Function Generator Project

Student:

Jay Darshan Vakil, Evan DeTienne

Teacher:

Dr Arnie Berger, Dr Tyler Folsom

Course:

B EE 425: Microprocessor system design

TABLE OF CONTENTS

ABSTRACT

This is the capstone of the class; it binds together whatever we learnt throughout the class and make a real-world application. Our task for the project was to build our own function generator that outputs sine, square and triangle waves! It is designed to teach us the fundamentals of working from the ground up; Learning how to design a prototype from a list of specifications, design a proper schematic and PCB, order all components and parts, and then solder and test!

INTRODUCTION

The design project is the core of our EE 425 class! The project allows us to show off what we have learned from our class and from our EE program. For this project, we will be designing and building our own function generator! We will be translating a generalized algorithmic problem statement into a microprocessor-based design, we will choose the components that create the best solution for our design specifications, and we will be using a modern CAD design tool to create the schematic diagram!

MATERIALS

For this project we used the following materials:

- Arduino nano
- AD9833 module
- NE5534 OP-Amp
- ICL7660 Switching voltage regulator
- 9-volt power supply
- Resistors (10k Ω, 14k Ω)
- 100k Potentiometer
- Capacitors (10μF)
- Rotary encoder
- EasyEDA schematic and PCB drawing tools
- JLC custom made PCB

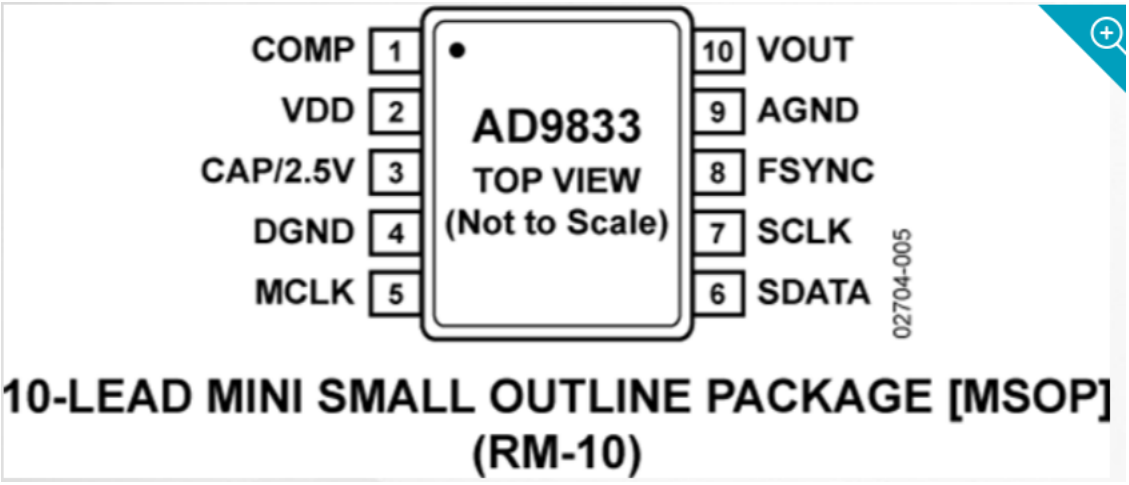
PROCEDURE AND TESTING

RESEARCH AND PLANNING

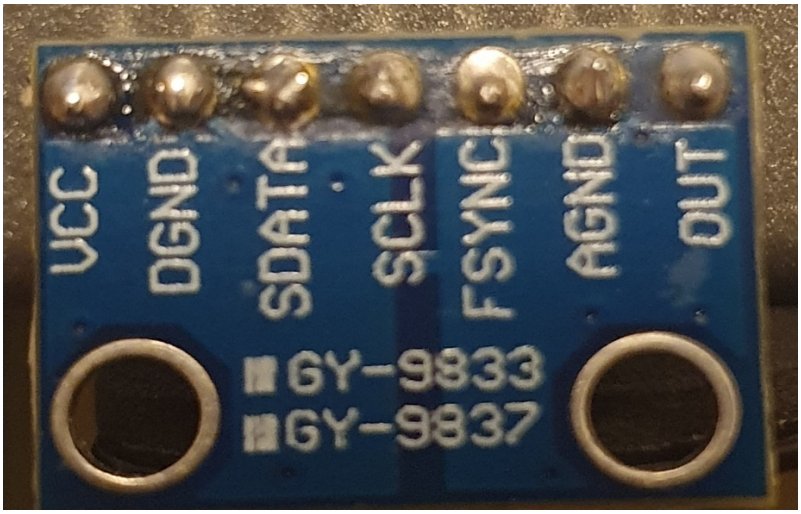
We were formally introduced to this project by Dr Berger in the first lecture of B EE 425. We were told to build our own function generator that can output sine, square, and triangle waves with amplitude +/- 5V and frequency 10Hz to 100KHz. The first step we took after hearing the assignment was to google "How to build a microcontroller-based function generator", many results came upon but that wouldn't be learning so we decided to go step by step. We first researched for a wave generator circuit, but we found something even better, a programmable chip that can generate sine, square, and triangle waves. This chip was the AD9833 chip (AD9833). Once we decided on this chip, we wanted a microcontroller that could program the chip, we chose the Arduino nano because it was smaller in size and more forums and help was available online compared to other microcontroller boards like Texas Instrument's Tiva. According to the datasheet, the output from the AD9833 chip would be 38mV to 650mV but according to the project requirements, we need an output of 5V, so we needed a gain of approximately 8. To get the gain we decided to use an opamp with a passive and active RC circuit to reduce the offset voltage. Then to get the inverting voltage to power the opamp, we used the IC ICL7660 inverting voltage regulator recommended to us by Dr Berger to invert the 9V.

DESIGN

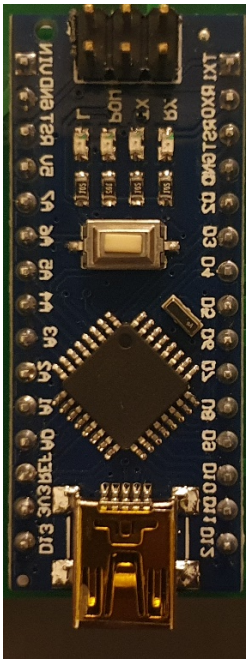
After doing the research to get the parts we need and getting their specifications we decided to lay them out and make the connections required. First, we started with the AD9833 chip.



THE OUTLINE OF IC CHIP AD9833 (AD9833).



THE PICTURE OF OUR AD9833 MODULE WE BOUGHT FROM AMAZON.COM.

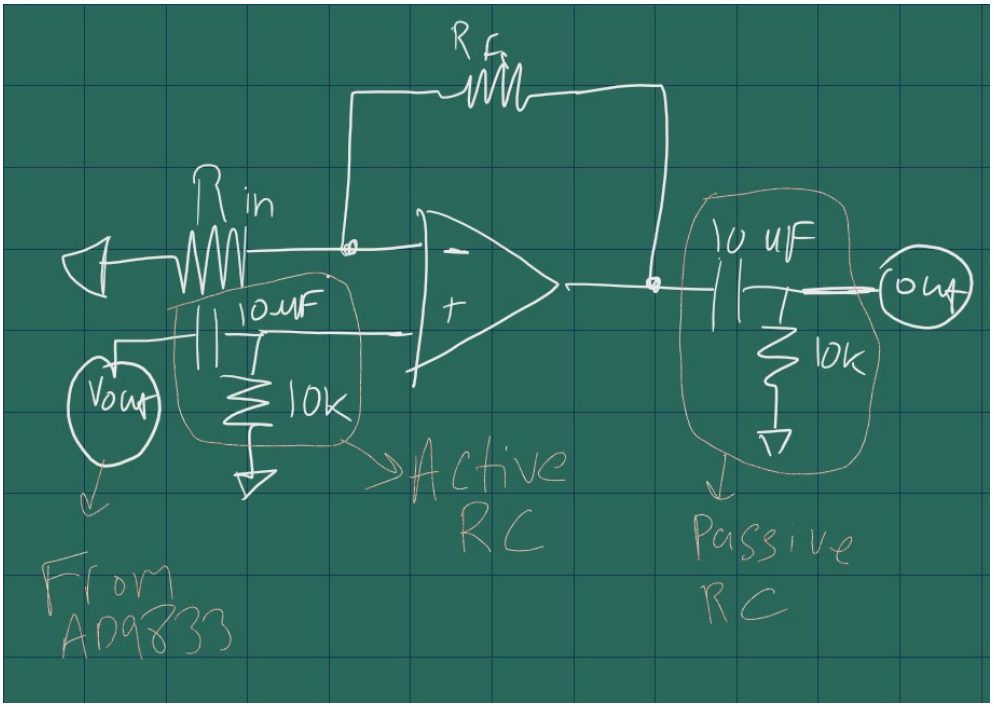


A PICTURE OF OUR ARDUINO NANO BOARD.

