CymSTAR Ethernet-to-Synchro Project

ECEN 4024

Dr. John Ohara

Professor Nate Lannan

5 May 2023

Name	**Role**
Christian Moser	Code Architect and Scrum Master
Dylan Gore	Hardware Manager and Point of Contact
Remington Ward	Software Engineer and Scrum Master
Justin Brown	Hardware Engineer and Scribe

Contents

1 Project Description - Remington and Justin

Research available technologies and design a hardware, software product that enables a host computer to drive a synchro receiver, a gauge or needle in a plane cockpit using a synchro transmitter that is controlled over an ethernet protocol. This is to help military pilots get a more realistic feel in a simulated aircraft when they turn a dial or knob. We need the host computer to be able to transfer/receive through a RJ45 Ethernet Cable. The transferred information will be to tell the receiver Synchro to turn a dial or a knob to an accurate told position. The receiving information will be to tell the host computer that the receiver Synchro has turned to the correct position. Shown in the figure below is the full system, and everything in the blue box is what we are developing, the Transceiver Synchro.

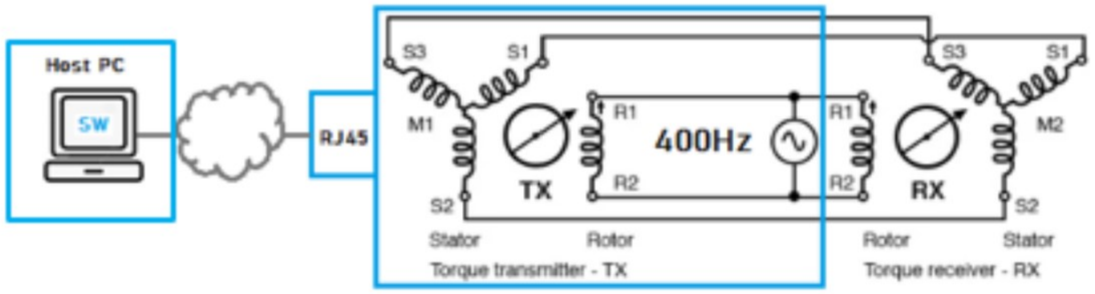


Figure 1: Everything in the blue box is what is being developed with a Beaglebone Black (Host PC) and hardware found to mimic a Transceiver Synchro.

CymStar wants us to be able to make two to four devices with the budget they gave us ($2000). All the research we did for available on the market Synchros we found that the prices range from $2,500-$5,000 dollars. So our personal objective is to be able to make four devices with the budget and have left over money in the end.

The system should be:

- Reliable, less than 10 pounds, and be no more than the size as a power brick (10*5*3 inches)
- Cheap and easy to produce
- Able to work for years even after software or firmware updates happen with no issues

2 Team Structure - All

Each member worked within their assigned tasks.

2.1 Technical Contributions

2.1.1 Christian Moser

Christian worked on Software Planning, GAANT Planning, UX Development, Software Development, Firmware Development, Networking Configuration

2.1.2 Dylan Gore

Dylan worked on Hardware Planning, Hardware Schematic Design and Testing, PCB Layout and Routing, Researching Designs, and Testing Synchros.

2.1.3 Remington Ward

Remington worked on Software Planning, SPI Pin Configurations, SPI Code Architect, Researched shifting signals with an RC circuit, Testing Synchros and recording produced signals.

2.1.4 Justin Brown

Justin worked on Hardware Block Diagram, PCB routing, Meet the team, Gantt Chart, Risk Management.

3 Ethical & Professional Considerations - Christian

3.1 Environmental Constraints

If incorrectly built, an electrical shock could cause health concerns. The device if built wrong or handled wrong could shock if not powered/unplugged correctly and if the wires are too thin they could overheat from too much current and melt the shielding causing a fire. The device will use electricity resources to be able to function and run.

3.2 Performance Constraints

If the device is built wrong it will make the simulations not as realistic resulting in bad training for the pilot and putting his/her life more at risk due to the device not being a 1/1 from simulation to reality.

3.3 Regulations

- IPC-2221 - Circuit Board Design Standard
- IEEE/ISO/IEC 26514-2021 - Software Development
- ISO 29.020 - Electrical Safety MIL-HDBK-225A - Military standard for synchros

3.4 Industry Standards

- IEEE 802.3 - Ethernet Cable Protocol: RJ45 Standard
- IEEE Code of Ethics
- IEEE 295-1969 - Power Transformer Standard
- IPC-2221 - Circuit Board Design Standard
- IEEE/ISO/IEC 26514-2021 - Software Development

4 Design Constraints - Remington

- $2000 budget for 2 - 4 identical devices
- Device must work with 26V and 400Hz
- Device must be about the size of a power brick (2U Height Maximum).
- Weight: 10 pounds or less
- Dimensions: 10 * 5 * 3 inches or less
- The device should be able to input ethernet packets from Host computer and output the electrical signals to drive the Synchro Receiver.
- The user should be able to input a position to drive the Synchro and the ethernet packets should be sent to the Synchro device.

5 Hardware Implementation - Dylan and Justin

5.1 Design

5.1.1 Overview

Since we're only needing to receive data over the Ethernet protocol and act in place of a transmitter synchro, we're able to use a simple cape PCB on top of an Arduino Uno Rev3. The Arduino does not have an Ethernet port, but it has enough pins for digital pins, serial communication and power. By using the Arduino we're able to use the built in libraries to handle the RJ45 protocol. The Ethernet data can then be decoded to extrapolate the desired synchro angle, then SPI control signals can modulate our PCB cape to output the correct signals.

The first issue we came across when designing the circuit was the power constraint. While the synchro's in this application will pull low currents, the sine voltage peak required will be over 30V. To handle this we decided to use an external power supply that can supply high voltage, rail to rail op-amps with +/-40V to source the high voltage required by the synchros.

The next issue we came across is the requirement for the output voltage amplitudes to vary from 0V to the required 11.8Vrms. Each output will be phase aligned with each other and the rotor but the amplitudes will vary according to the desired angle. The equations that dictate these amplitudes will be listed below.

$$\begin{aligned} V_{S1} &= 16.68755\cos(\theta)\cdot\sin(\omega t) \\ V_{S2} &= 16.68755\cos(\theta+120^\circ)\cdot\sin(\omega t) \\ V_{S1} &= 16.68755\cos(\theta+240^\circ)\cdot\sin(\omega t) \end{aligned}$$

Our method for the variable output voltages is to use digital potentiometers in a voltage divider setup. These digital potentiometers can vary from 100 ohms to 10K ohms and can be adjusted using an SPI protocol built into the devices. We chose to use two digital potentiometers so we would be able to get more precision as well as get as close to 0V as possible. All 6 of the digital potentiometers can be written to every 36 microseconds, meaning we can change each of their values 69 times before the 400Hz reference voltage makes a complete cycle. The digital potentiometers we chose each have 1024 steps in them which allows an accuracy of 9 ohms per step. This will give us an angle accuracy of 0.03065°.

To make an accurate reference voltage we decided to use digital synthesis chips, which can output a sine wave with a controllable phase to account for op-amp phase drift. One caveat to the digital synthesis IC is the DC voltage bias on the output. The digital synthesis IC can't output a negative voltage so it is DC shifted up by 0.36V. To account for this we use a differential amplifier circuit to remove the DC bias.

Since there are various different sub circuits one concern was the different voltages each circuit needed. To fix this we have two circuits that can output both +/-5V as well as +/-2.5V for power. The 2.5V is important because that's the maximum voltage the digital potentiometers can drop across their terminals. Due to this constraint we have an extra stage in our circuit to vary the amplitude using a low voltage op-amp.

The final constraint in our hardware design was making sure the signals are all phase aligned on the output with the rotor. A non-inverting amplifier circuit's output can't bring the output voltage lower than the input voltage's amplitude due to the "1+" in the output equation. To get around this we used an inverting amplifier to allow for a higher voltage swing output. However, by making these signals negative, they will be phase shifted 180° to the rotor. To fix this we use an inverting amplifier circuit for the amplification of the rotor circuit, whereas the rest use a non-inverting. This will ensure the output voltages are all phase aligned to each other.

5.2 Critical Sections

5.2.1 Digital Synthesis

The first issue we needed to tackle was generating a reliable sine wave, preamplification, that we could vary the phase of. We needed to be able to vary the phase just in case our subsequent circuit's non-idealities pulled each signal out of phase with each other.

We tested two methods but ultimately we decided the best solution was to use a digital synthesis IC for each of the 4 desired signals. While you could technically use one IC, we wanted to use 4 separate IC's to adjust for the non idealities mentioned previously.

The device we chose, the AD9833, can output a variety of waves including a sine wave. Since aircraft operate at 400 Hz we want to generate a 400 Hz sine wave from each of the IC's. The IC's use the SPI serial communication protocol which we will control via the Arduino Uno. The AD9833 also has registers to allow you to control various different aspects of the output wave which makes it the ideal choice for this project.

The output of this device is a 400 Hz sine wave with 0.72Vpp, however it has a DC offset of 0.36V. To combat this we added a capacitor to the output of the device which will filter the DC component out of the sine wave.

Another interesting issue we came across is the MCLK signal on the device. This signal needs to be an accurate square wave ranging up to 25 MHz, with 25 MHz being the default for the frequency calculations. When testing, our function generator couldn't generate an accurate square wave with such high frequency which caused some issues on the output. To combat this we were able to adjust our register value according to the equation below using a smaller reference frequency.

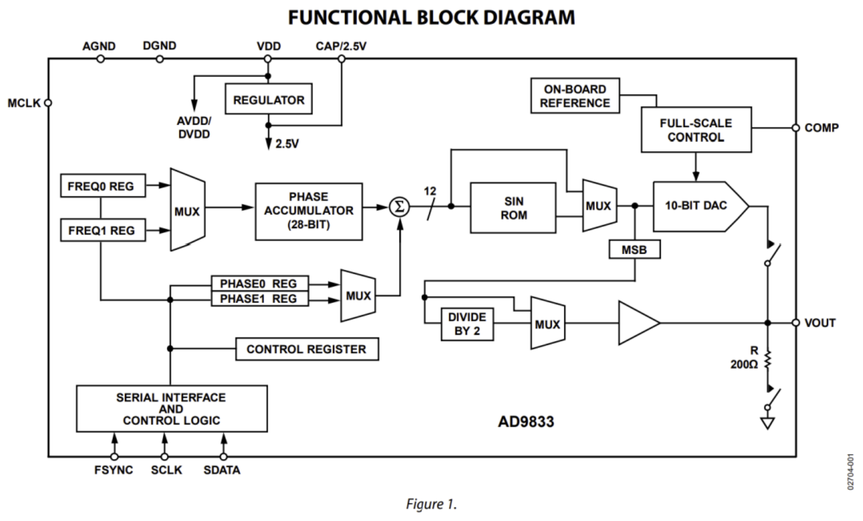


Figure 2: Digital Synthesis Internal

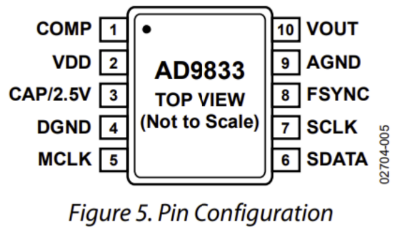


Figure 3: Digital Synthesis IC Pinout

$$Output_{Frequency} = (F_{MCLK}/2^{28}) * F_{Reg}$$

The RJ45 ethernet to Arduino adapter outputs a stable MCLK signal so we are using that as the MCLK input instead of a PWM signal from the Arduino.

5.2.2 Voltage Regulation

The first stage Op-Amps require a +/-5V supply, however the Arduino only provides us with +5V. To fix this we needed a solution to transform the 5V to -5V with a common ground between them.

To make the device as simple as possible we found a cheap IC that can give us a stable -5V and a command ground to provide to the Op-Amps. All we needed was the IC and two decoupling capacitors to get a steady +/-5V supply.

The digital synthesis on the other hand requires +/-2.5 instead of the +/-5 we've been using. Since we don't already have a 2.5V supply rail but we do have a 5V rail we had to get creative.

We used an Op-Amp, which takes in 2.5V using a voltage divider and acts as a buffer. The output of this buffer goes to an NPN/PNP circuit which gives us a common ground. So from the 5V rail to the common ground is +2.5V and from the common ground to the actual ground is -2.5V.

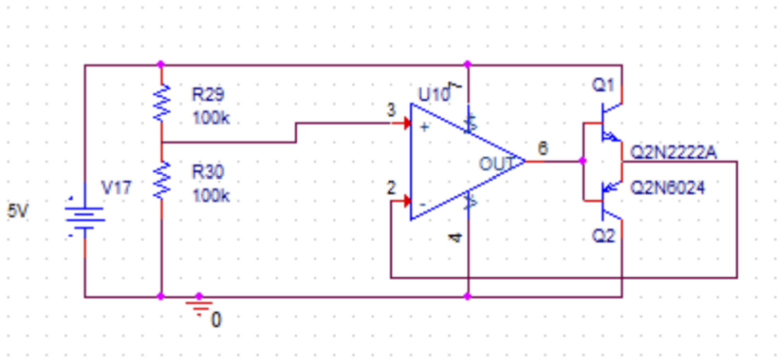


Figure 4: 2.5V Regulator Design

5.2.3 Digital Amplitude Variation

The main problem we had when designing this device was how to vary the voltage of each of the stators independently of each other. After exploring various techniques with MOSFET's, we decided on digital potentiometers.

