ECE 493 Senior Advanced Design Project

Arbitrary Waveform Generator
Final Project Report

Team members: James Schaeffler
German Kuznetsov
Hussain Zainal
Vikram Arunachalam
Yifei Gao
William Denham

Faculty Advisor: Dr. Jens-Peter Kaps

ECE 493
Date of Submission: December 4, 2023

Contents:

1. Executive Summary

This project involves the design and construction of a USB Waveform generator device that meets the academic requirements of Electrical and Computer Engineering undergraduate students. The primary purpose of this is to design an affordable USB Waveform generator that can be readily available and at low cost, even in chip shortage conditions. The secondary purpose of this device should be to have high accuracy and precision. The design should have similar features to other portable electrical engineering test tools such as the ability to supply typical sine, square, and triangular waveforms; as well as an arbitrary waveform specified by the user.

The planning stages of the project occurred in ECE 492, where the team was split up into three separate teams of the front-end circuit design, microcontroller (MCU), and Graphical User Interface. Individual team members were tasked with researching and coming up with potential designs to better understand and develop a solution to their part of the project. They came up with conceptual design sketches for the front-end, graphical user interface, and the microcontroller; then compared each design against the design criteria to select the best design. Design and implementation of a beta GUI was developed in these stages, as well as the breadboarded version of the analog front-end. The implementation and testing stages occurred in ECE 493, where the individual components came together into the completed system. The testing and implementation determined the design criteria has been met and thus the problem is considered solved.

2. Problem Statement

2.1 Motivation and Identification of Need

For many years, undergraduate Electrical and Computer Engineering students have been required to purchase all-in-one oscilloscope tools such as the Analog Discovery 2 (AD2) and the Advanced Active Learning Module (ADALM 2000) to complete laboratory experiments. This was due to the inability to access state of the art bench equipment in the lab, during the COVID-19 pandemic. These devices allow students to conduct lab experiments anywhere, without the need to rely on the availability of the hardware labs. The pandemic has led to a resurgence in demand for these devices as students around the world are trying to get one for their academic studies. As a result of this increased demand, there is a chip shortage and prices continue to rise as chip supply fails to meet the demand.

Tools such as the ADALM2000 and Analog Discovery 2 cost around \$236 and \$399 respectively, which are not affordable to the average college student. In order to reduce the amount required to purchase one of these tools, this project aims to create an affordable usb waveform generator. This project contains a USB communication port with a two channel, 12-bit resolution Digital-Analog converter. The sampling rate operates at 2 MHz and the sampling rate is around 5 MSPS. This device will come with a custom design Graphical User Interface (GUI) where students can supply pre-programmed waveforms as well as user-specified waveforms to circuits. This new device will pair with the oscilloscope being designed and developed by another senior design team to create a new multifunction laboratory tool for use by undergraduate Electrical and Computer Engineering students.

2.2 Market Review

Device	Device Picture	Channels	Resolution (bit)	Sample Rate (MS/s)	Voltage Range (V)	Buffer Size (Sample/Channel)	Price ($)
Analog Discovery 2 (AD2)		2	14	100	5	16384	399
Advanced Active Learning Module (ADALM2000)		2	12	75	5	64k	236.25
Dataman 531 Arbitrary Waveform Generator		1	12	100	4.5	16384	695
Handtek2000 Handheld Oscilloscope + Waveform Generator + Multimeter		2	8	250	5	6k	174
PicoScope 2204A Benchtop Oscilloscope		2	12	100	2	4k	165

Table 1: Current Alternative Waveform Generators and Oscilloscope Multifunction Tools

3. Approach:

3.1 Problem Analysis

The problem required the creation of a low cost AWG able to communicate with a GUI program running on the PC, and generate the waveforms requested by the user.