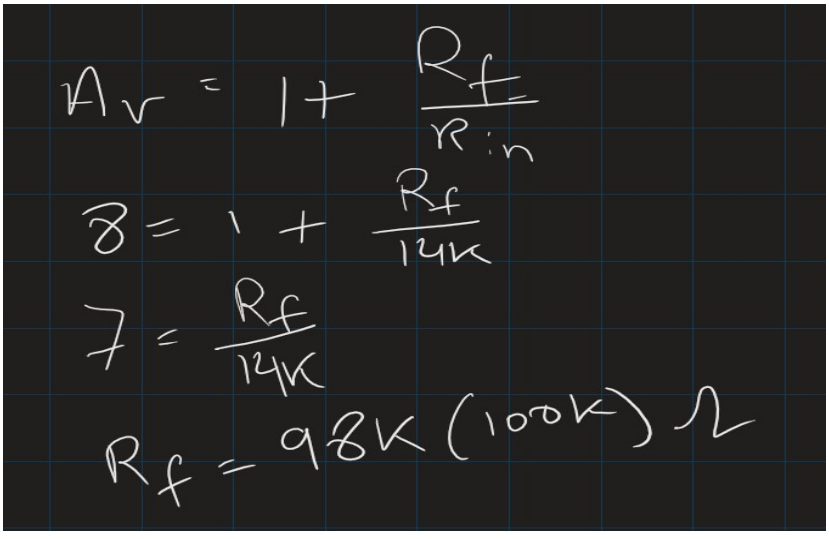
We powered the Arduino through the Vin pin which can take an input voltage of 7V-12V, and as we were supplying it with a 9V voltage supply we were under the maximum voltage. We would connect the ground(gnd) pin of the Arduino to the ground of the power supply. Then, we connected the Vcc of the Ad9833 module to the 5V pin of the Arduino, and the ground pin of the Arduino to the DGND pin of the AD9833 module. Then we connected the SData pin of the AD9833 module to the digital pin 11 of the Arduino. The Sclk pin of the AD9833 module would be connected to the digital pin 13 of the Arduino. The Fsync of the AD9833 would be connected to the digital pin 10 of the Arduino. The AGnd could be

connected to the same ground pin of the Arduino but the skepticism of shorting the grounds made us connect the AGnd to the ground of the power supply. The last pin Vout is the output pin that would output that would generate the sine, square and triangle waves, this pin would be the non-inverting input of the NE5534 opamp to boost the signal to +/-5V.

The next major designing we had to do was to build a non-inverting op-amp circuit that fulfilled the gain requirements without clipping the output signal. Using the opamp non-inverting circuit below, we calculated for the values of Rin and Rf.



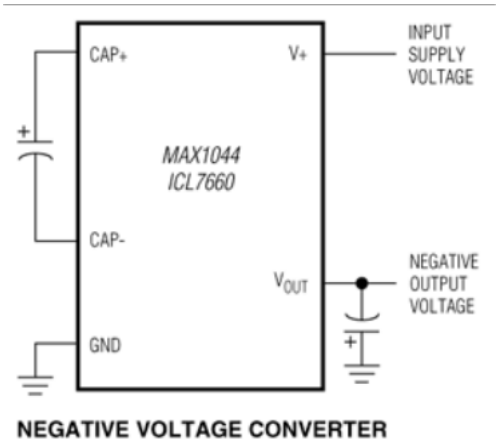
OUR OPAMP NON-INVERTING CIRCUIT.



OUR CALCULATIONS TO FIND THE RESISTOR VALUES.

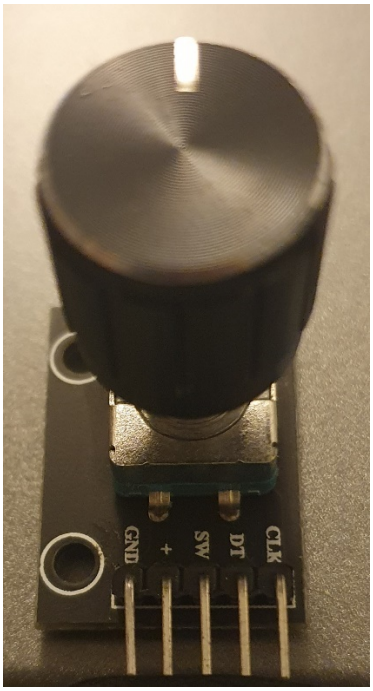
We assumed the Rin to be 14kΩ and then used the non-inverting amplifier's gain formula to calculate for the Rf. When our gain is 8, the Rf results to approximately100kΩ. 100kΩ is when the output is at 5V, so when we want to lower the amplitude, we can lower the Rf resistance, so we decided to use a 100kΩ potentiometer.

For the ICL7660, the schematic was already given by the manufacturer, MaximIntegrated.



THIS IS THE CIRCUIT SCHEMATIC FOR THE NEGATIVE VOLTAGE REGULATOR CIRCUIT (ICL7660).

The next designing to do was the rotary encoder. The rotary encoder is to control the frequency of the output wave.



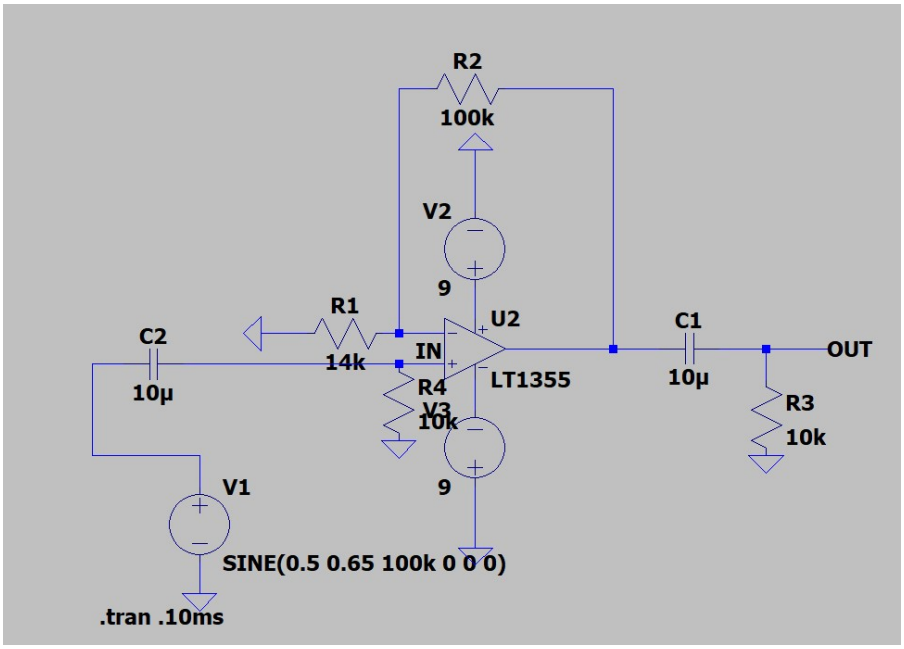
A PICTURE OF OUR ROTARY ENCODER

The clock of the rotary encoder is connected to the digital pin 4 of the Arduino. Then the DT pin of the encoder is connected to the digital pin 5 of the Arduino, and the switch which is the button on the rotary encoder is connected to the digital pin 6 of the Arduino. The Vcc (+) and the ground of the encoder are connected to the 5V and ground on the Arduino respectively.

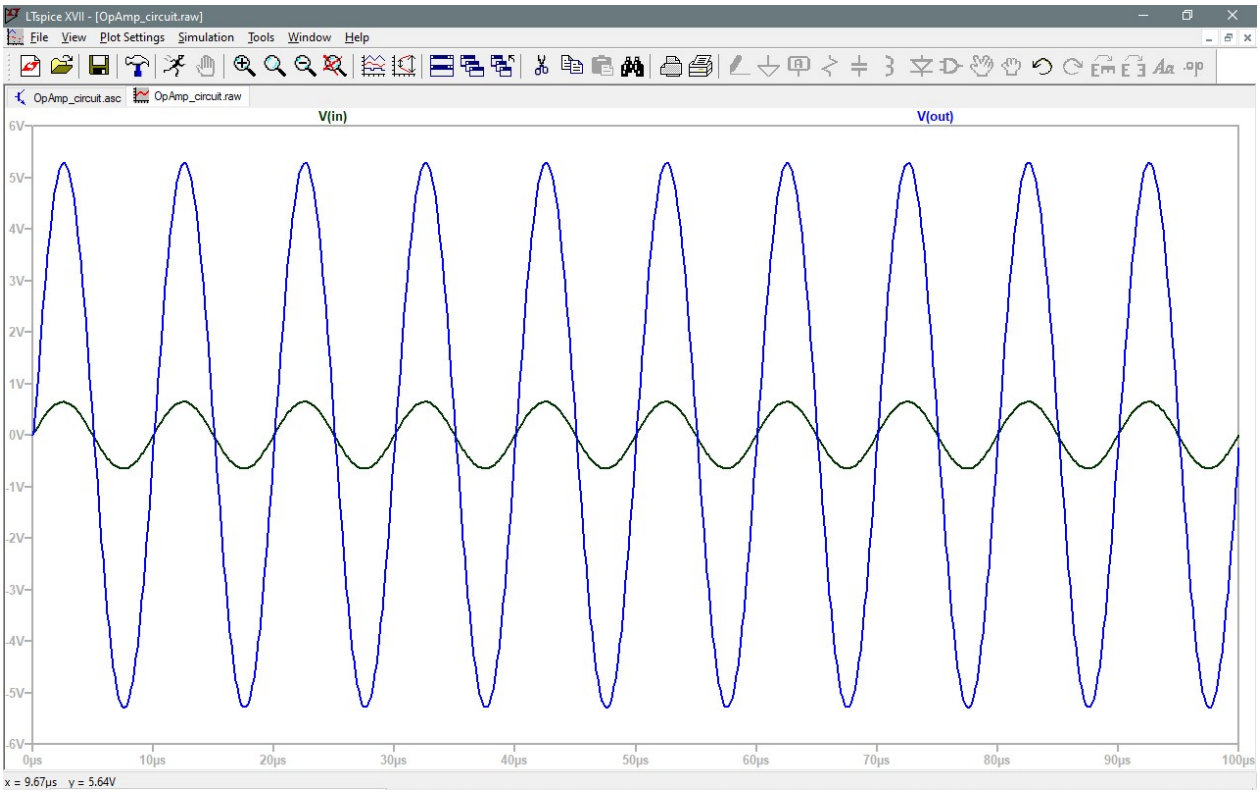
SIMULATION

After the designing the circuit, we decided to test if it outputs what we want by simulating the circuit in LTspice. We set the AD9833 output signal to a sine wave with input signal of 650mV, frequency of 100Khz and an offset of 500mV.