Digital potentiometers, similar to physical ones, can be controlled to vary the resistance between the low and the wiper pin. Unlike traditional potentiometers, with digital ones you can vary the resistance using a serial communication protocol, in our case SPI.

While there are variations of the part we got, make sure the part being tested with is linear as some of the variations aren't meant to be used as a variable potentiometer.

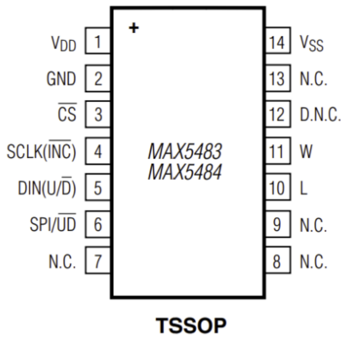


Figure 8: Digital Potentiometer Pinout (4)

Figure 5: Digital Potentiometer Pinout

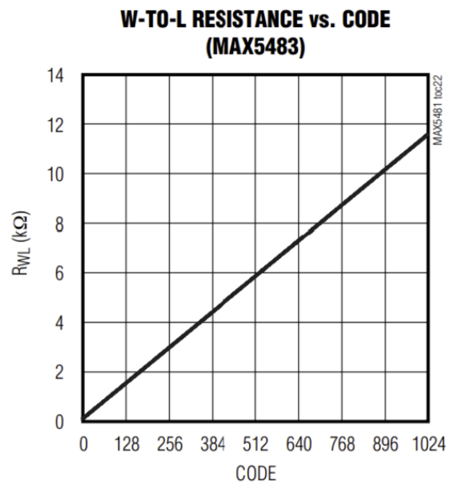


Figure 6: Ideal Linear Resistance Potentiometer Curve

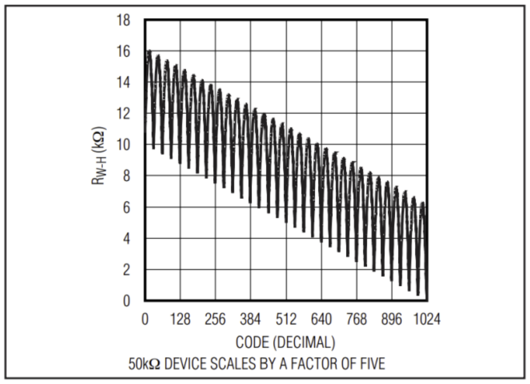


Figure 1. Resistance from W to H vs. Code (10kΩ Voltage-Divider)

Figure 7: Non-Ideal Non-Linear Resistance Potentiometer Curve

Voltage Step per Degree:

$$16.68755cos(1) - 16.6875cos(2) = 0.00762V$$

Step Size:

$$10,000/1024 = 9.765\Omega$$

Inverting Op-Amp:

$$(-100/10,000) + (109,765/10,000) = 0.0009\Omega$$

As mentioned previously these potentiometers can only have +/-2.5V across them so an extra stage was needed to be added in order to accommodate them.

To give us the highest possible resolution we decided on digital potentiometers with 1024 taps in each of them. This means there are 1024 different resistances that can be chosen from which gives us high accuracy.

The larger the resistance limit the closer to zero we could get, however the less accurate it will be. To balance these trade-offs we decided on a 10K potentiometers with 1024 steps each. This will give us a step size of about 9 ohms per step which gives us an accuracy of less than 0.12 degrees.

We use these potentiometers in an inverting amplifier circuit to adjust the gain from 0V to 2.5V in the first stage.

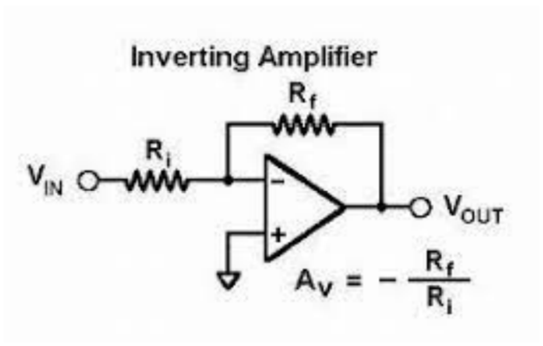


Figure 8: Inverting Amplifier

5.2.4 High Voltage Amplifier

The high voltage amplification section uses fixed resistors so they can handle the voltage drop sing we will be working with 11.8 and 26 Vrms. As stated before the stator sections will be non-inverting while the rotor will be inverting.

Gain

$$= 1 + (\frac{V_2}{V_1})$$

5.3 Design Testing

Our entire design was tested in ORCad using the simulation models built in. We inputted a reference sine voltage from what would be the digital synthesis chips and then tested the outputs from 40mVp to 11.8Vrms. The final design being tested and the simulation outputs will be shown below. The first two figures show the circuit at its 40mVp state and the last two show the 11.8Vrms state.

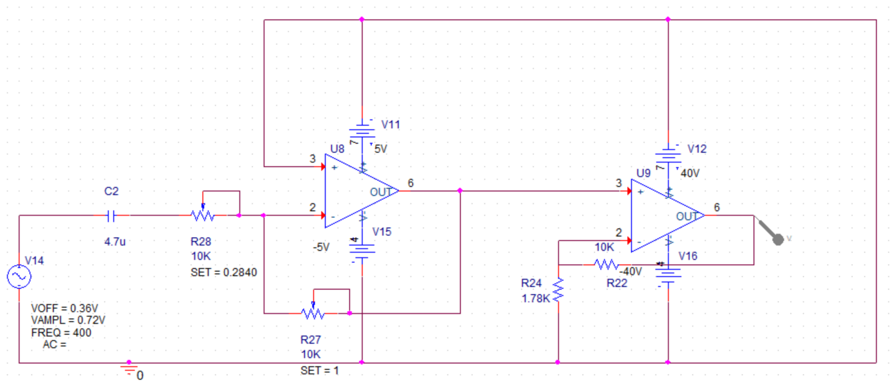


Figure 9: OrCAD Circuit Model

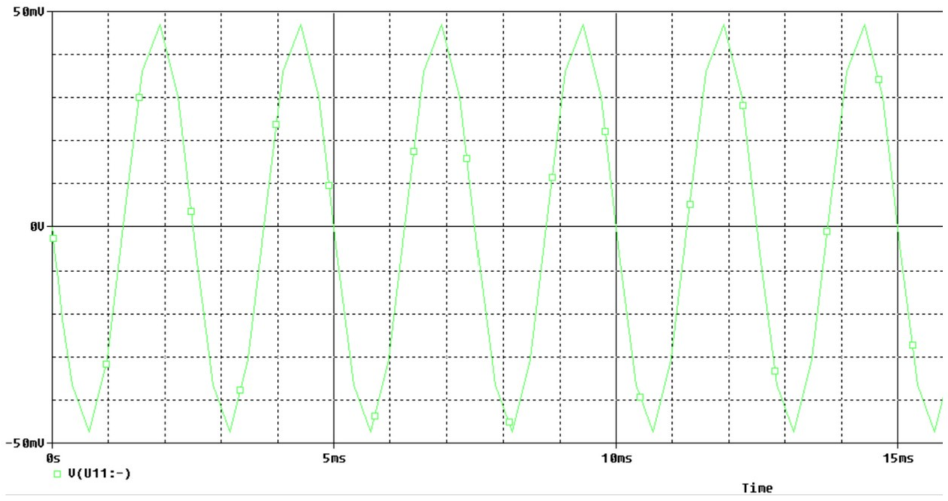


Figure 10: Circuit Waveform Output

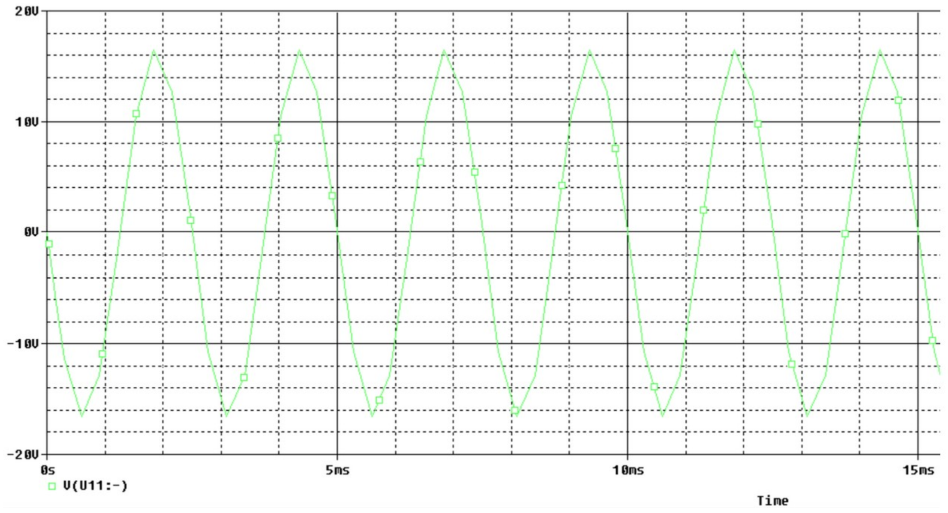


Figure 11: Waveform output

5.4 Design Choices - Reasoning

The previous designs of a digital to synchro converter used an expensive, no longer produced device called a Scott-T transformer. Instead of trying to source a transformer we decided to use digital hardware instead.

After testing some implementations of a sine wave including using PWM generation from the Arduino Uno, we decided to go with digital synthesis. We chose to use the digital synthesis IC due to its easy functionality, low cost and easy phase shifting capabilities.

We decided to use the digital potentiometers because after testing some other options with transistors this gave us the best reliability. The other circuits we tried to come up with didn't work reliably and due to the low cost of the potentiometers we decided on them pretty early in the design.

Another option we had for high power output was using transformers which would allow us to forgo the external power supply. Due to the high cost and bulkiness we decided to go with the op-amps. The main issue with op-amps is the reliability and possible phase shifting. These issues didn't change our decision in the end because all of those issues can be fixed with software.

The final major decision we made was on the use of the Arduino Uno. We explored other options such as a Raspberry Pi and a BeagleBone Black. We ultimately decided on the Arduino Uno due to the high compatibility, reliability and low cost. In addition to this we already had some Arduino Uno's so that made it easier to test.

5.5 PCB

The first thing we had to do before routing the PCB was to create parts in KiCad since most of them didn't exist in the libraries. While most of the packages were standard the schematic part had to be created from scratch.

After getting all of the parts inputted into KiCad we then connected everything according to our design and checked everything with the design rule check built into KiCad. Certain things we built into the schematic are multiple pages for easy readability as well as assigning the chip select and SPI pins to each of the chips. Everything on the schematic should be easily readable and will be included below.

After verifying our schematic and making sure everything was hooked up according to the manufacturer specifications we started placing the components on the PCB. Each device was placed to not only look as professional as possible but making routing as easy as possible. One issue we had was the limited size for routing due to having to fit in the Arduino Uno shield specifications. We decided to use a 4 layer PCB so we can have 3 layers to route traces and have a ground plane to allow for easier routing. We made sure to include plenty of test pins on our PCB to allow for easy testing the lab to make sure everything works as intended.

We then needed to route all of the pins, while adhering to the PCB manufacturers specifications. We decided to order through JLCPCB since their services are fast and high quality. We started by inputting all of the design constraints JLCPCB states on their website to make sure we don't do anything that they can't produce. Included this is our via sizing, which we made as small as possible to allow for more space as well as trace size which we sized to allow for up to 1A of current using an online trace size calculator.

The routing was one of the more difficult aspects of designing the PCB. Initially we started by fully routing one IC at a time but it proved very inefficient with the limited space available on the PCB. After scrapping that methodology, we decided to route the power to

