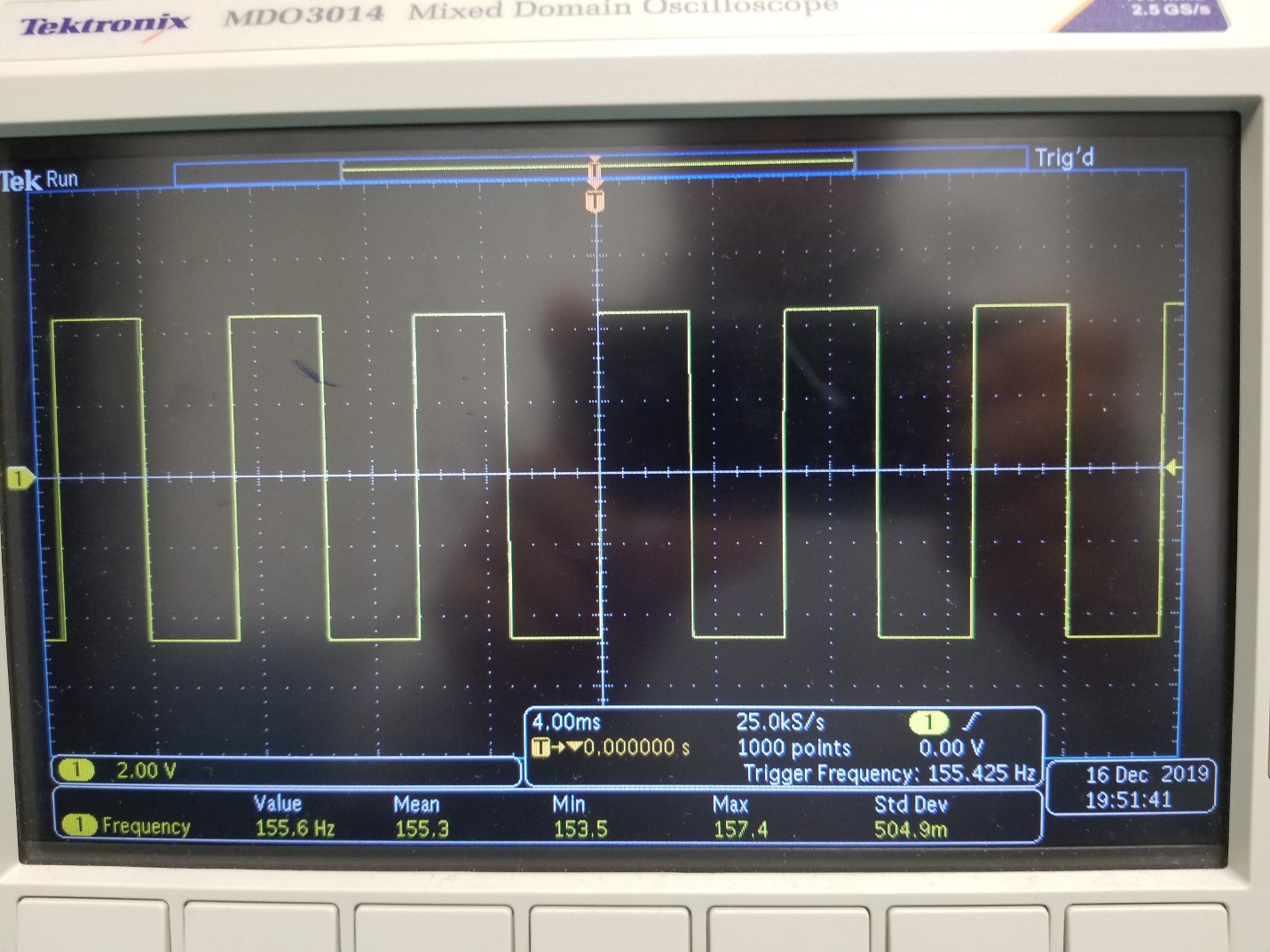
Prices provided by Digikey for trough hole type cut packaging components.