R1 is the Rin 14k, R2 is the Rf 100k, C2 and R4 pair is the active RC circuit, C1 and R3 pair is the passive RC filter circuit. LT1355 opamp is used instead of NE5534 opamp because they have similar characteristics and NE5534 is not available in the LTspice directory.



THIS IS OUR LTSPICE CIRCUIT.

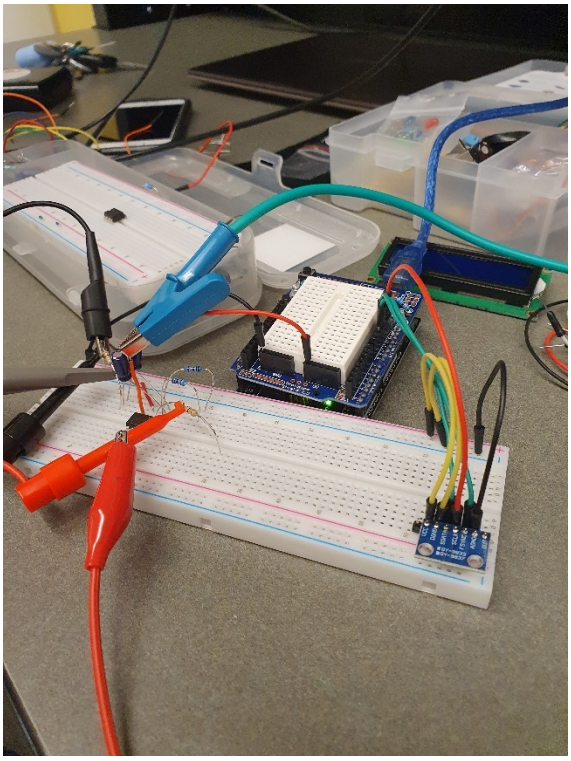


THIS IS OUR LTSPICE RESULTS

After we ran the simulation, we could see that an input of 650mV resulted in approximately 5V of output, resulting in a gain of about 8. The black signal is the input signal from the AD9833, and the blue signal is the output signal from the opamp. Once we confirmed our design in LTspice, we moved on to testing the circuit on a breadboard.

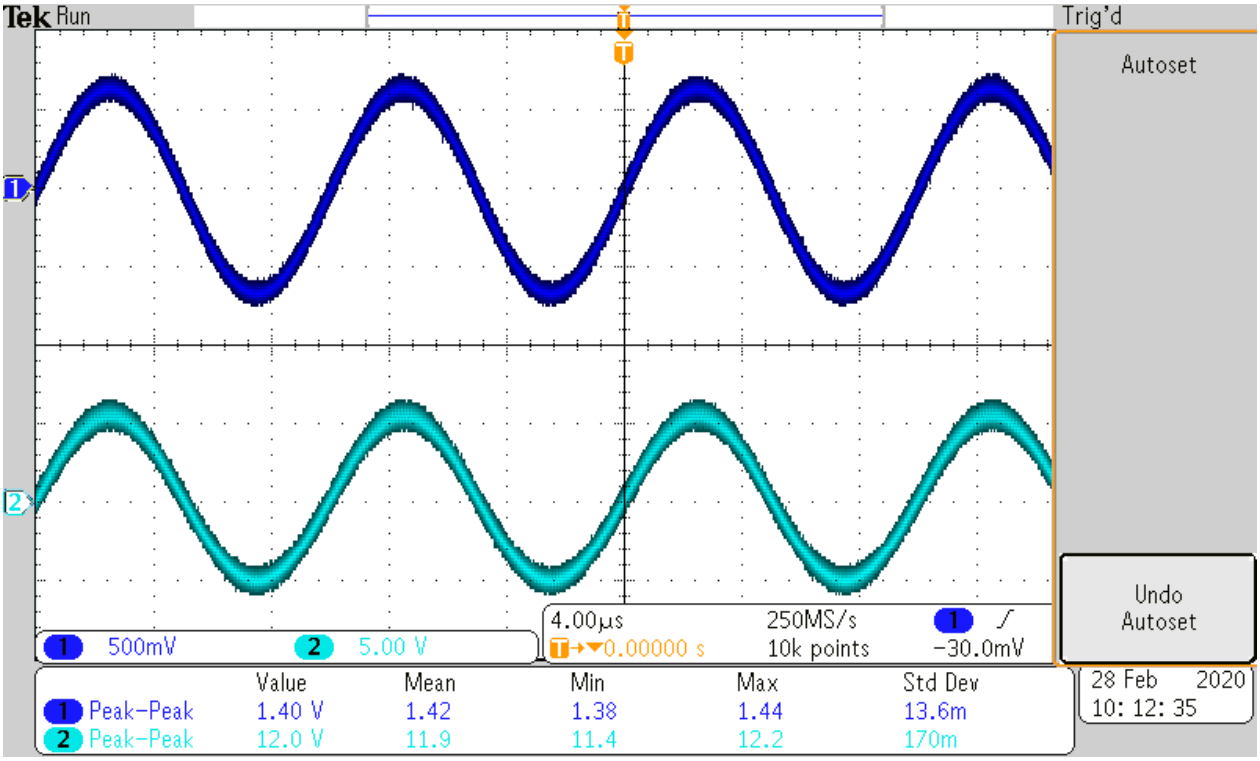
TESTING ON A BREADBOARD

We assembled the opamp circuit on a breadboard along with the ad9833 module. We used an Arduino uno for testing so that there is more space on the breadboard to test upon, but for the actual design we are still going with the Arduino nano. Both models use the same microprocessor, Atmel ATmega 328p, the only difference between the two is the size.



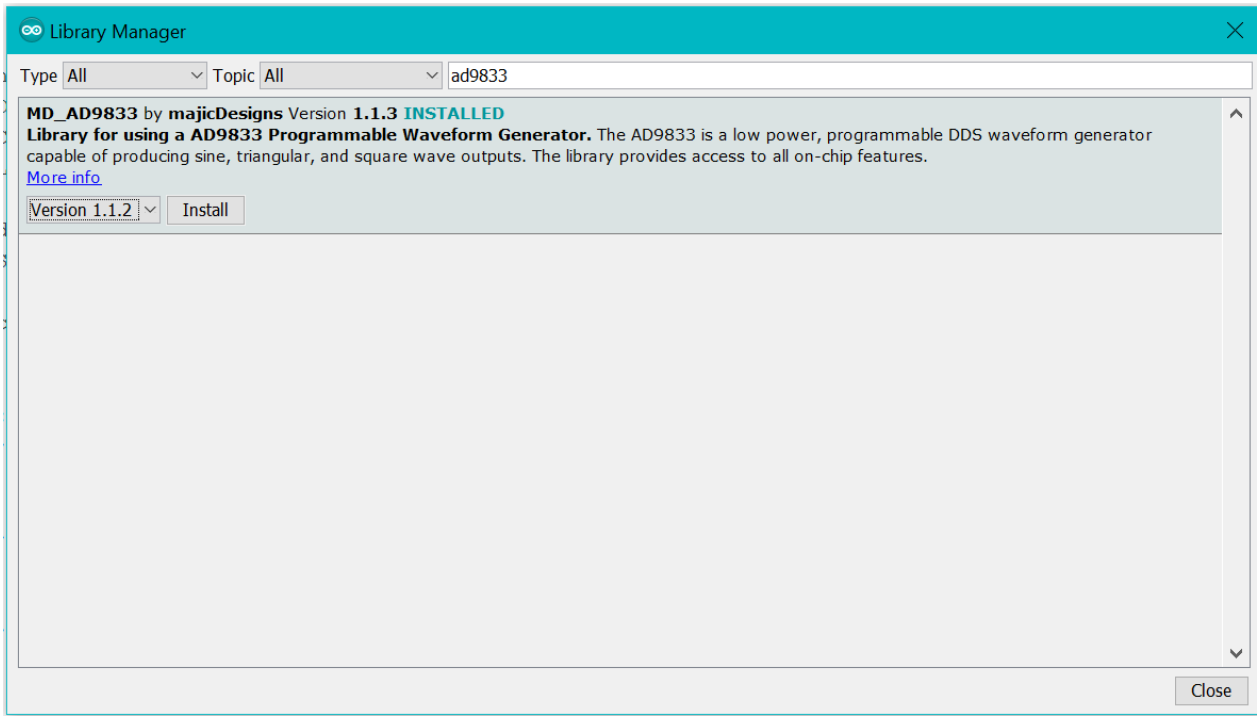
A PICTURE OF OUR CIRCUIT IN A BREADBOARD.