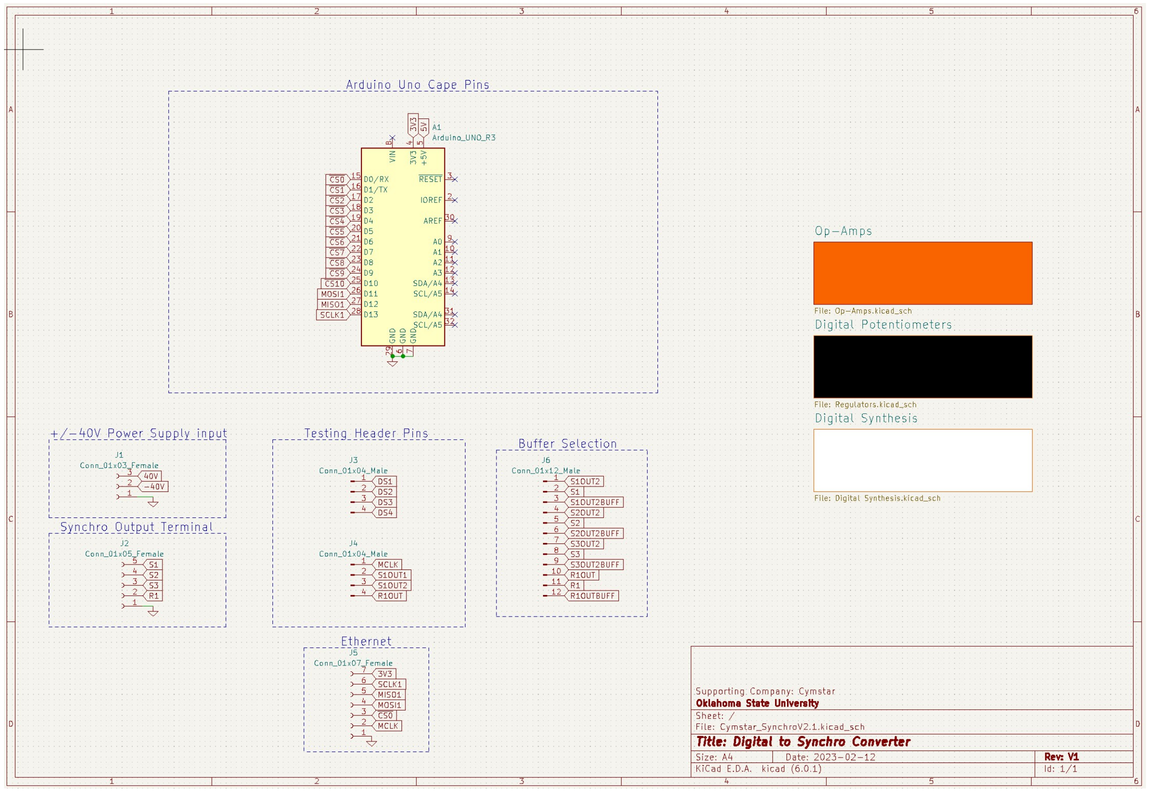
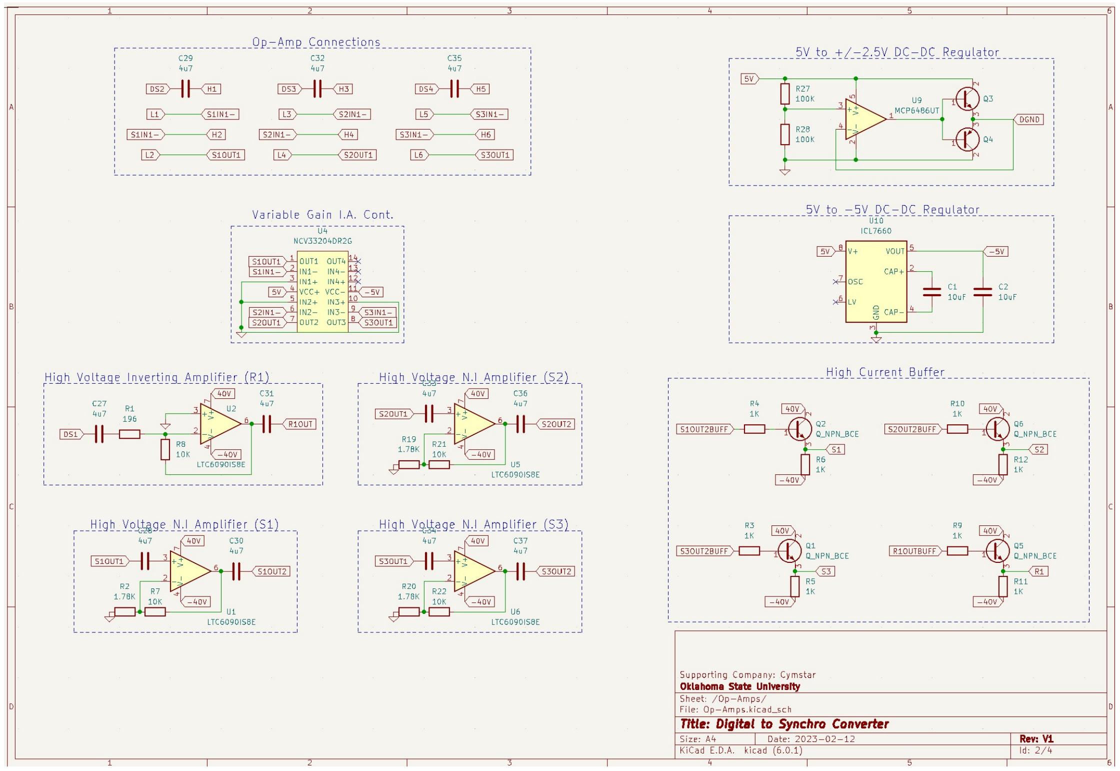

Arduino Uno Cape Pins
A1
Arduino_UNO_R3
Op−Amps
File: Op−Amps.kicad_sch
Digital Potentiometers
File: Regulators.kicad_sch
Digital Synthesis
File: Digital Synthesis.kicad_sch
+/−40V Power Supply input
Testing Header Pins
Buffer Selection
Synchro Output Terminal
Ethernet
Supporting Company: Cymstar
Oklahoma State University
Sheet: /
File: Cymstar_SynchroV2.1.kicad_sch
Title: Digital to Synchro Converter
Size: A4
Date: 2023−02−12
Rev: V1
KiCad E.D.A. kicad (6.0.1)
Id: 1/1

Figure 12: Arduino Uno Pins

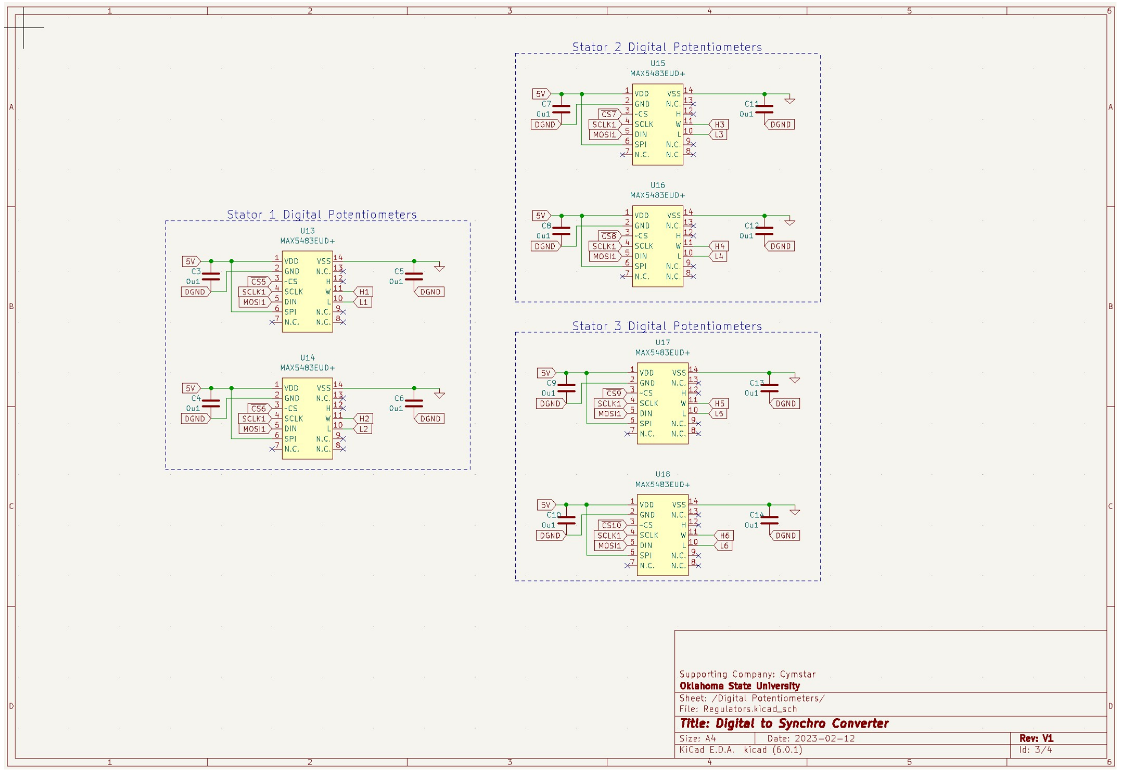
each component first and then connect the pins of closely located components.

The first and fourth layers were used to place components on and routing pins to each other based on the schematics. The second layer was primarily used for routing power to components and other miscellaneous routing that could not be done on the first and fourth layers. The third layer was used as a ground plane that every IC connected to.



Op-Amp Connections
Variable Gain I.A. Cont.
5V to +/-2.5V DC-DC Regulator
5V to -5V DC-DC Regulator
High Voltage Inverting Amplifier (R1)
High Voltage N.I Amplifier (S2)
High Voltage N.I Amplifier (S1)
High Voltage N.I Amplifier (S3)
High Current Buffer
Supporting Company: Cymstar
Oklahoma State University
Sheet: /Op-Amps/
File: Op-Amps.kicad_sch
Title: Digital to Synchro Converter
Size: A4
Date: 2023-02-12
Rev: V1
KiCad E.D.A. kicad (6.0.1)
Id: 2/4

Figure 13: Arduino Uno Shield Pin Connections to Components



Stator 1 Digital Potentiometers
Stator 2 Digital Potentiometers
Stator 3 Digital Potentiometers
Supporting Company: Cymstar
Oklahoma State University
Sheet: /Digital Potentiometers/
File: Regulators.kicad_sch
Title: Digital to Synchro Converter
Size: A4
Date: 2023-02-12
Rev: V1
KiCad E.D.A. kicad (6.0.1)
Id: 3/4

Figure 14: Arduino Uno Shield Pin Connections to Components 2

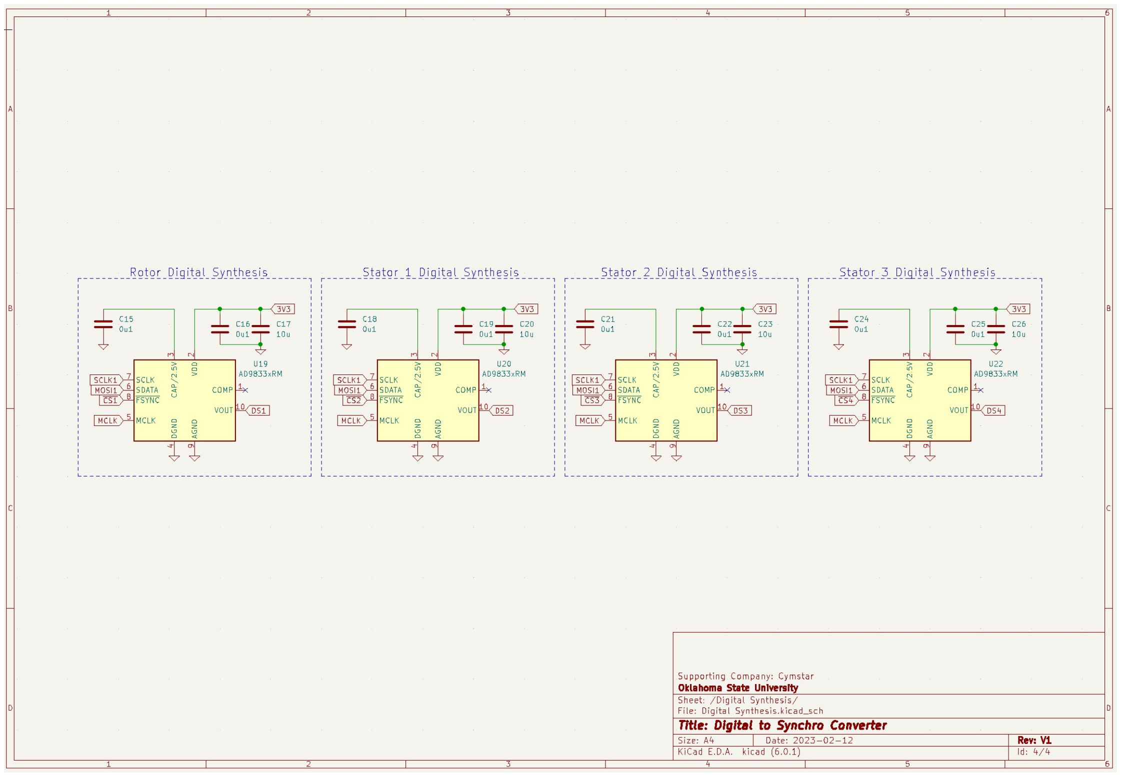


Figure 15: Arduino Uno Shield Pin Connections to Components 3

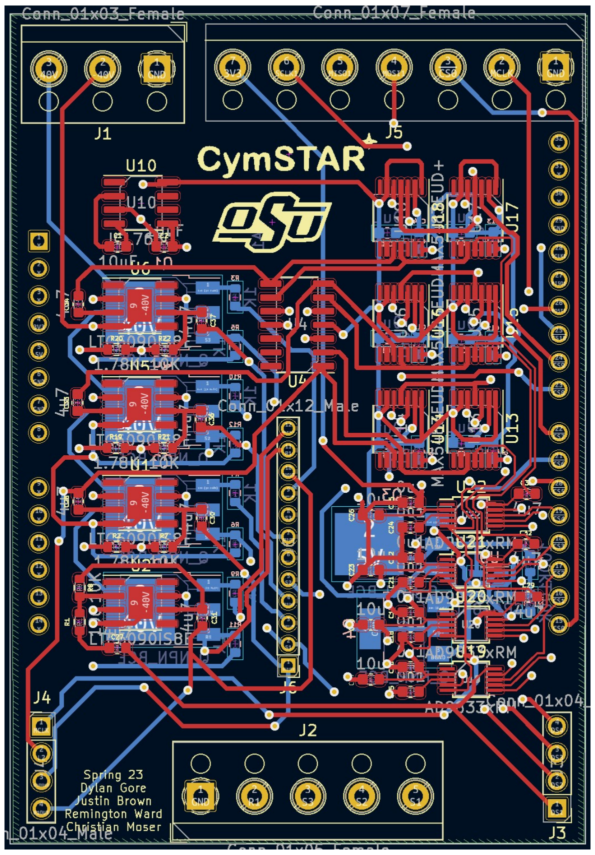


Figure 16: PCB Design in KiCAD

6 Software Implementation - Christian and Remington

6.1 Arduino Board - Remington

The Arduino board we went with was the Arduino Uno Rev3. The main reasons we went with the Arduino Uno Rev3 is because it's a smaller board compared to other Arduino's (2.7*2.1 inches) to help fit in the required dimensions given to us, it weighs only 25 grams, and it has enough I/O pins for all of the digital chip components and for the Ethernet cape. Arduino is easy to write code for as it's all in C and the digital components we had that needed to be written in SPI already had libraries made for them.