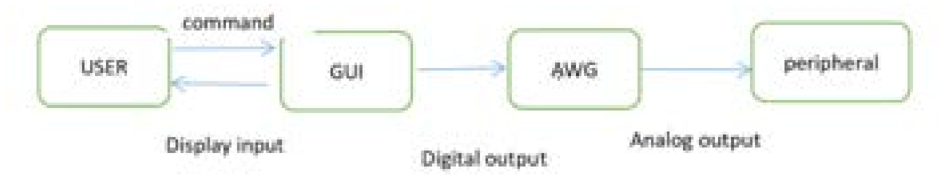


Diagram 1 : Problem Analysis Flowchart

The problem required the usage of DACs to create the waveforms, as well as op-amps to condition the waveforms to the correct output voltage levels. Furthermore, this problem required two pieces of software that can communicate via USB - the GUI software running on the PC, and the MCU firmware responsible for running the device and generating the waves. Also required is a pair of regulators to generate a dual polarity voltage rail for the op-amps.

3.2 Our Approach

One way to reduce cost is to reduce complexity. By selecting a MCU with built in peripherals like DACs and USB, we can reduce the cost of our device. However, the MCU should still be powerful enough so that the AWG remains a viable option when it comes to being used in an academic setting.

3.2.1 Hardware Approach

We settled on the STM32-F303RE MCU as our microcontroller of use. Despite its price tag of 10$, the STM32 includes two DACs, DMA, and USB, meaning it is very well suited for our use. The two DACs can be used to produce the two waveforms channels, and the DMA can be used to supply the DACs with data without load on the CPU. The USB interface could allow the MCU to communicate with a computer without the need for an external USB<->UART converter, which is needed for microcontrollers like the AVR (used in Arduino).

The MCU can only generate a wave in the 0-3.3V range. In order to condition the signal to the right levels, we will need to use op-amps to apply a gain to the signal and shift the signal to the right levels. By carefully designing the circuit, we can minimize parts count and keep cost down.

To supply the aforementioned op-amps, we will need to generate a positive and negative voltage rail to power the op-amps. This will require a boost and a buck converter, to generate the positive and negative voltage rails respectively. Considering that our output signal should be in the -5 to 5V range, and that typical (non rail-to-rail) op-amps cannot reach their supply voltage, the negative supply rail should be less than -5V and the positive supply rail should be over 5V.

3.2.2 Graphical User Interface Approach

Our graphical user interface must be capable of allowing the user to specify the waveform they want, And transmitting these settings to the user. There are 4 main specifications for the waveform:

- Frequency of the waveform.
- Amplitude.
- Offset.
- Shape (triangle, square, sign, or user defined)

Furthermore, we could extend the functionality to allow the user to enter duty cycle for square waves, and phase when there are more than two waves. These options will be configured via graphical user interface elements such as numerical text boxes or drop down menus, and a preview of the output waveform will be displayed in the GUI. To simplify coding, we chose to use the GUI library PyQt. It can be used via python, and works with any of the major three operating systems.

3.3 Alternative Approaches

There are not a lot of alternative approaches when it comes to the opamp and power supply of the AWG hardware. However, there are some alternative MCUs to use. One example is the RP2040. It has a ARM Cortex CPU with higher clock speeds than the STM32. It is also cheaper. However, it does not come with an internal DAC and would require an external DAC. Another approach would be to use a Direct Digital Synthesis IC such as the AD9837 instead of a DAC. This approach would completely alleviate the task of generating the waveform from the MCU, allowing us to use a very anemic MCU. However, a chip like the AD9837 is expensive at $6.77, and only has one channel.

3.4 Background Knowledge:

3.4.1 Microcontroller Unit (MCU)

MCU is an intelligent semiconductor IC that consists of Arithmetic and Logic Unit(ALU), Register set, Control Unit, Internal bus, and Interface to System bus which is to connect to memory and I/O ports. The MCU is used in lots of different types of applications which include washing machines, radio, and controllers. MCU is similar but less sophisticated than System on a Chip. The first MCU was developed in 1971 which is called TMS1000. For MCU characteristics, it has a low price for high-volume applications. It has lower clock frequencies when compared to DSPS and it is up to 100MHz. Also,

Figure 1: STM32 – Nucleo – F303RE

low power consumption and limited memory. For our project, we use the NUCLEO-F091RC as our boardand the microprocessor on it is STM32F091RC. For the MCU we use, its frequency is up to 48 MHz. It has 128 to 256 Kbytes of Flash memory. 32K bytes of SRAM with HW parity. It has a 12-channel DMA controller. One 12-bit, 1.0us ADC, and its conversion range is 0 to 3.6V.