In the picture above, we used the signal from the function generator set at an amplitude of 650mV, offset of 500mV and a frequency of 100KHz, this mimics the AD9833. We used a function generator to test the opamp circuit at first before using the AD9833 chip. This was our output.



Signal one is our input from the function generator and signal two is the output from the opamp.

Then we ran a sample code from the Arduino library for AD9833 by MajicDesigns.



A SCREENSHOT OF THE LIBRARY AND HOW WE GOT IT FROM THE ARDUINO IDE

```
#include <SPI.h>
#include <MD_AD9833.h>

// Pins for SPI comm with the AD9833 IC
#define DATA  11        ///< SPI Data pin number
#define CLK   13        ///< SPI Clock pin number
#define FSYNC 10        ///< SPI Load pin number (FSYNC in AD9833 usage)

MD_AD9833       AD(FSYNC); // Hardware SPI
//MD_AD9833     AD(DATA, CLK, FSYNC); // Arbitrary SPI pins

// Character constants for commands
const char CMD_HELP = '?';
const char BLANK = ' ';
const char PACKET_START = ':';
const char PACKET_END = ';';
const char CMD_FREQ = 'F';
const char CMD_PHASE = 'P';
const char CMD_OUTPUT = 'O';
const char OPT_FREQ = 'F';
const char OPT_PHASE = 'P';
const char OPT_SIGNAL = 'S';
const char OPT_1 = '1';
const char OPT_2 = '2';
const char OPT_MODULATE = 'M';
const uint8_t PACKET_SIZE = 20;

void setup()
{
  Serial.begin(57600);
```

```
const uint8_t PACKET_SIZE = 20;

void setup()
{
  Serial.begin(57600);
  AD.begin();
  usage();
}

void usage(void)
{
  Serial.print(F("\n\n[MD_AD9833_Tester]"));
  Serial.print(F("\n?\thelp - this message"));
  Serial.print(F("\n\n:<cmd><opt> <param>;"));
  Serial.print(F("\n:f1n;\tset frequency 1 to n Hz"));
  Serial.print(F("\n:f2n;\tset frequency 2 to n Hz"));
  Serial.print(F("\n:fmn;\tset frequency modulation to n Hz"));
  Serial.print(F("\n:p1n;\tset phase 1 to n in tenths of a degree (1201 is 120.1 deg)"));
  Serial.print(F("\n:p2n;\tset phase 2 to n in tenths of a degree (1201 is 120.1 deg)"));
  Serial.print(F("\n:ofn;\toutput frequency n or modulation [n=1/2/m]"));
  Serial.print(F("\n:opn;\toutput phase n or modulation [n=1/2/m]"));
  Serial.print(F("\n:osn;\toutput signal type n [n=(o)ff/(s)ine/(t)riangle/s(q)uare]"));
}

uint8_t htoi(char c)
{
  c = toupper(c);

  if (c >= '0' && c <= '9')
    return(c - '0');
  else if (c >= 'A' && c <= 'F')
```

```
uint8_t htoi(char c)
{
  c = toupper(c);

  if (c >= '0' && c <= '9')
      return(c - '0');
  else if (c >= 'A' && c <= 'F')
      return(c - 'A' + 10);
  else
      return(0);
}

char nextChar(void)
// Read the next character from the serial input stream
// Blocking wait
{
  while (!Serial.available())
    ; /* do nothing */
  return(toupper(Serial.read()));
}

char *readPacket(void)
// read a packet and return the
{
  static enum { S_IDLE, S_READ_CMD, S_READ_MOD, S_READ_PKT } state = S_IDLE;
  static char cBuf[PACKET_SIZE + 1];
  static char *cp;
  char c;

  switch (state)
```

```
  {
  case S_IDLE:   // waiting for packet start
    c = nextChar();
    if (c == CMD_HELP)
    {
      usage();
      break;
    }
    if (c == PACKET_START)
    {
      cp = cBuf;
      state = S_READ_CMD;
    }
    break;

  case S_READ_CMD:   // waiting for command char
    c = nextChar();
    if (c == CMD_FREQ || c == CMD_PHASE || c == CMD_OUTPUT)
    {
      *cp++ = c;
      state = S_READ_MOD;
    }
    else
      state = S_IDLE;
    break;

  case S_READ_MOD: // Waiting for command modifier
    c = nextChar();
    if (c == OPT_FREQ || c == OPT_PHASE || c == OPT_SIGNAL ||
      c == OPT_1 || c == OPT_2 || c == OPT_MODULATE)
    {
```

```
      c == OPT_1 || c == OPT_2 || c == OPT_MODULATE)
    {
      *cp++ = c;
      state = S_READ_PKT;
    }
    else
      state = S_IDLE;
    break;

  case S_READ_PKT: // Reading parameter until packet end
    c = nextChar();
    if (c == PACKET_END)
    {
      *cp = '\0';
      state = S_IDLE;
      return(cBuf);
    }
    *cp++ = c;
    break;

  default:
    state = S_IDLE;
    break;
  }

  return(NULL);
}

void processPacket(char *cp)
// Assume we have a correctly formed packet from the pasing in readPacket()
{
```

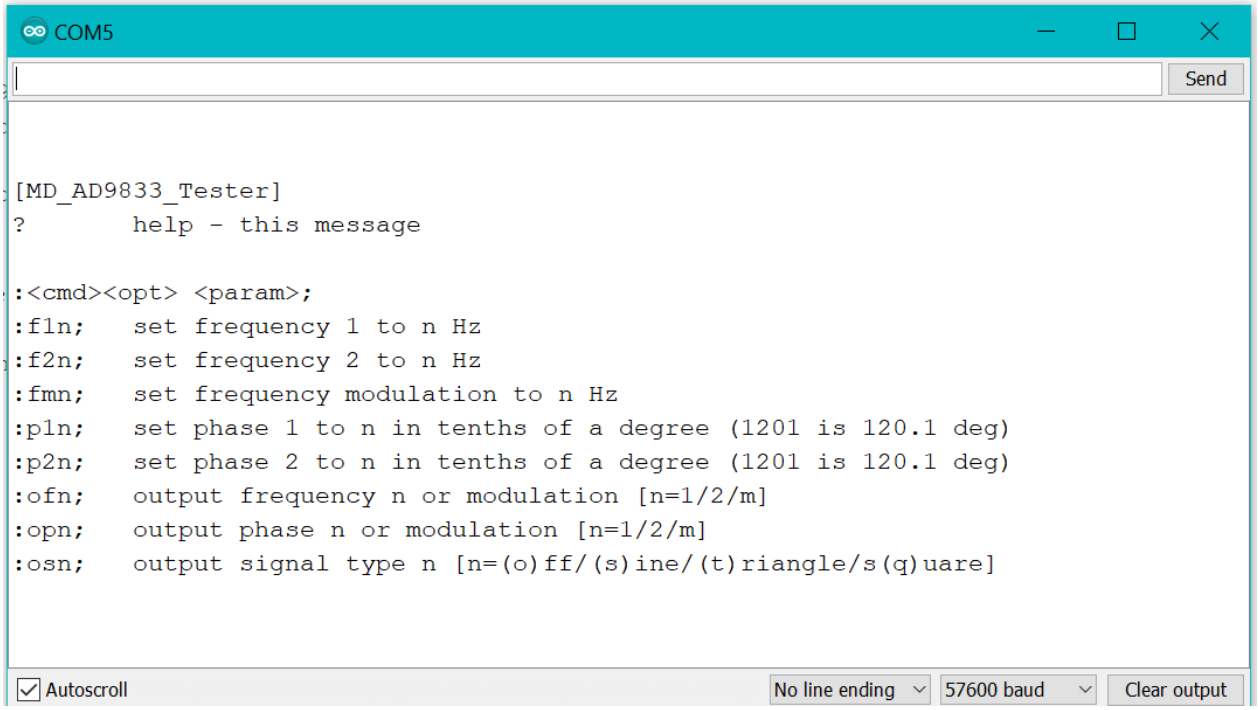
```
  uint32_t  ul;
  MD_AD9833::channel_t chan;
  MD_AD9833::mode_t mode;

  switch (*cp++)
  {
  case CMD_FREQ:
    switch (*cp++)
    {
    case OPT_1: chan = MD_AD9833::CHAN_0; break;
    case OPT_2: chan = MD_AD9833::CHAN_1; break;
    case OPT_MODULATE: /* do something in future */ break;
    }

    ul = strtoul(cp, NULL, 10);
    AD.setFrequency(chan, ul);
    break;

  case CMD_PHASE:
    switch (*cp++)
    {
    case OPT_1: chan = MD_AD9833::CHAN_0; break;
    case OPT_2: chan = MD_AD9833::CHAN_1; break;
    }

    ul = strtoul(cp, NULL, 10);
    AD.setPhase(chan, (uint16_t)ul);
    break;

  case CMD_OUTPUT:
    switch (*cp++)
```

```
    switch (*cp++)
    {
    case OPT_FREQ:
      switch (*cp)
      {
      case OPT_1: chan = MD_AD9833::CHAN_0; break;
      case OPT_2: chan = MD_AD9833::CHAN_1; break;
      case OPT_MODULATE: /* do something in future */ break;
      }
      AD.setActiveFrequency(chan);
      break;

    case OPT_PHASE:
      switch (*cp)
      {
      case OPT_1: chan = MD_AD9833::CHAN_0; break;
      case OPT_2: chan = MD_AD9833::CHAN_1; break;
      case OPT_MODULATE: /* do something in future */ break;
      }
      AD.setActivePhase(chan);
      break;

    case OPT_SIGNAL:
      switch (*cp)
      {
      case 'O': mode = MD_AD9833::MODE_OFF;    break;
      case 'S': mode = MD_AD9833::MODE_SINE;   break;
      case 'T': mode = MD_AD9833::MODE_TRIANGLE;  break;
      case 'Q': mode = MD_AD9833::MODE_SQUARE1;  break;
      }
      AD.setMode(mode);
```

```
    case OPT_PHASE:
      switch (*cp)
      {
      case OPT_1: chan = MD_AD9833::CHAN_0; break;
      case OPT_2: chan = MD_AD9833::CHAN_1; break;
      case OPT_MODULATE: /* do something in future */ break;
      }
      AD.setActivePhase(chan);
      break;

    case OPT_SIGNAL:
      switch (*cp)
      {
      case 'O': mode = MD_AD9833::MODE_OFF;    break;
      case 'S': mode = MD_AD9833::MODE_SINE;   break;
      case 'T': mode = MD_AD9833::MODE_TRIANGLE;  break;
      case 'Q': mode = MD_AD9833::MODE_SQUARE1;  break;
      }
      AD.setMode(mode);
      break;
    }
    break;
  }

  return;
}


void loop()
{
  char  *cp;

  if ((cp = readPacket()) != NULL)
    processPacket(cp);
}
```