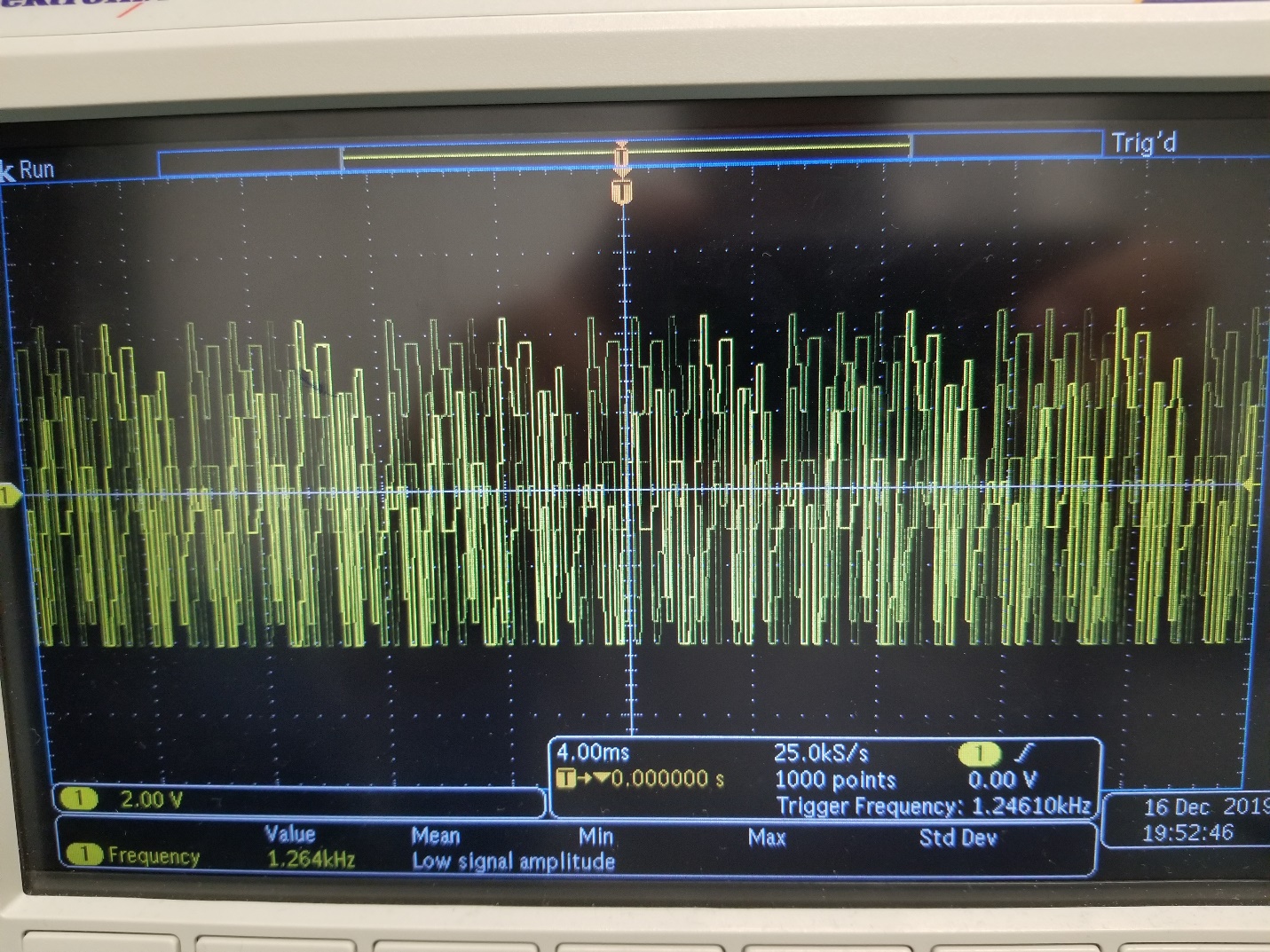
Photos

Sine Wave

Triangle Wave

Sawtooth Wave

Square Wave

White noise Wave

Implementation

The project consists of 4 parts that put together will play a sound representing an analog signal. The first part is the wave table which is a list of 32 entries that can hold a value between 0 to 16 or in this case 0x0 to 0xF in HEX values which we are using. Every value at each entry represents a different amplitude position of the signal at the speaker. Negative values are not allowed in the table.

The pin counter is the second part of the waveform generator. It is based on periodic reproduction of the wavetable using the microcontroller to store and read the wave table or better called, sampling the wave table. The counter reads the values of the wavetable at each entry and outputs a binary value from 0000 to 1111 at the output pins respective to the wave table hex value. Each output pin represents one of the bits of the read. The pin counter also has a delay function that activate every time a value is read from the wave table. The sum of this delay results in a frequency value from 20hz to 19khz. This range is achieved with the help of potentiometer R1 and a map function that give the delay function a value depending on the resistance at the potentiometer.

The main program also possesses a switch SW1 that allows the user to switch among waveforms in the memory. The waveforms are square, sawtooth, triangle, sine, and pseudo white noise. The switch function is activate using a push button to cycles trough the waveforms. The pseudo white noise waveform is created every time the waveform is cycled through. A random number generator assigns 32 random values from 0x0 to 0xF to the main entry but it does not update after reading it.