The Arduino Uno Rev3 operates at 5V so plugging it into a computer is optimal, but it can almost be plugged into a small voltage supply as the recommended input voltage is 7-12V. The clock speed of the Arduino is 16MHz which is plenty fast when using the built in clock for our chips. The Uno has 14 Digital I/O Pins and we use all 14 of the pins for our components. We use pins 1-13 for the SPI devices and we use pin 0 for the Ethernet. We use pin 0 for the Ethernet as it is the RX and we only need to receive information from the host computer through the Ethernet, and we have no reason to transmit information back to the host computer.

One problem with using the Ethernet is that we can not serial write to a terminal to check that the values are exactly what they need to be. This is one downfall of using Ethernet with Arduino, as we are not able to decode as easily when sending byte strings through TCP.

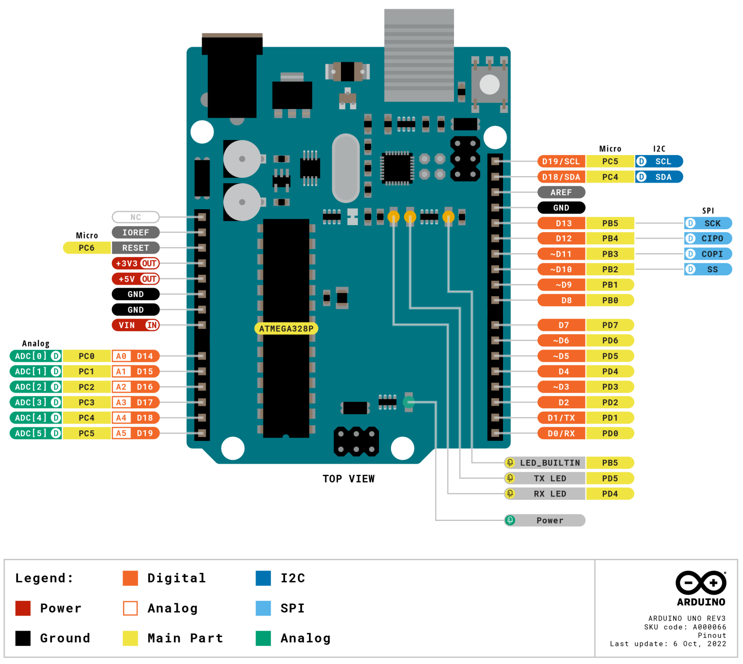


Figure 17: Arduino Uno Rev3 pinout.

6.2 Ethernet Module - Remington

The Arduino Ethernet cape we went with was the HiLetgo ENC28J60 module for the RJ45 Ethernet connector. This Ethernet cape has an on-board 25MHz crystal that we utilized as the master clock for the digital chips to make sure they all operate together with the rising-edge. We used 6.25MHz as the master clock frequency as we found out that the higher the master clock frequency the less reliant the digital components were at outputting the sine waves. The recommended power supply for the module is 3.3V which is plugged into the 3V3 terminal on the PCB in section J5.

We have the Ethernet module plugged into the 3V3 terminal on the PCB (Pin VCC on the Ethernet module), SCLK terminal (Pin SCK), MCLK terminal (Pin CLKOUT), MISO terminal (Pin SO), MOSI terminal (Pin SI), CS terminal (Pin CS), and GND terminal (Pin GND).

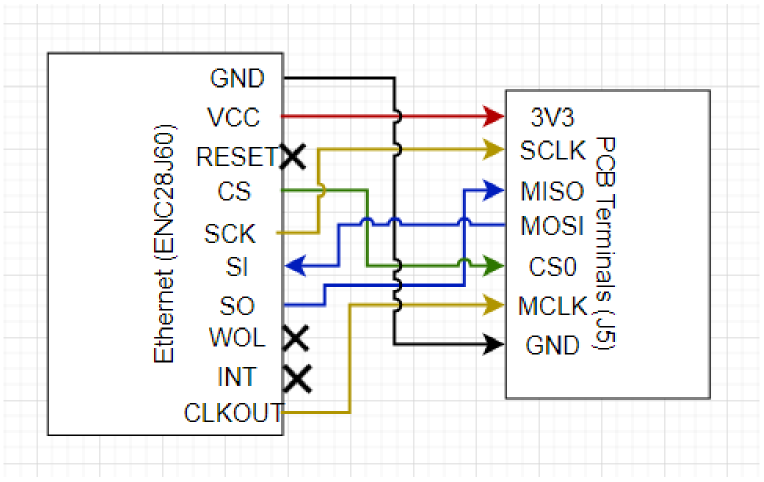
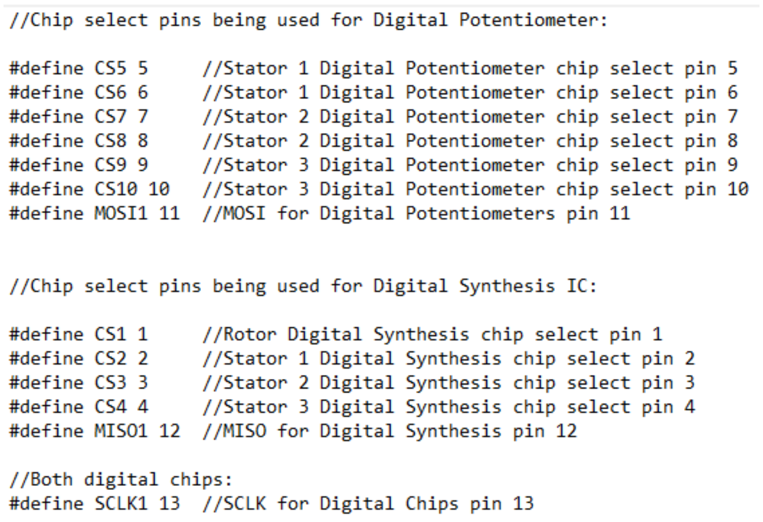


Figure 18: Ethernet Connections to the J5 PCB terminals.

6.3 Arduino Pin Configuration - Remington

The pin configuration for the SPI digital chips does not require any cables to be plugged into the board as it is all done with the PCB cape that was specifically made for the Arduino Uno Rev3. All of the chips have already been defined with their own chip-select from the Arduino pins. Below are the defined chip-select commands that are used for each digital chip and it states which chip select pin is for what digital chip.



```
//Chip select pins being used for Digital Potentiometer:

#define CS5 5     //Stator 1 Digital Potentiometer chip select pin 5
#define CS6 6     //Stator 1 Digital Potentiometer chip select pin 6
#define CS7 7     //Stator 2 Digital Potentiometer chip select pin 7
#define CS8 8     //Stator 2 Digital Potentiometer chip select pin 8
#define CS9 9     //Stator 3 Digital Potentiometer chip select pin 9
#define CS10 10   //Stator 3 Digital Potentiometer chip select pin 10
#define MOSI1 11  //MOSI for Digital Potentiometers pin 11


//Chip select pins being used for Digital Synthesis IC:

#define CS1 1     //Rotor Digital Synthesis chip select pin 1
#define CS2 2     //Stator 1 Digital Synthesis chip select pin 2
#define CS3 3     //Stator 2 Digital Synthesis chip select pin 3
#define CS4 4     //Stator 3 Digital Synthesis chip select pin 4
#define MISO1 12  //MISO for Digital Synthesis pin 12

//Both digital chips:
#define SCLK1 13  //SCLK for Digital Chips pin 13
```

Figure 19: Defining chip selects for each of the pins and commenting what each select is connected to.

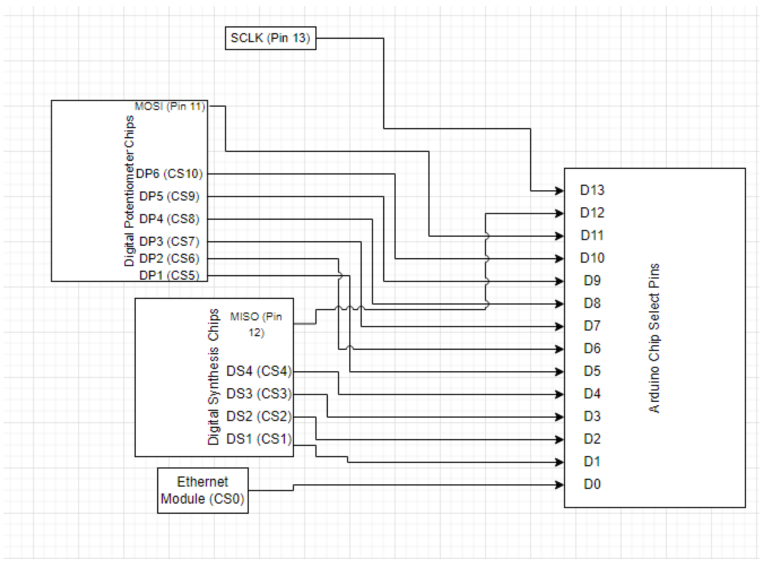


Figure 20: Digital Chips, Ethernet, and SCLK pin connections.

6.4 Equations - Christian

We utilized the following equation for determining the firmware values:

$$\bar{V_s} = A_0 \cos\left(\theta - \bar{\phi_{err}}\right)$$

where $A_0 = 16.68755$ and $\bar{\phi_{err}} = \begin{bmatrix} 0^\circ \\ 120^\circ \\ 240^\circ \end{bmatrix}$

Extrapolating from above, we used the following to derive step-size calculations:

$$V_{si+1} = \frac{d[\bar{V_{si}}]}{d\theta}$$
$$V_{si+1} - V_{si} = 0.009$$

Thus we derived per voltage step size.

We also accounted for bias from phase measurements of real-life testing:

$$\Delta_\phi = \bar{\phi} - \bar{\beta}$$

where $\beta = 51.84^\circ$

and $\bar{\phi_{err}} = \Delta_\phi = \begin{bmatrix} 0^\circ \\ 120^\circ \\ 240^\circ \end{bmatrix}$

6.5 Firmware Libraries - Remington

We use four libraries with three of the libraries being specifically created for the digital chips and the Ethernet module we use in our design. The one built-in library that we utilize through Arduino is the math.h library that does not require any downloading and comes with the Arduino IDE download. The digital synthesis chips use the AD9833.h library created by Billwilliams1952 on GitHub (2). The digital potentiometer chips use the MAX5481.h library created by robertfchapman on GitHub (3). The Ethernet module, HiLetgo ENC28J60, uses the UIPEthernet.h library created by JAndrassy on GitHub (4). All three of these libraries are available on the GitHub we made for this project and can be downloaded from there in the "libraries" folder. When importing the libraries into the Arduino IDE click Sketch ¿ Include Library ¿ Add, and make sure the library is a .ZIP folder when adding the library.

6.6 Graphical User Interface (GUI) - Christian

6.6.1 Introduction

This section describes how to use the Synchro Digital Controller, which is a Python program that allows you to control a device over TCP/IP. The program includes a GUI built with Tkinter, which provides a variety of controls and visualizations for the device.

6.6.2 Requirements

The following software packages are required to use the Synchro Digital Controller:

- Python 3.x
- Tkinter
- Matplotlib
- NumPy
- Asyncio
- PySerial
- PIL

In addition, you will need access to a copy of the ethernet-to-synchro device that can be controlled over TCP/IP or you can override the pre-programmed TCP/IP address and port value to create your own backend for demonstrative purposes.

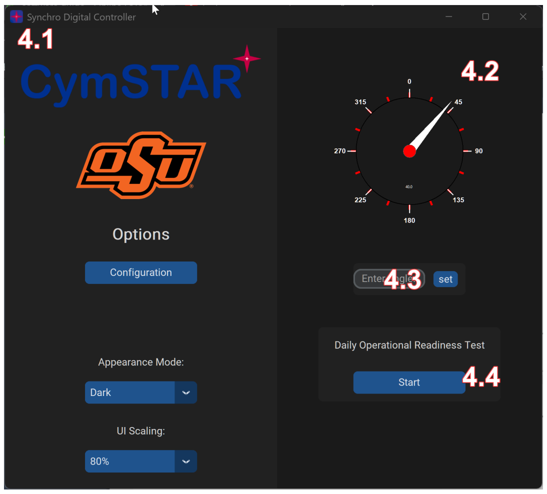
6.6.3 Installation

To install the Synchro Digital Controller, follow these steps:

1. Download the source code from the GitHub repository.
2. Open a terminal or command prompt and navigate to the directory where the source code is located.
3. Install the required packages using the following command:
```
    python3 -m pip install -r requirements.txt
```
4. Run the program using the following command:
```
    python3 main.py
```

6.6.4 Usage

When you run the program, a GUI window will appear that contains several controls and visualizations for the device.



Appearance Mode: Dark

UI Scaling: 80%

Figure 21: Image capture of GUI with the following sections labelled

Sidebar The sidebar contains several options for configuring the program:

- **Configuration**: Opens a configuration dialog that allows you to set the IP address and port for the connected device, change the starting angle, and turn on an extra menu for programming the Arduino through the GUI.
- **Appearance Mode**: Allows you to select the appearance mode for the program. The available modes are "Light", "Dark", and "System".
- **UI Scaling**: Allows you to adjust the size of the user interface. The available options are "80%", "90%", "100%", "110%", and "120%".

Main Gauge The gauge displays the current angle value of the synchro device on a scale of degrees out of 360°.