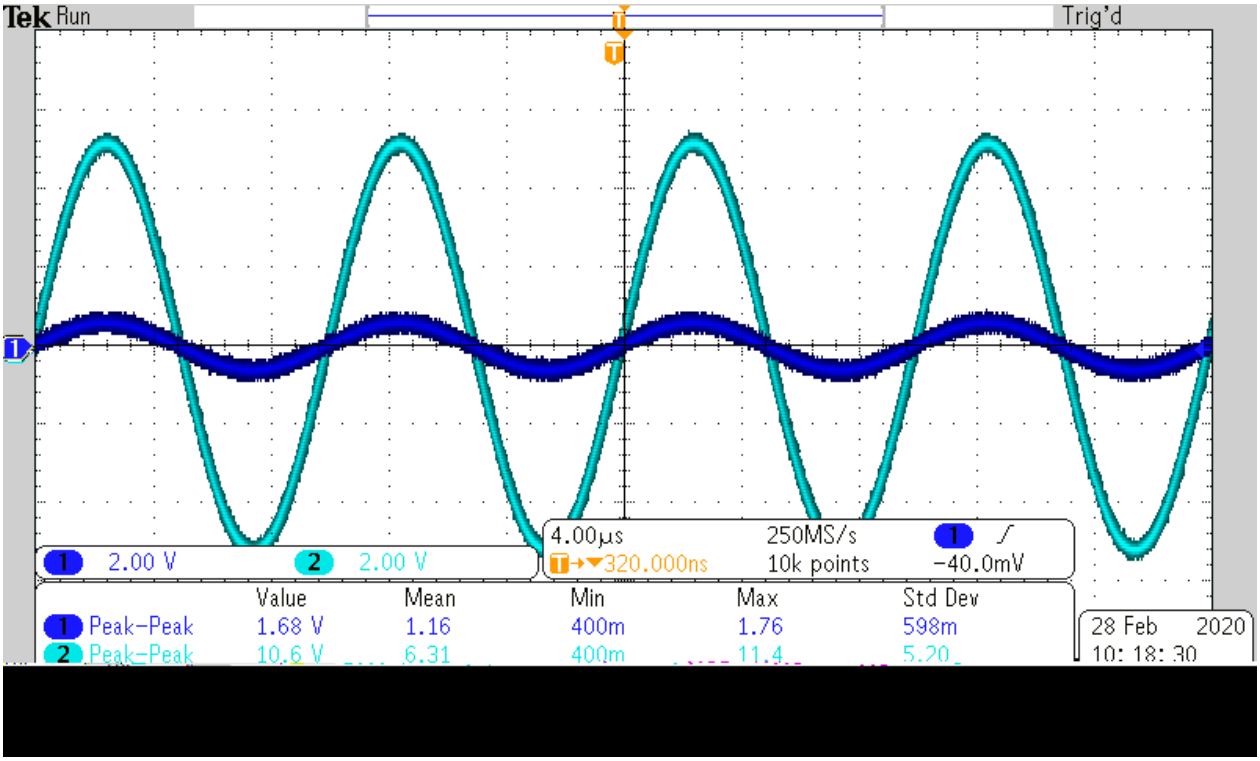
THIS IS THE CODE WE GOT FROM THE LIBRARY BY MAJICDESIGNS.

When we uploaded this to the Arduino and ran it into the AD9833 module, the menu was shown on the serial monitor of the Arduino IDE.



THE SERIAL MONITOR OF THE ARDUINO IDE AFTER UPLOADING THE CODE.

After following the instructions on the interface and getting a good output we got this output:



After getting this output, we were comfortable with the circuit we designed, so we started making the schematic and designing the PCB.

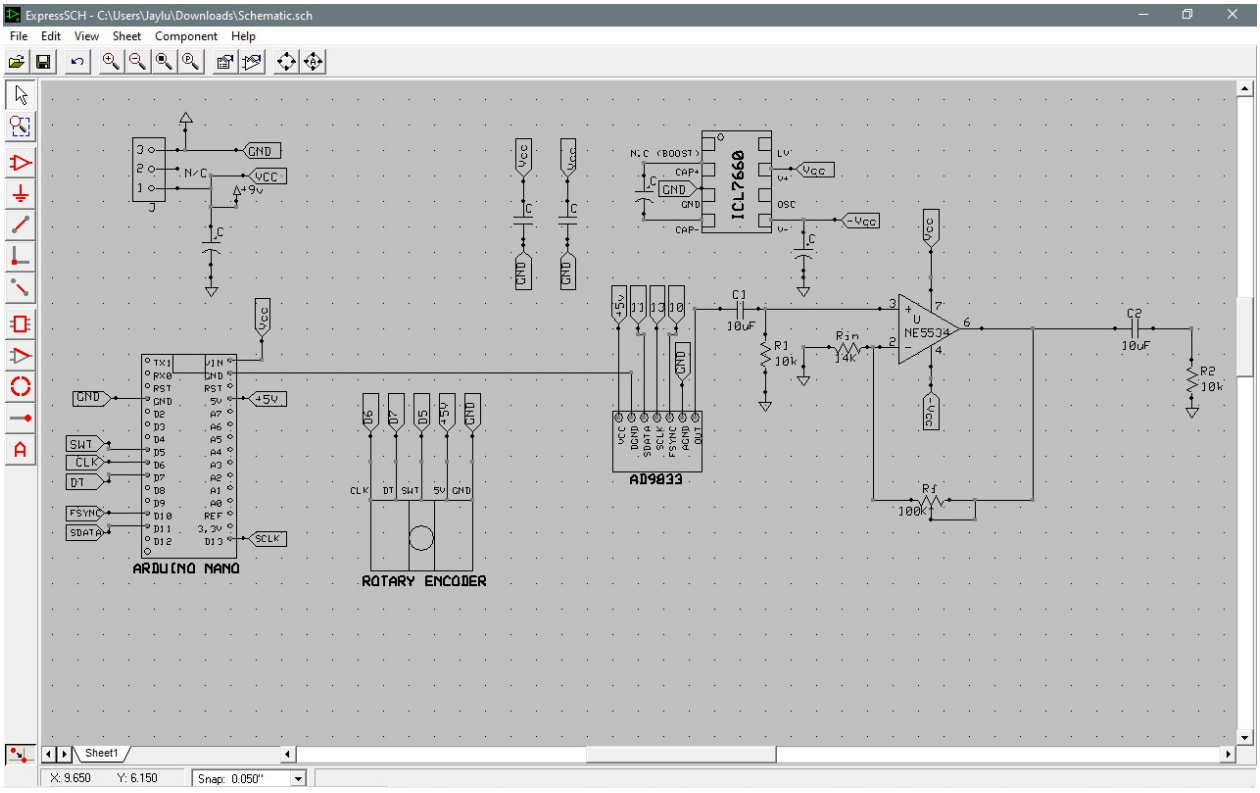
Active Amplifier Circuit

This seemed like a very straight forward circuit, but we probably spent the most time on this! We were able to eventually get the wave we wanted, but we ran into several problems first. The op amp would clip the voltage before it got to the +5V. So, it looked like we had a voltage from -5V to +4V clipped like +4V was the rail. We also had troubles getting the circuit to perfectly offset around 0 as well.

After rebuilding the circuit numerous times and tweaking and replacing parts we saw what we wanted! From an input of 0.65V peak and an offset of 0.5V we were able to output a sine wave with a voltage of 5V peak with 0 offset using our active and passive RC filter!