Part 3 is conversion from digital to analog is performed by the DAC0800 circuit. The circuit used is based on the same circuit from the typical application from the datasheet and lab 4. We only use the first four digital input to achieve the conversion and the settling time is of 100ns which gives the DAC circuit more than enough time to convert the digital signal inputted from the microcontroller since issues would only be noticeable at frequencies higher than 10Mhz.

Finally, The output of the DAC circuit is fed into an 741 op amp which acts as a sound amplifier for the speaker portion of the project. For this part, the sample circuit from lab4-figure 4 is used and a potentiometer R6 is added for control of signal amplitude of the U2 input signal. One of the potentiometer input comes from pin 2 of the DAC circuit. The other input goes to ground and the middle output is sent to pin 2. As the potentiometer knob is adjusted, the signal to the speaker circuit goes linearly from high to low to represent full volume to mute. To increase the volume of the speaker, a gain was added 400% using the formula below:

Vout = -(20kohms/5kohms)1.19Volts = -(4)1.19volts = 4.76Volts

Issues

Arduino UNO microcontroller has a poor processing power. I found out about while testing the code for the pincounter part of the project. When the pin counter reads a value from the wave table, it executes 4 digitalwrite functions from pin 0 to pin 3. These digitalwrite functions take 52 cycle instructions which severely hinder the speed at which the outputs can be written. Even when manipulating the input and outputs with direct IO register manipulation, the highest speed reached with the Arduino UNO was of 1.5Khz. The solution to this issue was to obtain a different microcontroller that, with minimal code modification, would process the pincounter functions much faster. The Teensy 4.0 was used in placed of the Arduino. The new output was clocked at 19Khz which is the highest attained frequency when using the delaymicroseconds function for period manipulation.

The speaker signal was also very noisy and would not be loud enough to hear when using the output of the OP741 to drive the speaker. The solution to this problem was to add a polarized capacitor in series with the speaker. This ensure an even flow of voltage.

Even at high frequencies, noise was able to be heard. The speaker was receiving noise signals from other sources that I was not able to find the origin. This noise could be DC noise, thermal noise, or radio waves. To filter this noise a low pass filter would be required. Using a capacitor of 0.047uf, the optimal resistor value would be:

R = 1/(2 \* pi \* 0.047uf \* 20khz) = 169ohms

Code

// Array to hold hex values

int currentEntry[32]= {0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,

0xF,0xF,0xF,0xF,0xF,0xF,0xF,0xF,0xF,0xF,0xF,0xF,0xF,0xF,0xF,0xF};

int currentEntrySize = 32;

// timer variable to hold period/32

double timer = 1;

// variables to cycle trough the different waves

int button\_state = 0;

int lastbutton\_state = 0;

int wave\_state = 0;

unsigned long lastDebounceTime = millis();

unsigned long debounceDelay = 250;

//declaration of helping functions

void SawTooth();

void SquareWave();

void TriangleWave();

void SineWave();

void PinCounter();

void setup() {

Serial.begin(9600);

pinMode(0,OUTPUT);

pinMode(1,OUTPUT);

pinMode(2,OUTPUT);

pinMode(3,OUTPUT);

pinMode(A0, INPUT); //input potentiometer for frequency

pinMode(5, INPUT); //input pushbutton toggle waves

pinMode(13, OUTPUT);

}

void loop() {

// reading button and cyling waves with switch statement

button\_state = digitalRead(5);

if ((millis() - lastDebounceTime) > debounceDelay) {

if (button\_state != lastbutton\_state){

if (button\_state == HIGH){

wave\_state += 1;

switch(wave\_state){

case 0: SquareWave(); Serial.print("Square \n"); break;

case 1: SawTooth(); Serial.print("SawTooth \n"); break;

case 2: TriangleWave(); Serial.print("Triangle \n"); break;

case 3: SineWave(); Serial.print("Sine \n"); break;

case 4: WhiteWave(); Serial.print("White Noise \n"); break;

}

if (wave\_state > 4){

wave\_state = -1;

}

}

}

lastbutton\_state = button\_state;

}

//Adjusting timer at loop() will allow the wave to be updated after the total period is over

timer = map(analogRead(A0), 0 , 1023, 1 , 1500);

PinCounter();

}

// Waves Functions

void SquareWave(){

memset(currentEntry, 0, sizeof(currentEntry));

int SquareWave[32] = {0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,

0xF,0xF,0xF,0xF,0xF,0xF,0xF,0xF,0xF,0xF,0xF,0xF,0xF,0xF,0xF,0xF};

for (int i = 0; i < 32; ++i){

currentEntry[i] = SquareWave[i];

}

}

void SawTooth(){

memset(currentEntry, 0, sizeof(currentEntry));

int SawTooth[32] = {0x0,0x0,0x1,0x1,0x2,0x2,0x3,0x3,0x4,0x4,0x5,0x5,0x6,0x6,0x7,0x7,