Angle Entry Box The entry box below the gauge can be used to input any floating-point number to change the dial value and transmit the angle to the connected device.

Device Operational Readiness Test This section utilizes an additional panel to run an operational readiness test. This test works by incrementing the gauge by 90° until returning back to 0° to test the accuracy and operational readiness of the connected device.

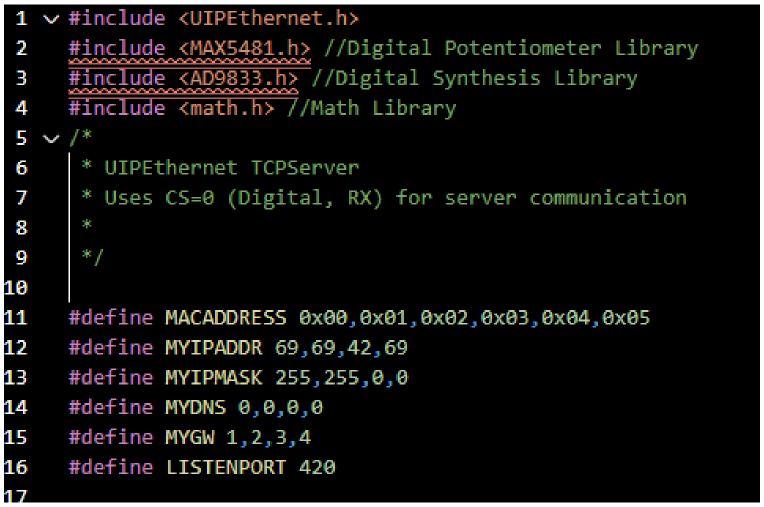
6.6.5 Troubleshooting

If you encounter any issues while using the Synchro Digital Controller application, please consult the documentation or contact the developer for assistance.

6.7 Serial Peripheral Interface - SPI - Remington

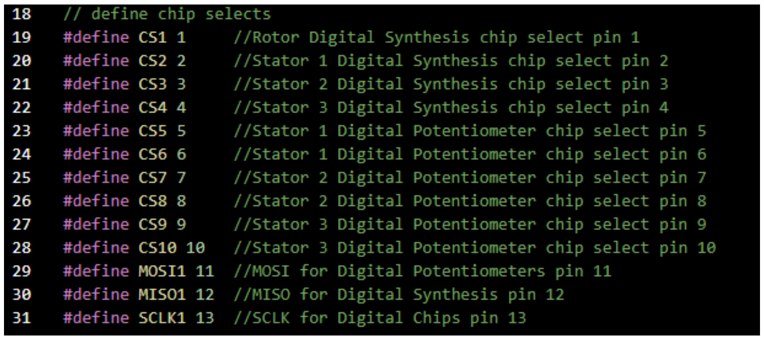
The SPI portion of the computer host is what drives the Digital Synthesis Chip and the Digital Potentiometer Chip. We chose to use SPI over the other pin connections because it gives us more control when using the Arduino Uno Rev3, and these two chips work best with SPI.

The SPI code and TCP server code was compiled together as we need to take in the information from the GUI through the Ethernet. The code first imports the necessary libraries needed to control the digital chips, get information from the Ethernet, and allow the program to implement math equations. Next the MAC Address, IP Address and masking, and the listen port are all defined with the pins for the digital chips and the Ethernet chip select. The constant values under the defining pins are set to these values: The MCLK frequency is set to 6.25MHz, and this is because the output from the crystal on the Ethernet module is 6.25MHz. The stator frequency is set to 400Hz as we need the output sine signals to be in 400Hz from the digital synthesis chips, and the initial resistance can be changed in the GUI to what initial angle one wants to start at. It is default set to step 1023, the max steps on the digital potentiometers. The minimum and maximum resistance the digital potentiometers can be set to are also set as constants.



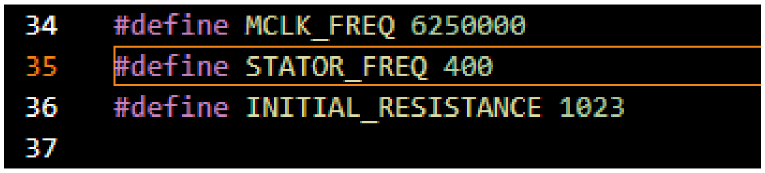
```
#include <UIPEthernet.h>
#include <MAX5481.h> //Digital Potentiometer Library
#include <AD9833.h> //Digital Synthesis Library
#include <math.h> //Math Library
/*
 * UIPEthernet TCPServer
 * Uses CS=0 (Digital, RX) for server communication
 *
 */

#define MACADDRESS 0x00,0x01,0x02,0x03,0x04,0x05
#define MYIPADDR 69,69,42,69
#define MYIPMASK 255,255,0,0
#define MYDNS 0,0,0,0
#define MYGW 1,2,3,4
#define LISTENPORT 420
```

Figure 22: Importing necessary libraries and setting Ethernet values.



```
// define chip selects
#define CS1 1     //Rotor Digital Synthesis chip select pin 1
#define CS2 2     //Stator 1 Digital Synthesis chip select pin 2
#define CS3 3     //Stator 2 Digital Synthesis chip select pin 3
#define CS4 4     //Stator 3 Digital Synthesis chip select pin 4
#define CS5 5     //Stator 1 Digital Potentiometer chip select pin 5
#define CS6 6     //Stator 1 Digital Potentiometer chip select pin 6
#define CS7 7     //Stator 2 Digital Potentiometer chip select pin 7
#define CS8 8     //Stator 2 Digital Potentiometer chip select pin 8
#define CS9 9     //Stator 3 Digital Potentiometer chip select pin 9
#define CS10 10   //Stator 3 Digital Potentiometer chip select pin 10
#define MOSI1 11  //MOSI for Digital Potentiometers pin 11
#define MISO1 12  //MISO for Digital Synthesis pin 12
#define SCLK1 13  //SCLK for Digital Chips pin 13
```

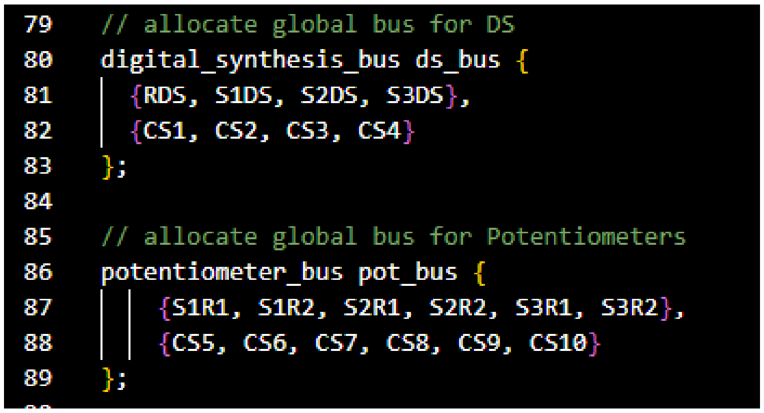
Figure 23: Defining digital pin chip selects.



```
#define MCLK_FREQ 6250000
#define STATOR_FREQ 400
#define INITIAL_RESISTANCE 1023

```

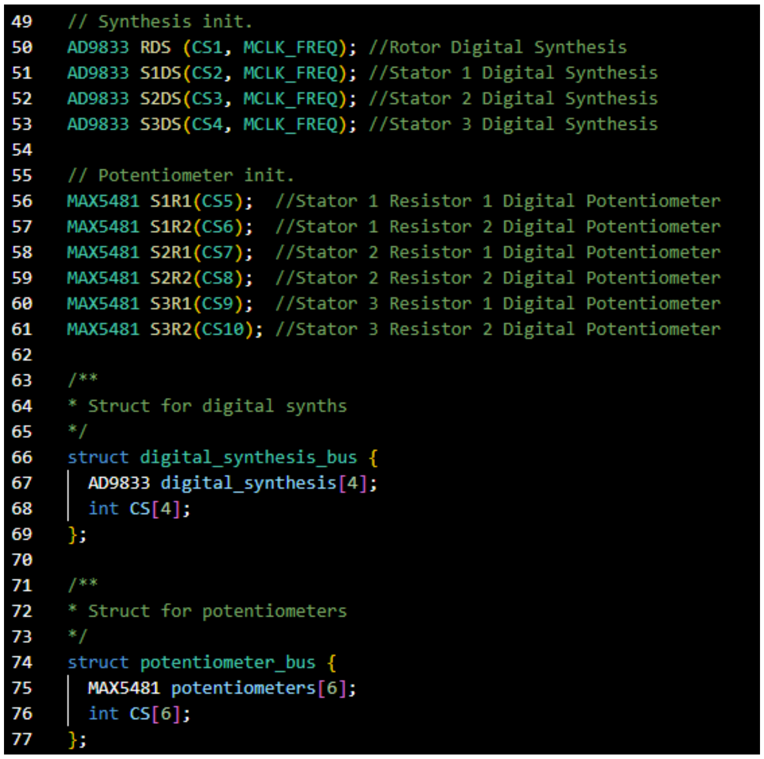
Figure 24: Defining constants for the digital chips.

We then initialize the digital synthesis and digital potentiometer chips. Digital synthesis chips are initialized like: AD9833 definedName(CS number, Master clock freq). Digital potentiometer chips are initialized like: MAX5481 definedName(CS number). After we structure and allocate the digital potentiometers and digital synthesis chips into a struct bus with a set array storage size depending on how many chips of each are being used: digital synthesis are set to size 4 and digital potentiometers are set to size 6. Next in the void function initDigitalSynths() a for loop is used to set all 4 digital synthesis chips to begin, apply the wanted signal and set the frequency output. The void initPotentiometers() uses a for loop to set all 6 digital potentiometers to begin and to start reading the wiper resistance. Outside the for loop the wipers on the digital potentiometers are set to the initial resistance variable (1023 steps).



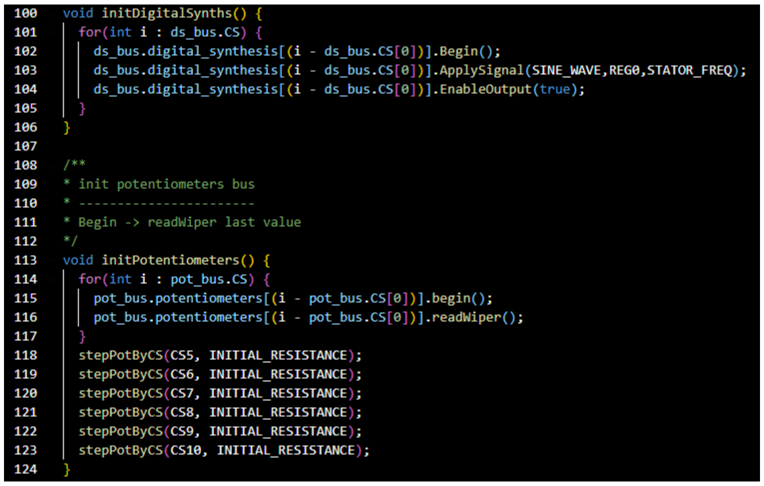
```
// allocate global bus for DS
digital_synthesis_bus ds_bus {
  {RDS, S1DS, S2DS, S3DS},
  {CS1, CS2, CS3, CS4}
};

// allocate global bus for Potentiometers
potentiometer_bus pot_bus {
    {S1R1, S1R2, S2R1, S2R2, S3R1, S3R2},
    {CS5, CS6, CS7, CS8, CS9, CS10}
};
```

Figure 25: Storing the digital chip calls into bus arrays.



```
// Synthesis init.
AD9833 RDS (CS1, MCLK_FREQ); //Rotor Digital Synthesis
AD9833 S1DS(CS2, MCLK_FREQ); //Stator 1 Digital Synthesis
AD9833 S2DS(CS3, MCLK_FREQ); //Stator 2 Digital Synthesis
AD9833 S3DS(CS4, MCLK_FREQ); //Stator 3 Digital Synthesis

// Potentiometer init.
MAX5481 S1R1(CS5);  //Stator 1 Resistor 1 Digital Potentiometer
MAX5481 S1R2(CS6);  //Stator 1 Resistor 2 Digital Potentiometer
MAX5481 S2R1(CS7);  //Stator 2 Resistor 1 Digital Potentiometer
MAX5481 S2R2(CS8);  //Stator 2 Resistor 2 Digital Potentiometer
MAX5481 S3R1(CS9);  //Stator 3 Resistor 1 Digital Potentiometer
MAX5481 S3R2(CS10); //Stator 3 Resistor 2 Digital Potentiometer

/**
* Struct for digital synths
*/
struct digital_synthesis_bus {
  AD9833 digital_synthesis[4];
  int CS[4];
};

/**
* Struct for potentiometers
*/
struct potentiometer_bus {
  MAX5481 potentiometers[6];
  int CS[6];
};
```

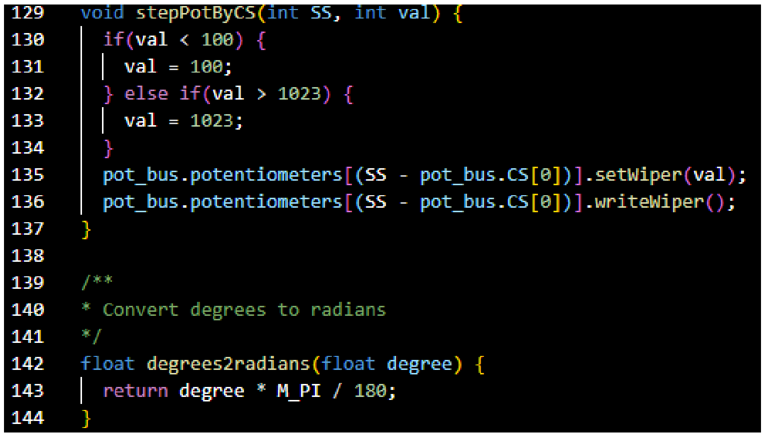
Figure 26: Initializing digital synthesis and digital potentiometer chips.



```
void initDigitalSynths() {
  for(int i : ds_bus.CS) {
    ds_bus.digital_synthesis[(i - ds_bus.CS[0])].Begin();
    ds_bus.digital_synthesis[(i - ds_bus.CS[0])].ApplySignal(SINE_WAVE,REG0,STATOR_FREQ);
    ds_bus.digital_synthesis[(i - ds_bus.CS[0])].EnableOutput(true);
  }
}

/**
* init potentiometers bus
* ----------------------
* Begin -> readWiper last value
*/
void initPotentiometers() {
  for(int i : pot_bus.CS) {
    pot_bus.potentiometers[(i - pot_bus.CS[0])].begin();
    pot_bus.potentiometers[(i - pot_bus.CS[0])].readWiper();
  }
  stepPotByCS(CS5, INITIAL_RESISTANCE);
  stepPotByCS(CS6, INITIAL_RESISTANCE);
  stepPotByCS(CS7, INITIAL_RESISTANCE);
  stepPotByCS(CS8, INITIAL_RESISTANCE);
  stepPotByCS(CS9, INITIAL_RESISTANCE);
  stepPotByCS(CS10, INITIAL_RESISTANCE);
}
```

Figure 27: Setting constant vales to the digital synthesis and digital potentiometer chips.