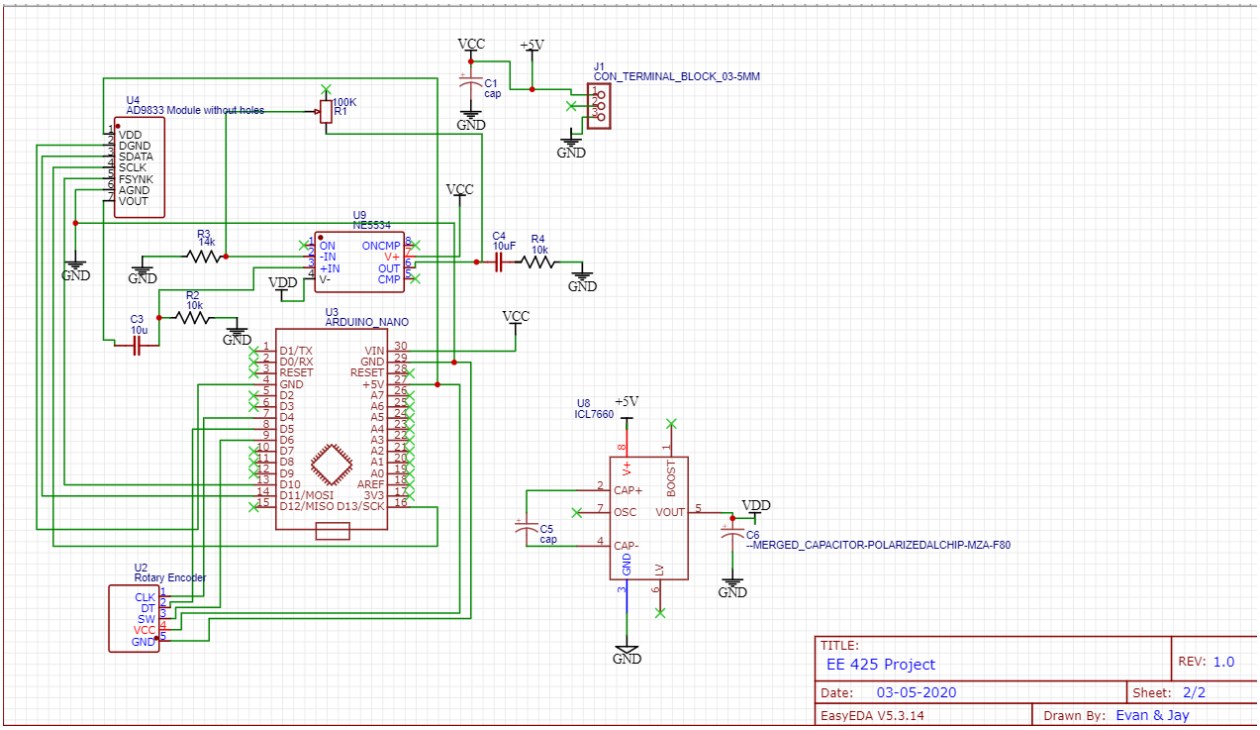
DESIGNING THE PCB (PRINTED CIRCUIT BOARD)

To start designing the schematic, we started off with express PCB and the two-designing software, ExpressSCH to design the schematic and ExpressPCB to design the PCB. One trouble that we ran into was that the schematic software didn't have a wide range of offline and online library, so we had to often build our own components using shapes. This method worked for the schematic but wasn't accurate. As you can see in the image below, the schematic had more hand build shapes rather than library components.

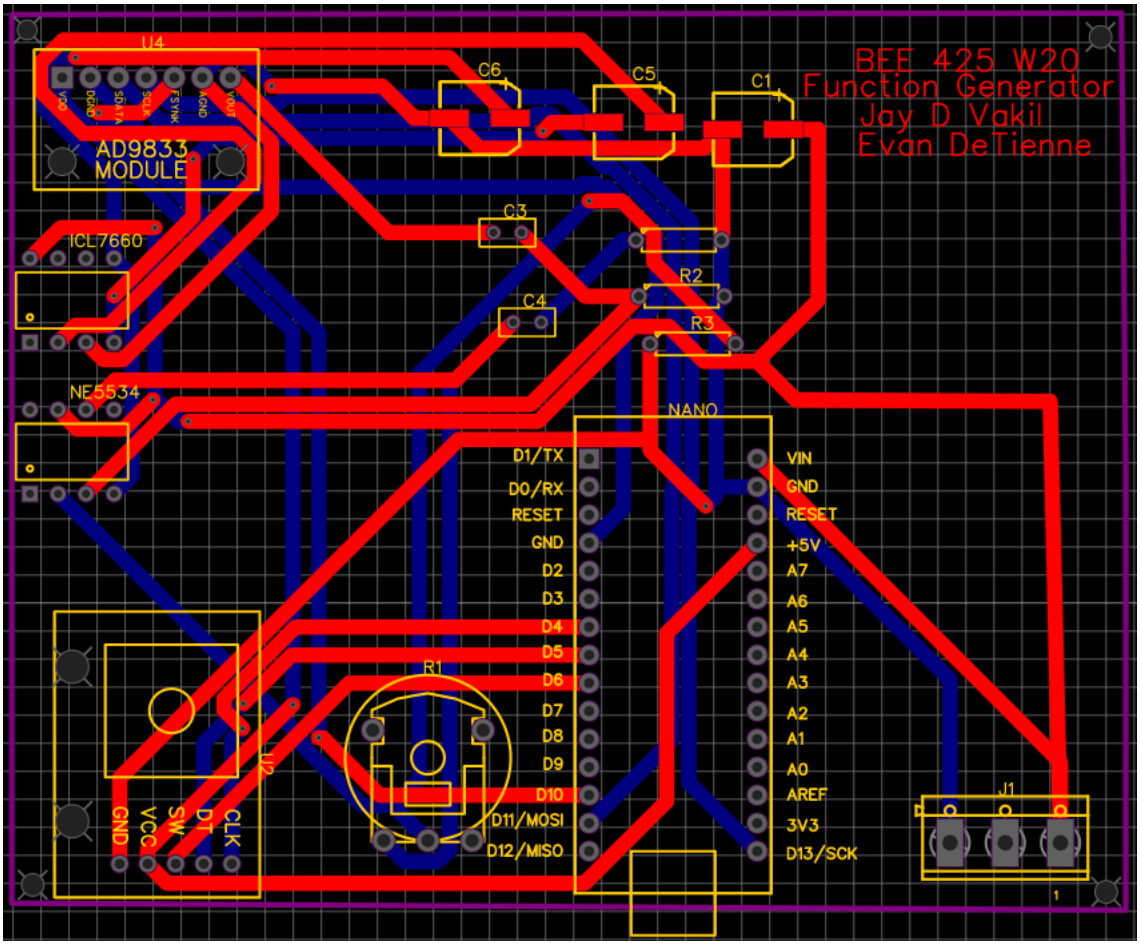


OUR SCHEMATIC MADE USING EXPRESSSCH.

Also, ExpressPCB offered PCBs' at a very expensive rate (ExpressPCB), so we chose JLCPCB and its online schematic and pcb designing software easyEDA.com to design our schematic. Our schematic looked like this:



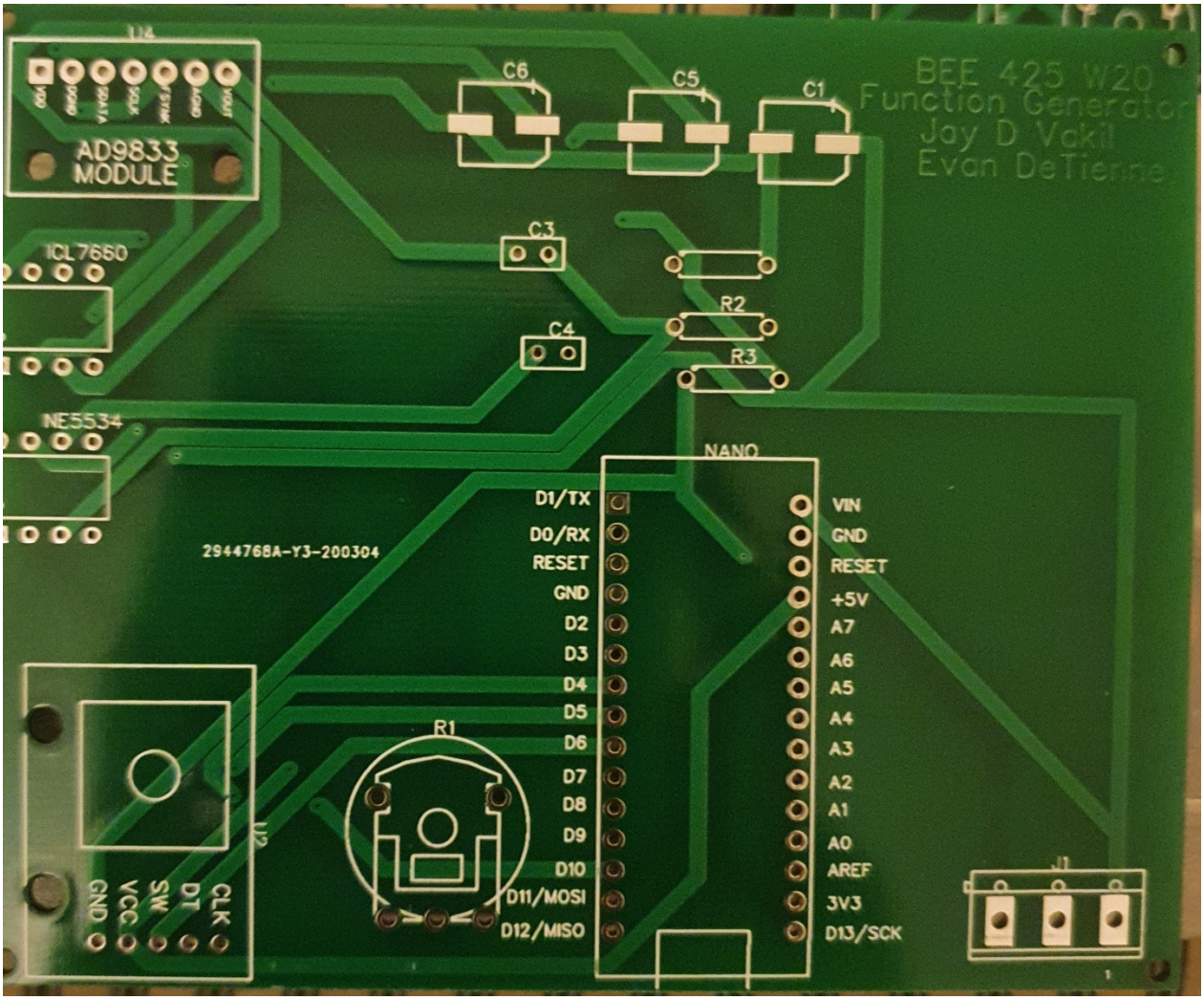
One good thing about easyEDA is that it allows us to export the schematic into a PCB and autotrace the traces on the PCB. So, using the feature, we made our PCB.