0x8,0x8,0x9,0x9,0xA,0xA,0xB,0xB,0XC,0XC,0XD,0XD,0XE,0XE,0XF,0XF};

for (int i = 0; i < 32; ++i){

currentEntry[i] = SawTooth[i];

}

}

void TriangleWave(){

memset(currentEntry, 0, sizeof(currentEntry));

int Triangle[32] = {0x0,0x1,0x2,0x3,0x4,0x5,0x6,0x7,0x8,0x9,0xA,0xB,0xC,0xD,0xE,0xF,

0xE,0xD,0xC,0xB,0xA,0x9,0x8,0x7,0X6,0X5,0X4,0X3,0X2,0X1,0X0,0X0};

for (int i = 0; i < 32; ++i){

currentEntry[i] = Triangle[i];

}

}

void SineWave(){

memset(currentEntry, 0, sizeof(currentEntry));

int Sine[32] = {0x7,0x9,0xb,0xc,0xe,0xf,0xf,0x10,0x10,0x10,0xf,0xf,0xe,0xc,0xb,0xa,

0x8,0x6,0x5,0x4,0x2,0x1,0x1,0x0,0x0,0x0,0x1,0x1,0x2,0x4,0x5,0x6};

for (int i = 0; i < 32; ++i){

currentEntry[i] = Sine[i];

}

}

void WhiteWave(){

memset(currentEntry, 0, sizeof(currentEntry));

int White[32];

for (int i = 0; i < 32; ++i){

White[i] = random(0x0,0xF);

currentEntry[i] = White[i];

}

}

void PinCounter(){

for(int i = 0; i < 32; ++i){

if(i == 32){

i = 0;

}

switch(currentEntry[i]){

case 0x0: digitalWriteFast(3,LOW); digitalWriteFast(2,LOW);digitalWriteFast(1,LOW);digitalWriteFast(0,LOW); break;

case 0x1: digitalWriteFast(3,LOW); digitalWriteFast(2,LOW);digitalWriteFast(1,LOW);digitalWriteFast(0,HIGH); break;

case 0x2: digitalWriteFast(3,LOW); digitalWriteFast(2,LOW);digitalWriteFast(1,HIGH);digitalWriteFast(0,LOW); break;

case 0x3: digitalWriteFast(3,LOW); digitalWriteFast(2,LOW);digitalWriteFast(1,HIGH);digitalWriteFast(0,HIGH); break;

case 0x4: digitalWriteFast(3,LOW); digitalWriteFast(2,HIGH);digitalWriteFast(1,LOW);digitalWriteFast(0,LOW); break;

case 0x5: digitalWriteFast(3,LOW); digitalWriteFast(2,HIGH);digitalWriteFast(1,LOW);digitalWriteFast(0,HIGH);break;

case 0x6: digitalWriteFast(3,LOW); digitalWriteFast(2,HIGH);digitalWriteFast(1,HIGH);digitalWriteFast(0,LOW); break;

case 0x7: digitalWriteFast(3,LOW); digitalWriteFast(2,HIGH);digitalWriteFast(1,HIGH);digitalWriteFast(0,HIGH); break;

case 0x8: digitalWriteFast(3,HIGH); digitalWriteFast(2,LOW);digitalWriteFast(1,LOW);digitalWriteFast(0,LOW); break;

case 0x9: digitalWriteFast(3,HIGH); digitalWriteFast(2,LOW);digitalWriteFast(1,LOW);digitalWriteFast(0,HIGH); break;

case 0xA: digitalWriteFast(3,HIGH); digitalWriteFast(2,LOW);digitalWriteFast(1,HIGH);digitalWriteFast(0,LOW); break;

case 0xB: digitalWriteFast(3,HIGH); digitalWriteFast(2,LOW);digitalWriteFast(1,HIGH);digitalWriteFast(0,HIGH); break;

case 0xC: digitalWriteFast(3,HIGH); digitalWriteFast(2,HIGH);digitalWriteFast(1,LOW);digitalWriteFast(0,LOW); break;

case 0xD: digitalWriteFast(3,HIGH); digitalWriteFast(2,HIGH);digitalWriteFast(1,LOW);digitalWriteFast(0,HIGH); break;

case 0xE: digitalWriteFast(3,HIGH); digitalWriteFast(2,HIGH);digitalWriteFast(1,HIGH);digitalWriteFast(0,LOW); break;

case 0xF: digitalWriteFast(3,HIGH); digitalWriteFast(2,HIGH);digitalWriteFast(1,HIGH);digitalWriteFast(0,HIGH); break;

}

delayMicroseconds(timer);

}

}