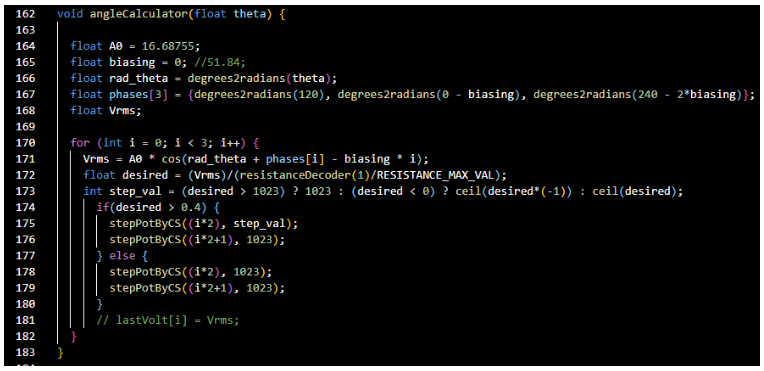
While testing, the digital potentiometers were shorted due to all of them being set to 0 steps (70 ohms). To make sure that the resistance of all the digital potentiometers would never all be set to 70 ohms a function was created to check and if they are set all to 0 steps then set them to 100 steps to avoid shorting. There is also a simple function made to change degrees to radians since the math library does not automatically convert.



```
void stepPotByCS(int SS, int val) {
  if(val < 100) {
    val = 100;
  } else if(val > 1023) {
    val = 1023;
  }
  pot_bus.potentiometers[(SS - pot_bus.CS[0])].setWiper(val);
  pot_bus.potentiometers[(SS - pot_bus.CS[0])].writeWiper();
}

/**
* Convert degrees to radians
*/
float degrees2radians(float degree) {
  return degree * M_PI / 180;
}
```

Figure 28: Checking to make sure the digital potentiometers are not all set to step 0 and a function to convert degrees to radians.

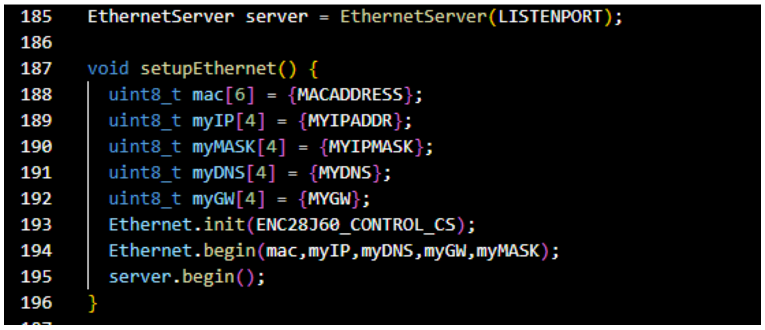
Using the voltage synchro formula with the new found stator degrees: [13.5967, -0.0621, -5.55376] are stored in a float array. Float current angle and float theta are called as global variables. Theta is the variable that stores the degree from the GUI, and the currAngle is saved as the current angle being used until a new theta value is added. An interrupt is used that keeps checking to see if theta does not equal the current angle. If that is the case then find the difference between the angles and set the new resistor values in the digital potentiometers. The stator voltage formula is used after a new theta is given from the GUI to determine the new digital potentiometer steps to acquire the 3 new stator signals.



```
void angleCalculator(float theta) {

  float A0 = 16.68755;
  float biasing = 0; //51.84;
  float rad_theta = degrees2radians(theta);
  float phases[3] = {degrees2radians(120), degrees2radians(0 - biasing), degrees2radians(240 - 2*biasing)};
  float Vrms;

  for (int i = 0; i < 3; i++) {
    Vrms = A0 * cos(rad_theta + phases[i] - biasing * i);
    float desired = (Vrms)/(resistanceDecoder(1)/RESISTANCE_MAX_VAL);
    int step_val = (desired > 1023) ? 1023 : (desired < 0) ? ceil(desired*(-1)) : ceil(desired);
      if(desired > 0.4) {
        stepPotByCS((i*2), step_val);
        stepPotByCS((i*2+1), 1023);
      } else {
        stepPotByCS((i*2), 1023);
        stepPotByCS((i*2+1), 1023);
      }
      // lastVolt[i] = Vrms;
  }
}
```

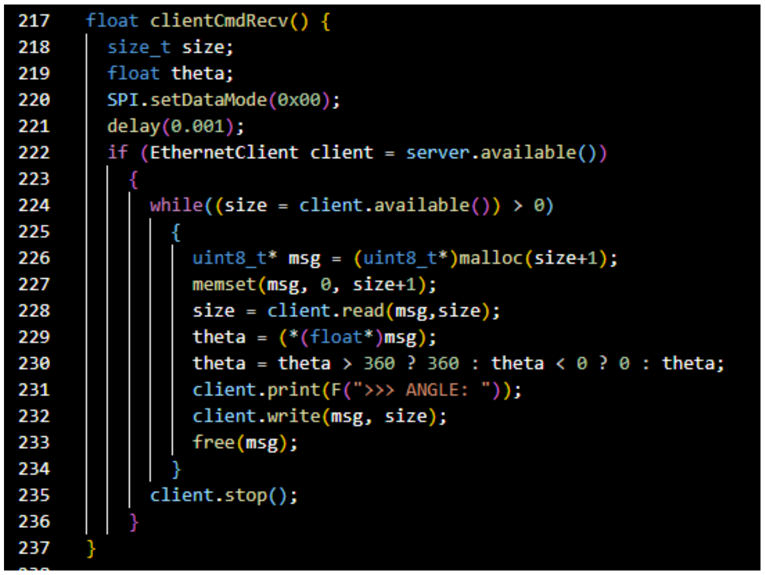
Figure 29: Function that utilizes stator voltage formula to set new digital potentiometer values.

The Ethernet is set up as a server and as a listen port since it is given new theta values from the host computer. A function is used setupEthernet() that uses all the defining for the Ethernet earlier and inputs them into uint8t arrays. The Ethernet is then initialized and set to begin and listen. The Ethernet is set to echo client to check that there is a connection available and sends an echo to the client and gets ready to write. Another client function clientCmdRecv() is for reading the sent data stream from the GUI and converts the stream to a float which is saved in the theta variable.



```
EthernetServer server = EthernetServer(LISTENPORT);

void setupEthernet() {
  uint8_t mac[6] = {MACADDRESS};
  uint8_t myIP[4] = {MYIPADDR};
  uint8_t myMASK[4] = {MYIPMASK};
  uint8_t myDNS[4] = {MYDNS};
  uint8_t myGW[4] = {MYGW};
  Ethernet.init(ENC28J60_CONTROL_CS);
  Ethernet.begin(mac,myIP,myDNS,myGW,myMASK);
  server.begin();
}
```

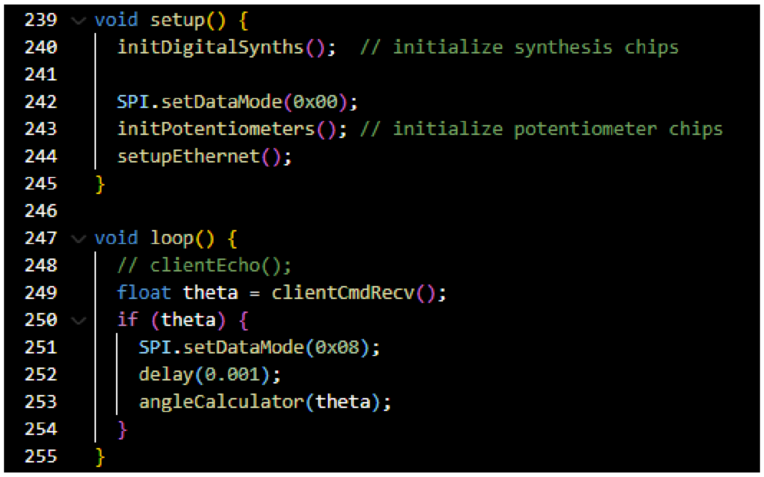
Figure 30: Setting up Ethernet values from constants defined.



```
float clientCmdRecv() {
  size_t size;
  float theta;
  SPI.setDataMode(0x00);
  delay(0.001);
  if (EthernetClient client = server.available())
    {
      while((size = client.available()) > 0)
        {
          uint8_t* msg = (uint8_t*)malloc(size+1);
          memset(msg, 0, size+1);
          size = client.read(msg,size);
          theta = (*(float*)msg);
          theta = theta > 360 ? 360 : theta < 0 ? 0 : theta;
          client.print(F(">>> ANGLE: "));
          client.write(msg, size);
          free(msg);
        }
      client.stop();
    }
}
```

Figure 31: Setting up TCP Ethernet server and reading Ethernet port for new theta values.

The setup() function is calling and initializing functions for the Ethernet, digital synthesis chips, digital potentiometers, and setting SPI data mode to 0x00. Inside the loop function theta is set equal to the clientCmdRecv() function and keeps checking to see if the theta value changes.



```
void setup() {
  initDigitalSynths();  // initialize synthesis chips

  SPI.setDataMode(0x00);
  initPotentiometers(); // initialize potentiometer chips
  setupEthernet();
}

void loop() {
  // clientEcho();
  float theta = clientCmdRecv();
  if (theta) {
    SPI.setDataMode(0x08);
    delay(0.001);
    angleCalculator(theta);
  }
}
```

Figure 32: Setting up function and looping function to check for new theta values.

7 Tests and Experimentation

7.1 Hardware Test Plan

- We tested the synchros in the Endeavor lab for functionality 26V rms, 400Hz sine wave on rotor, 3 oscilloscope probes on stator pins (S1, S2, S3).
- Verify PCB design in ORCAD.
- Using oscilloscope verify PCB output before connecting to synchro.

7.2 Hardware Test Results

- The Synchros were successfully tested in the Endveaor Labs to prove they work how we expect them to.
- In the original design we found issue with the BeagleBone Black so testing helped us decide to change to another device.
- The second revision was able to act as the rotor voltage but had some clipping issues on the negative edge of the stator waves.
- We identified a failure mode when testing as too much current was drawn through the digital potentiometers causing them to fail.