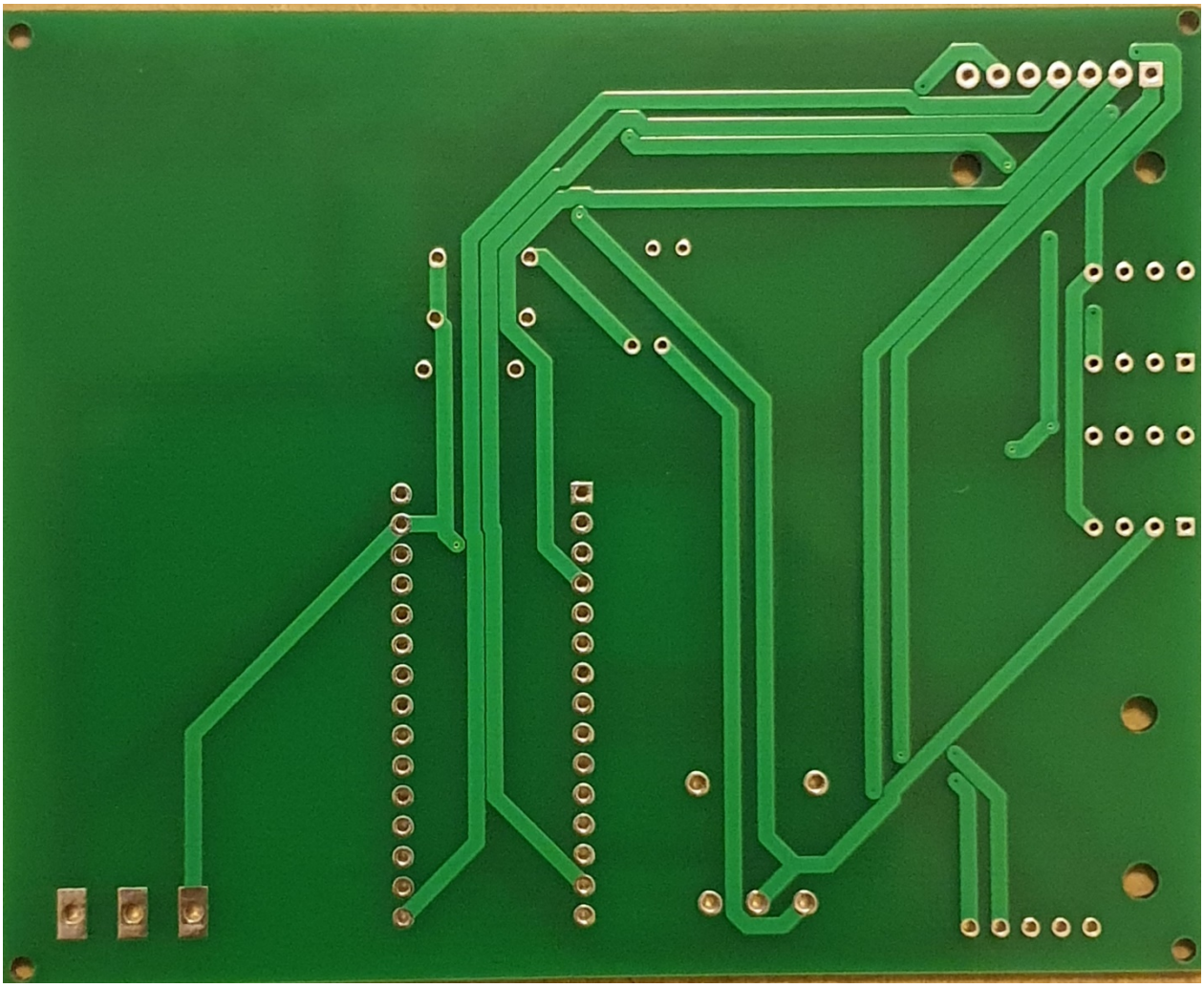
After getting the schematic and PCB, we asked Mohmmadreza Esmaeillou and Yahya Farah to have a design review session so we could confirm if our design is correct. After we had a design review and after making the viable changes recommended by the group, we ordered the PCB from JLCPCB in China. It arrived after a week.

ASSEMBLY OF THE PCB

After we got the pcb, which looked like this:

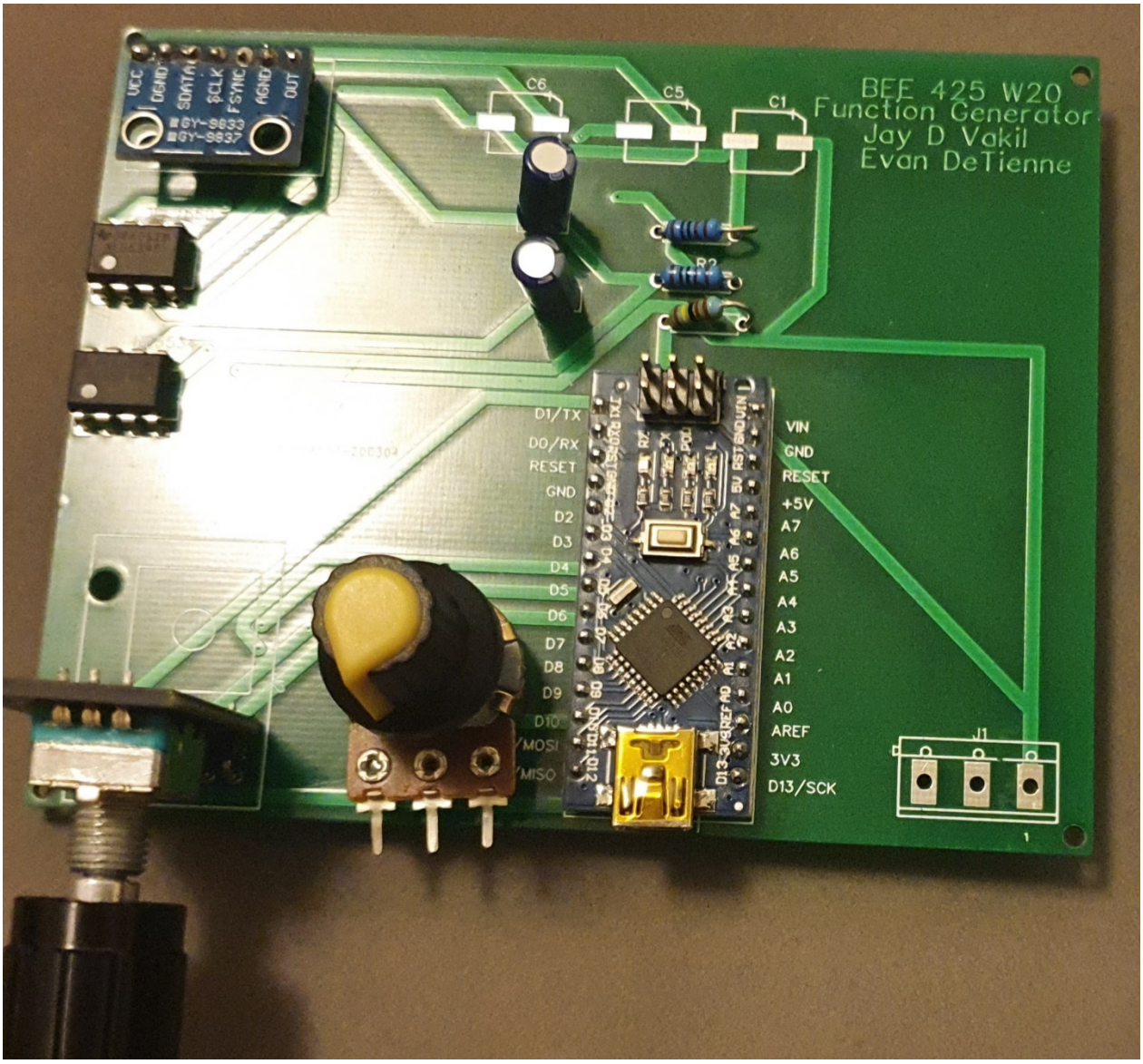


THE FRONT VIEW OF THE 2-LAYER PCB



THE BACK SIDE OF THE 2-LAYER PCB

After we got the pcb and checked for quality, which it passed, we referred to the schematic to see what component goes in which slot, after completing the matching we got most of the PCB ready for the use, but we couldn't solder parts or have other parts because of Covid-19 and the cancellation of college.