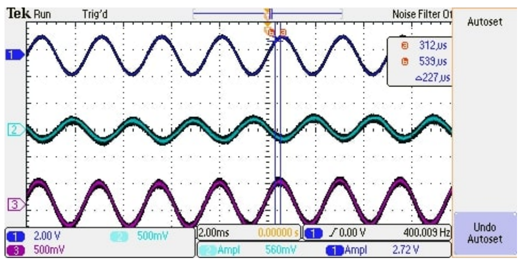


Figure 33: Waveforms of Synchro

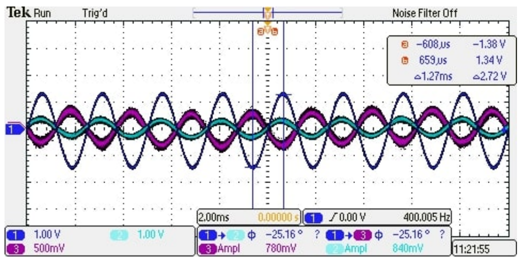


Figure 34: Waveforms of Synchro

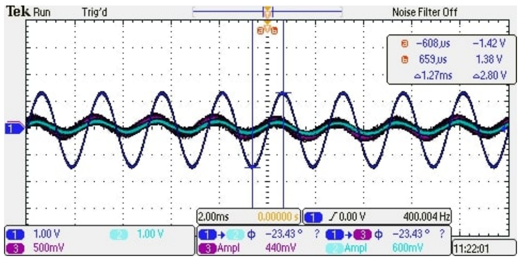


Figure 35: Waveforms of Synchro

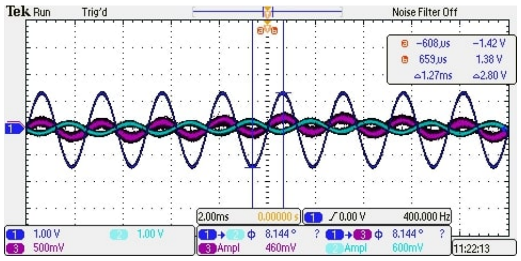


Figure 36: Waveforms of Synchro

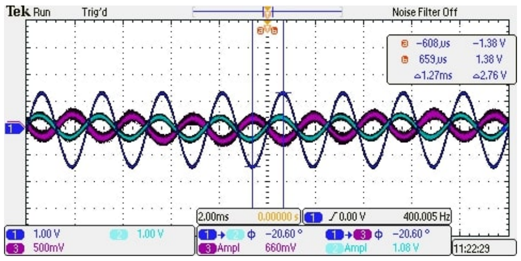


Figure 37: Waveforms of Synchro

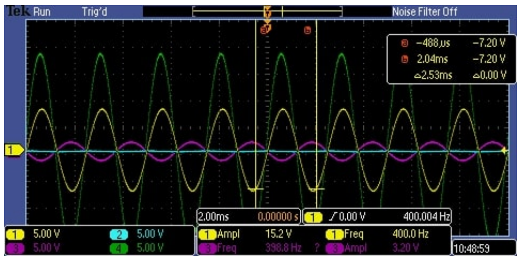


Figure 38: Waveforms of Synchro

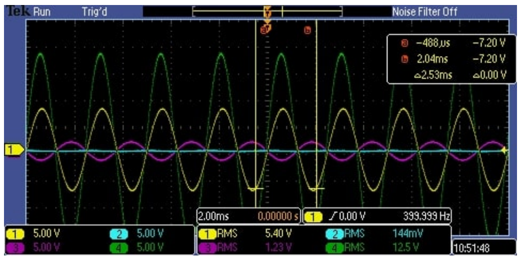


Figure 39: Waveforms of Synchro

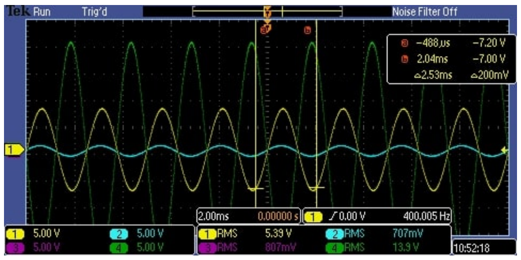


Figure 40: Waveforms of Synchro

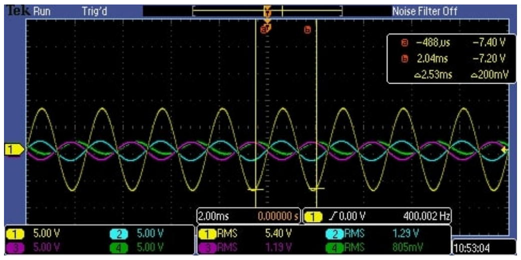


Figure 41: Waveforms of Synchro

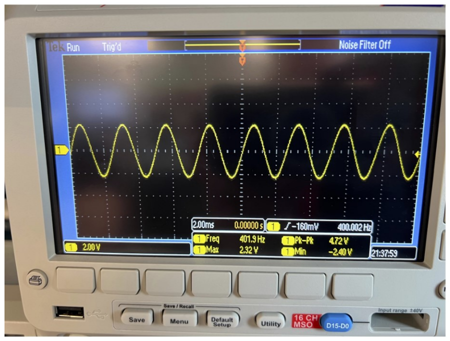


Figure 42: Generated Sine Wave for Rotor

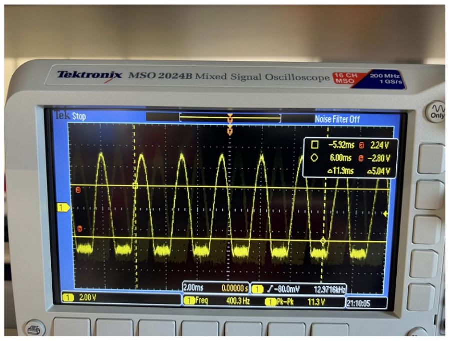
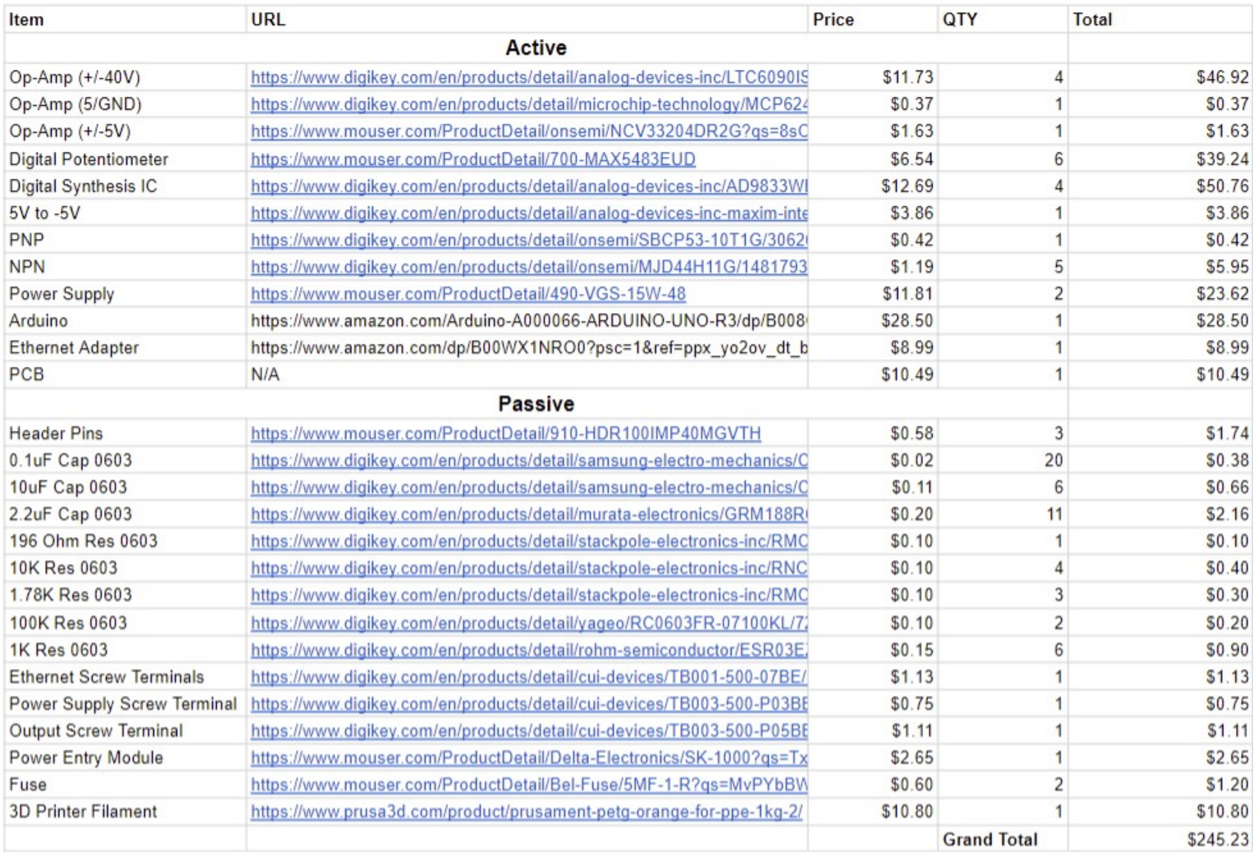


Figure 43: Generated Sine Wave with Clipping

8 Project Costs - Dylan



Item	URL	Price	QTY	Total
	Active			
Op-Amp (+/-40V)	https://www.digikey.com/en/products/detail/analog-devices-inc/LTC6090IS	$11.73	4	$46.92
Op-Amp (5/GND)	https://www.digikey.com/en/products/detail/microchip-technology/MCP624	$0.37	1	$0.37
Op-Amp (+/-5V)	https://www.mouser.com/ProductDetail/onsemi/NCV33204DR2G?qs=8sC	$1.63	1	$1.63
Digital Potentiometer	https://www.mouser.com/ProductDetail/700-MAX5483EUD	$6.54	6	$39.24
Digital Synthesis IC	https://www.digikey.com/en/products/detail/analog-devices-inc/AD9833WI	$12.69	4	$50.76
5V to -5V	https://www.digikey.com/en/products/detail/analog-devices-inc-maxim-inte	$3.86	1	$3.86
PNP	https://www.digikey.com/en/products/detail/onsemi/SBCP53-10T1G/3062	$0.42	1	$0.42
NPN	https://www.digikey.com/en/products/detail/onsemi/MJD44H11G/1481793	$1.19	5	$5.95
Power Supply	https://www.mouser.com/ProductDetail/490-VGS-15W-48	$11.81	2	$23.62
Arduino	https://www.amazon.com/Arduino-A000066-ARDUINO-UNO-R3/dp/B008	$28.50	1	$28.50
Ethernet Adapter	https://www.amazon.com/dp/B00WX1NRO0?psc=1&ref=ppx_yo2ov_dt_b	$8.99	1	$8.99
PCB	N/A	$10.49	1	$10.49
	Passive			
Header Pins	https://www.mouser.com/ProductDetail/910-HDR100IMP40MGVTH	$0.58	3	$1.74
0.1uF Cap 0603	https://www.digikey.com/en/products/detail/samsung-electro-mechanics/C	$0.02	20	$0.38
10uF Cap 0603	https://www.digikey.com/en/products/detail/samsung-electro-mechanics/C	$0.11	6	$0.66
2.2uF Cap 0603	https://www.digikey.com/en/products/detail/murata-electronics/GRM188R	$0.20	11	$2.16
196 Ohm Res 0603	https://www.digikey.com/en/products/detail/stackpole-electronics-inc/RMC	$0.10	1	$0.10
10K Res 0603	https://www.digikey.com/en/products/detail/stackpole-electronics-inc/RNC	$0.10	4	$0.40
1.78K Res 0603	https://www.digikey.com/en/products/detail/stackpole-electronics-inc/RMC	$0.10	3	$0.30
100K Res 0603	https://www.digikey.com/en/products/detail/yageo/RC0603FR-07100KL/7	$0.10	2	$0.20
1K Res 0603	https://www.digikey.com/en/products/detail/rohm-semiconductor/ESR03E	$0.15	6	$0.90
Ethernet Screw Terminals	https://www.digikey.com/en/products/detail/cui-devices/TB001-500-07BE/	$1.13	1	$1.13
Power Supply Screw Terminal	https://www.digikey.com/en/products/detail/cui-devices/TB003-500-P03BE	$0.75	1	$0.75
Output Screw Terminal	https://www.digikey.com/en/products/detail/cui-devices/TB003-500-P05BE	$1.11	1	$1.11
Power Entry Module	https://www.mouser.com/ProductDetail/Delta-Electronics/SK-1000?qs=Tx	$2.65	1	$2.65
Fuse	https://www.mouser.com/ProductDetail/Bel-Fuse/5MF-1-R?qs=MvPYbBW	$0.60	2	$1.20
3D Printer Filament	https://www.prusa3d.com/product/prusament-petg-orange-for-ppe-1kg-2/	$10.80	1	$10.80
			Grand Total	$245.23

Figure 44: Cost Breakdown

Company	Shipping Cost
Digikey	$6.99
JLCPCB	$38.45
Mouser	$7.99

Table 1: Cost per Distributor

9 Schedule - Christian

Below depicts different charts showcasing the velocity of our tasks on this project throughout the length of construction. Below we display GANNT chart, burnup/burndown charts, cumulative flows, and velocity.

9.1 GANTT Chart - Christian

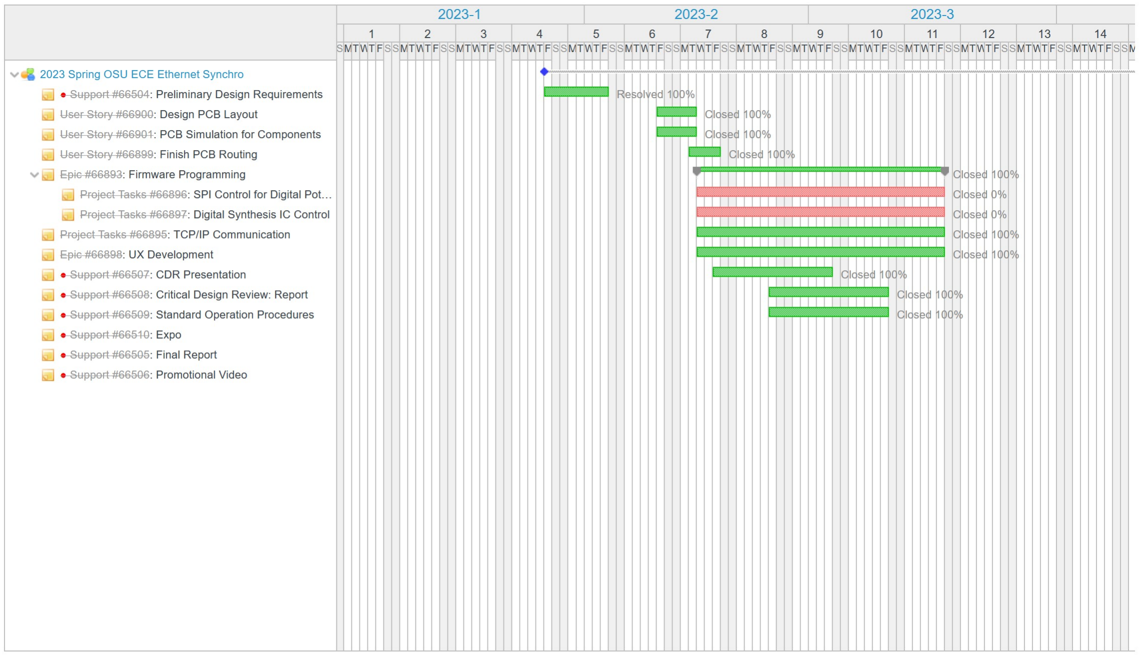

2023-1
2023-2
2023-3
1 2 3 4 5 6 7 8 9 10 11 12 13 14
2023 Spring OSU ECE Ethernet Synchro
~~Support #66504~~: Preliminary Design Requirements
~~User Story #66900~~: Design PCB Layout
~~User Story #66901~~: PCB Simulation for Components
~~User Story #66899~~: Finish PCB Routing
~~Epic #66893~~: Firmware Programming
~~Project Tasks #66896~~: SPI Control for Digital Pot...
~~Project Tasks #66897~~: Digital Synthesis IC Control
~~Project Tasks #66895~~: TCP/IP Communication
~~Epic #66898~~: UX Development
~~Support #66507~~: CDR Presentation
~~Support #66508~~: Critical Design Review: Report
~~Support #66509~~: Standard Operation Procedures
~~Support #66510~~: Expo
~~Support #66505~~: Final Report
~~Support #66506~~: Promotional Video
Resolved 100%
Closed 100%
Closed 100%
Closed 100%
Closed 100%
Closed 0%
Closed 0%
Closed 100%
Closed 100%
Closed 100%
Closed 100%
Closed 100%

Figure 45: GAANT Chart

9.2 Cumulative Flow of Stories and Features - Christian

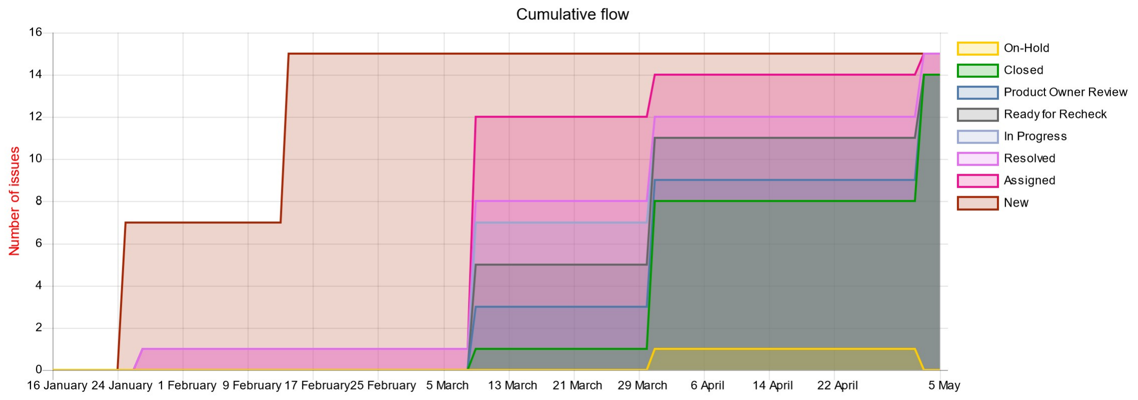


Figure 46: Cumulative Flow of Stories and Features

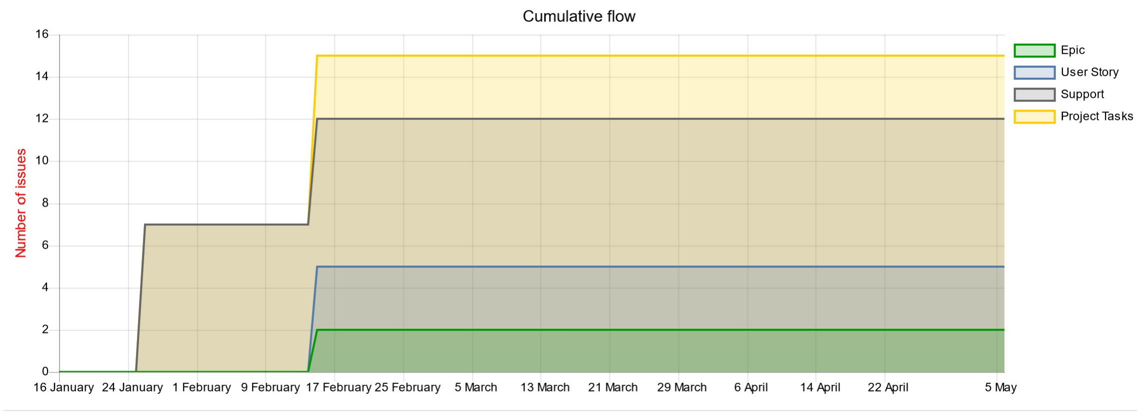


Figure 47: Cumulative Flow of Stories and Features

9.3 Burnup and Burndown Charts - Christian

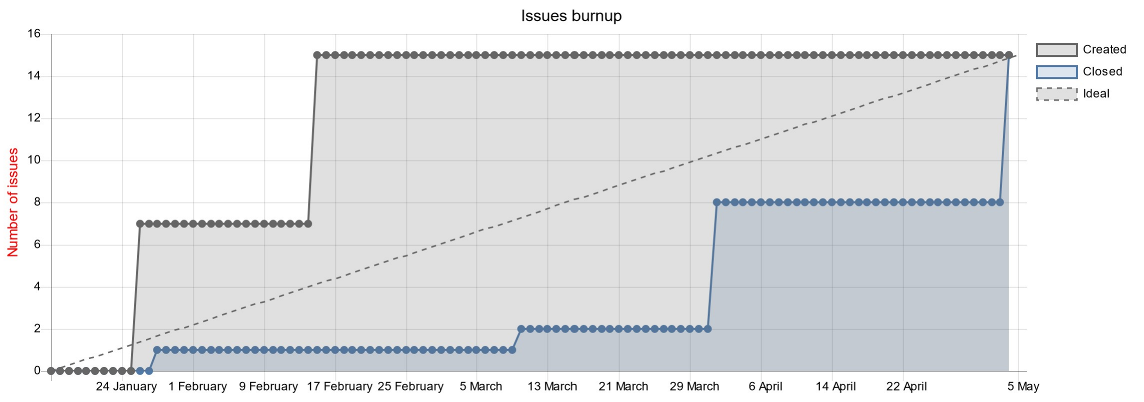


Figure 48: Burnup

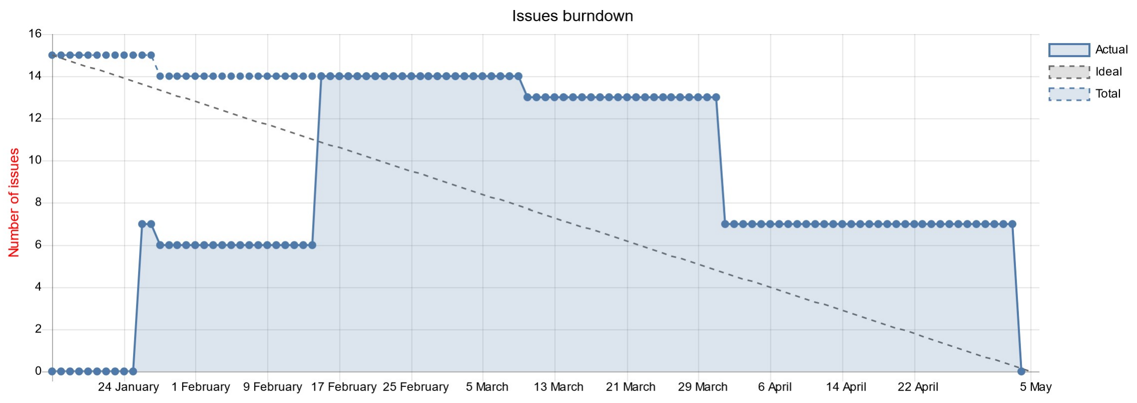


Figure 49: Burndown

9.4 Velocity and Lead/Cycle Time - Christian

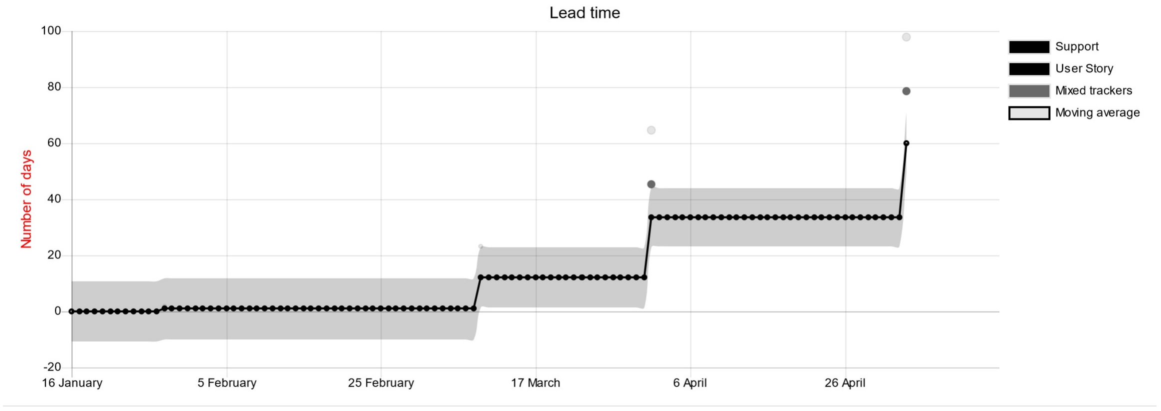


Figure 50: Cycle and Lead time

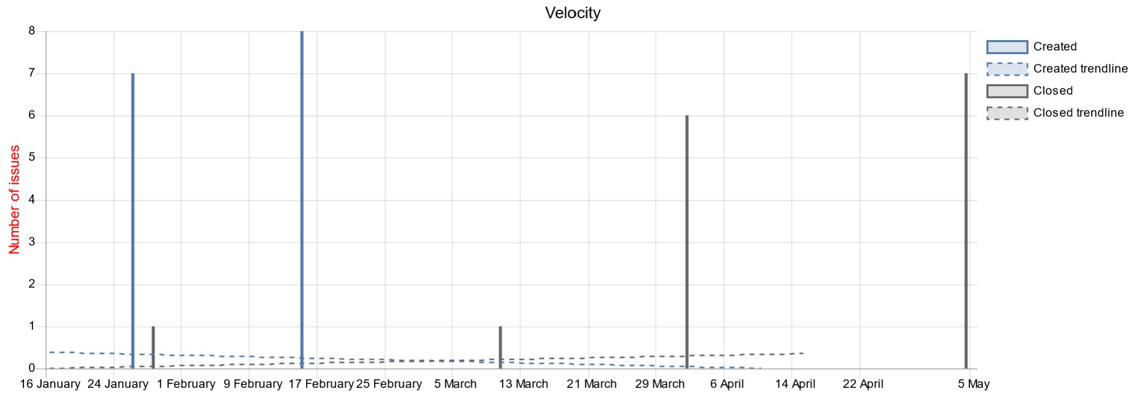


Figure 51: Velocity

10 Risk Management - Dylan

10.1 Delivery

10.1.1 Scheduling

Since everyone on our team has a busy, varying schedule it is a challenge to find times where we can all meet. To mitigate this issue we created a When2Meet poll and flexed our schedules to allow for ample time for working. We hold regular meetings with and without our advisor everyday of the week except for Thursday at varying times. We have found this allows us enough time to get ahead of the schedule we created and get all of the necessary tasks completed.

10.1.2 Online Ordering

One major issue we have run into is the increasing shipping delays for parts ordered through Mouser and Digikey. We have had some parts take as long as two weeks just to arrive, which puts a strain on our schedule. To mitigate this issue we have been trying to order our parts as early as possible so just in case there is a shipping delay we won't be behind schedule. Shipping delays are inevitable and we are trying our best to not let it affect the final result.

10.1.3 Payment System

Another issue we came across is the payment system for part orders through Cymstar. Cymstar has advised us that payment for parts orders can take as long as a week, which would not be ideal for our tight schedule. To mitigate this issue, we are ordering all parts on personal credit cards and then submitting an invoice to be reimbursed at a later date. This has worked up until this point as we are able to order parts whenever we need to instead of having to wait on payment first.

11 Improvements - All

11.1 Overall

The main issue we faced overall was shipping delays. We were trying to be frugal with the budget so we cold order multiple devices and wouldn't waste anything but since we had shipping delays it hurt the project. In the future ordering parts sooner rather than later will help ensure the project doesn't run out of time.

Another issue was documentation of the device for testing purposes. When you have two teams working on different things it's hard for everyone to know everything about a device. While we did a good job with documenting there was a miscommunication which lead to the demise of our digital potentiometers. There can never be such thing as too much documentation!

11.2 Hardware

We're still facing a clipping issue with the digital potentiometers. While we tested everything on breakout board prior to ordering the PCB's we never got a chance to test everything integrated due to lack of parts. So once again if we were to do this again we would order more things sooner rather than later.

Other than the clipping issues I don't think we would change much more on the hardware it performed exactly how we expected it to. One thing that could still be improved is the reduction of noise in the signals as that was a major issue in our design.

11.3 Software/Firmware

We would've liked to have had more time to work on the firmware part of the project to solidify the SPI and make the GUI look better. Everything works besides the angle calculation for the SPI code on the Arduino. We think the problem is just a misplaced variable in the function or a calculation wasn't typed correctly in one of the formulas. If we had more time we would've decoded the problem and would have everything functioning on the software and firmware side.