THIS IS OUR FINAL ASSEMBLED PCB.

We couldn't connect our potentiometer because the hole size was wrong, and it didn't fit. C6, C5, and C1 capacitors could be placed because it required soldering which we couldn't do because of college closure. We had ordered BNC holders and a connector for the power supply, but we cancelled the order because of shipment delay and Covid-19.

AD9833 CODE

AD9833

```
#include <SPI.h>
#include <MD_AD9833.h>

// Pins for SPI comm with the AD9833 IC
#define DATA  11  ///< SPI Data pin number
#define CLK   13  ///< SPI Clock pin number
#define FSYNC 10  ///< SPI Load pin number (FSYNC in AD9833 usage)
#define REclk 4
#define REdt  5
#define REswitch 6
int clkState, DTstate, counter = 0, modeSelect = 0;

MD_AD9833 AD(FSYNC); // Hardware SPI
//MD_AD9833 AD(DATA, CLK, FSYNC); // Arbitrary SPI pins

// Character constants for commands
const char CMD_HELP = '?';
const char BLANK = ' ';
const char PACKET_START = ':';
const char PACKET_END = ';';
const char CMD_FREQ = 'F';
const char CMD_PHASE = 'P';
const char CMD_OUTPUT = 'O';
const char OPT_FREQ = 'F';
const char OPT_PHASE = 'P';
const char OPT_SIGNAL = 'S';
const char OPT_1 = '1';
const char OPT_2 = '2';
const char OPT_MODULATE = 'M';
const uint8_t PACKET_SIZE = 20;

void setup()
{
  pinMode(REclk, INPUT);
  pinMode(REdt, INPUT);
  pinMode(REswitch, INPUT);
  DTstate = digitalRead(REclk);
  Serial.begin(57600);
  AD.begin();
}
```

```
}

void loop()
{
  clkState = digitalRead(REclk);
  if(clkState != DTstate){
    if(digitalRead(REdt)!=clkState){
      counter++;
    }else{
      counter--;
    }
  }

  char  *cp;

  if ((cp = readPacket()) != NULL)
    processPacket(cp);
  clkState = DTstate;
}



uint8_t htoi(char c)
{
  c = toupper(c);

  if (c >= '0' && c <= '9')
      return(c - '0');
  else if (c >= 'A' && c <= 'F')
      return(c - 'A' + 10);
  else
      return(0);
}

char nextChar(void)
// Read the next character from the serial input stream
// Blocking wait
{
  while (!Serial.available())
    ; /* do nothing */
  return(toupper(Serial.read()));
}



char *readPacket(void)
// read a packet and return the
{
  static enum { S_IDLE, S_READ_CMD, S_READ_MOD, S_READ_PKT } state = S_IDLE;
  static char cBuf[PACKET_SIZE + 1];
  static char *cp;
  char c;

  switch (state)
  {
  case S_IDLE:   // waiting for packet start
    c = nextChar();
    if (c == CMD_HELP)
    {
      break;
    }
    if (c == PACKET_START)
    {
      cp = cBuf;
      state = S_READ_CMD;
    }
    break;

  case S_READ_CMD:   // waiting for command char
    c = nextChar();
    if (c == CMD_FREQ || c == CMD_PHASE || c == CMD_OUTPUT)
    {
      *cp++ = c;
      state = S_READ_MOD;
    }
    else
      state = S_IDLE;
    break;

  case S_READ_MOD: // Waiting for command modifier
    c = nextChar();
    if (c == OPT_FREQ || c == OPT_PHASE || c == OPT_SIGNAL ||
      c == OPT_1 || c == OPT_2 || c == OPT_MODULATE)
    {
      *cp++ = c;
      state = S_READ_PKT;
```

```
    }
    else
      state = S_IDLE;
    break;

  case S_READ_PKT: // Reading parameter until packet end
    c = nextChar();
    if (c == PACKET_END)
    {
      *cp = '\0';
      state = S_IDLE;
      return(cBuf);
    }
    *cp++ = c;
    break;

  default:
    state = S_IDLE;
    break;
  }

  return(NULL);
}

void processPacket(char *cp)
// Assume we have a correctly formed packet from the pasing in readPacket()
{
  uint32_t  ul;
  MD_AD9833::channel_t chan;
  MD_AD9833::mode_t mode;

  switch (*cp++)
  {
  case CMD_FREQ:
    switch (*cp++)
    {
    case OPT_1: chan = MD_AD9833::CHAN_0; break;
    case OPT_MODULATE: /* do something in future */ break;
    }
    AD.setFrequency(chan, counter);
    break;
```

```
  case CMD_PHASE:
    switch (*cp++)
    {
    case OPT_1: chan = MD_AD9833::CHAN_0; break;
    }

    ul = strtoul(cp, NULL, 10);
    AD.setPhase(chan, (uint16_t)ul);
    break;

  case CMD_OUTPUT:
    switch (*cp++)
    {
    case OPT_FREQ:
      switch (*cp)
      {
      case OPT_1: chan = MD_AD9833::CHAN_0; break;
      case OPT_MODULATE: /* do something in future */ break;
      }
      AD.setActiveFrequency(chan);
      break;

    case OPT_PHASE:
      switch (*cp)
      {
      case OPT_1: chan = MD_AD9833::CHAN_0; break;
      case OPT_MODULATE: /* do something in future */ break;
      }
      AD.setActivePhase(chan);
      break;

      if(digitalRead(REswitch)==HIGH){
        modeSelect++;
      }
    case OPT_SIGNAL:
      switch (modeSelect)
      {
      case 0: mode = MD_AD9833::MODE_OFF;    break;
      case 1: mode = MD_AD9833::MODE_SINE;   break;
      case 2: mode = MD_AD9833::MODE_TRIANGLE;  break;
      case 3: mode = MD_AD9833::MODE_SQUARE1;  break;
      }
```

```
    }
  case OPT_SIGNAL:
    switch (modeSelect)
    {
    case 0: mode = MD_AD9833::MODE_OFF;    break;
    case 1: mode = MD_AD9833::MODE_SINE;   break;
    case 2: mode = MD_AD9833::MODE_TRIANGLE;  break;
    case 3: mode = MD_AD9833::MODE_SQUARE1;  break;
    }
    AD.setMode(mode);
    break;
  }
  break;
 }
 //if modeSelect is greater than 3 it resets.
 if(modeSelect > 3){
   modeSelect = 0;
 }

 return;
}
```

THIS IS OUR MODIFIED CODE FROM THE AD9833 CODE BY MAJICDESIGNS.

We had to use the code by MajicDesigns because we didn't know how the AD9833 is programmed and we didn't have enough time to get the hang of it. The code compiles properly but we had no means to test if it worked the way we wanted it to work.

CONCLUSION

In conclusion, we thought this was a great EE project! We had to use a lot of our previous EE classes and learned how to build a PCB from scratch. Troubleshooting was a bit of a pain but was still a great learning experience. We wish the Coronavirus didn't hit so we could've soldered all our components together and got a working demo together!

We think we could've gotten the whole thing to work if we could've gone to the school. The code was the most difficult part for us. We ran into issues and never resolved them, but we got close. The code we wrote and worked with is below in the appendices.

We got essentially everything else to work individually on the breadboard. We ordered all the parts and had them come in just in time for the school to close.

THOUGHTS

If we had to do it all over again, I think we would've used op amps. We used op amps in our B EE 433 electronic circuit design class to build a sine wave that operated at 10KHz. It wasn't too bad. Then from there making a triangle wave and square wave wouldn't have been too bad. Then we just would have to add rotary encoders and call it good.

ERRORS

Whilst testing our circuit we ran into some issues with the AD9833 module but Will Phang from our class B EE 425 willingly helped us out thoroughly in a very altruistic way. At first, we tried to use a NE5532 opamp, but it fried in the testing stage and the lab ran out of the NE5532 opamps, so we moved to NE5534 and it worked out for us. The biggest issue we faced in the pcb assembly was that we got a smaller hole size for the pcb and our potentiometer was of a bigger size. We would have used jumper wires to use the potentiometer hot glued to the chassis.

BIBLIOGRAPHY

"AD9833." *AD9833 Datasheet and Product Info | Analog Devices*, www.analog.com/en/products/ad9833.html#product-overview. Date accessed 3/19/20

"Arduino Nano". Arduino store, https://store.arduino.cc/usa/arduino-nano. Date accessed 3/19/20

"PCB CAD Software." *ExpressPCB*, www.expresspcb.com/express-pcb-manufacturing-service/.

"Switched-Capacitor Voltage Converters". Maxim Integrated. Date accessed 3/19/20

https://www.maximintegrated.com/en/products/power/charge-pumps/ICL7660.html. Date accessed 3/19/20

Dejan, et al. "How Rotary Encoder Works and How To Use It with Arduino." *HowToMechatronics*, 15 Dec. 2019, https://www.howtomechatronics.com/tutorials/arduino/rotary-encoder-works-use-arduino/. Date accessed 3/19/20

MajicDesigns. "MajicDesigns/MD_AD9833." *GitHub*, https://www.github.com/MajicDesigns/MD_AD9833/blob/master/examples/MD_AD9833_Test/MD_AD9833_Test.ino. Date accessed 3/19/20