3.4.2 Graphical User Interface (GUI)

We will use PyQtGraph(Figure 2) which is a GUI module to design the GUI for this project. PyQt supports both C++ and python and it is widely used for creating large-scale GUI-based programs. It provides creators with lots of different pre-built designs which helps creators save time. The QtWidgets has graphical components and related classes, such as buttons, windows, status bars, bitmaps, colors or fonts. Also, PyQt can run on Windows, Linux, Mac OS, and various UNIX platforms. For our project, on the user side, they can choose a hand drawn arbitrary waveform, sine, square, triangle and sawtooth.

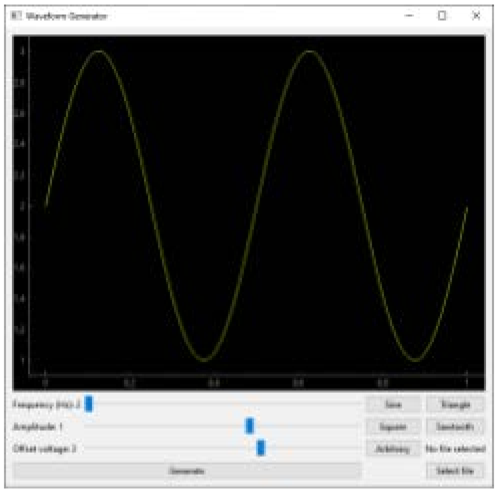


Figure 2: Prototype PyQtGraph

3.4.3 Digital to Analog Converter

A D/A converter takes the precise number and converts it into a physical quantity. Usually, the digital signal is a finite-precision time series data and the analog signal is a continually varying physical signal.

$$v_o = \frac{v_R}{(2^{N-1})} * D$$

v_o = voltage output

v_R = reference voltage

N = number of bits

D = Digital Number Input

Figure 3: DAC Output Voltage

3.4.4 Operational Amplifier (OpAmp)

An operational amplifier is an integrated circuit that can amplify electrical signals. It has 2 input pins and 1 output pin. Usually, an operational amplifier isn't used alone but connected to other circuits' components. For 1 op-amp circuit, it can be a non-inverting amplifier circuit, inverting amplifier circuit, voltage follower, etc. For this project, the circuit we design needs to let the output signal be large enough and also can drive a 20mA load.

Figure 4 : LMH6506 Variable Gain Amplifier

3.5 Project Requirements Specification:

3.5.1 Mission Requirements:

The project shall develop an affordable USB-powered arbitrary waveform generator that has a low cost requirement for undergraduate electrical and computer engineering students to perform laboratory experiments anywhere. The device shall utilize a custom PCB design for the analog front end and microcontroller that can interface to a Graphical User Interface to display the results.

3.5.2 Operational Requirements:

Input/output requirements:

- The device should have 2 analog output channels.

External Interface Requirements:

- Communication of the device is from the USB

Functional requirements

- The device **shall** create arbitrary waveforms, triangular, rectangular, sine
- The output bandwidth **should** be around 2MHz.
- The Sample Rate **will** be around 5 MSPS.
- The output **shall** be peak-to-peak 10V, adjustable +-2.5V.
- The output **will** be able to drive a 20 mA load and 12 bits resolution.
- The output **shall** display the wave, frequency, amplitude and offset selected.

3.5.3 Technology and System-Wide Requirements:

In order to run the device, the SMT32 IDE and Python based GUI **will** be able to work on most operating systems, i.e. Windows, Mac OS, and Linux.

4. System Design:

4.1 Functional Decomposition

The MCU software has a number of functions, including communication with the GUI, managing the DMA and PWM, managing the timers, and outputting the waveform. The hardware has the function of amplifying and offsetting the waveform in the analog front end, and generating a number of voltages required by the system. The GUI has the function of accepting user input, communicating with the MCU, and drawing graphs based on the user input.

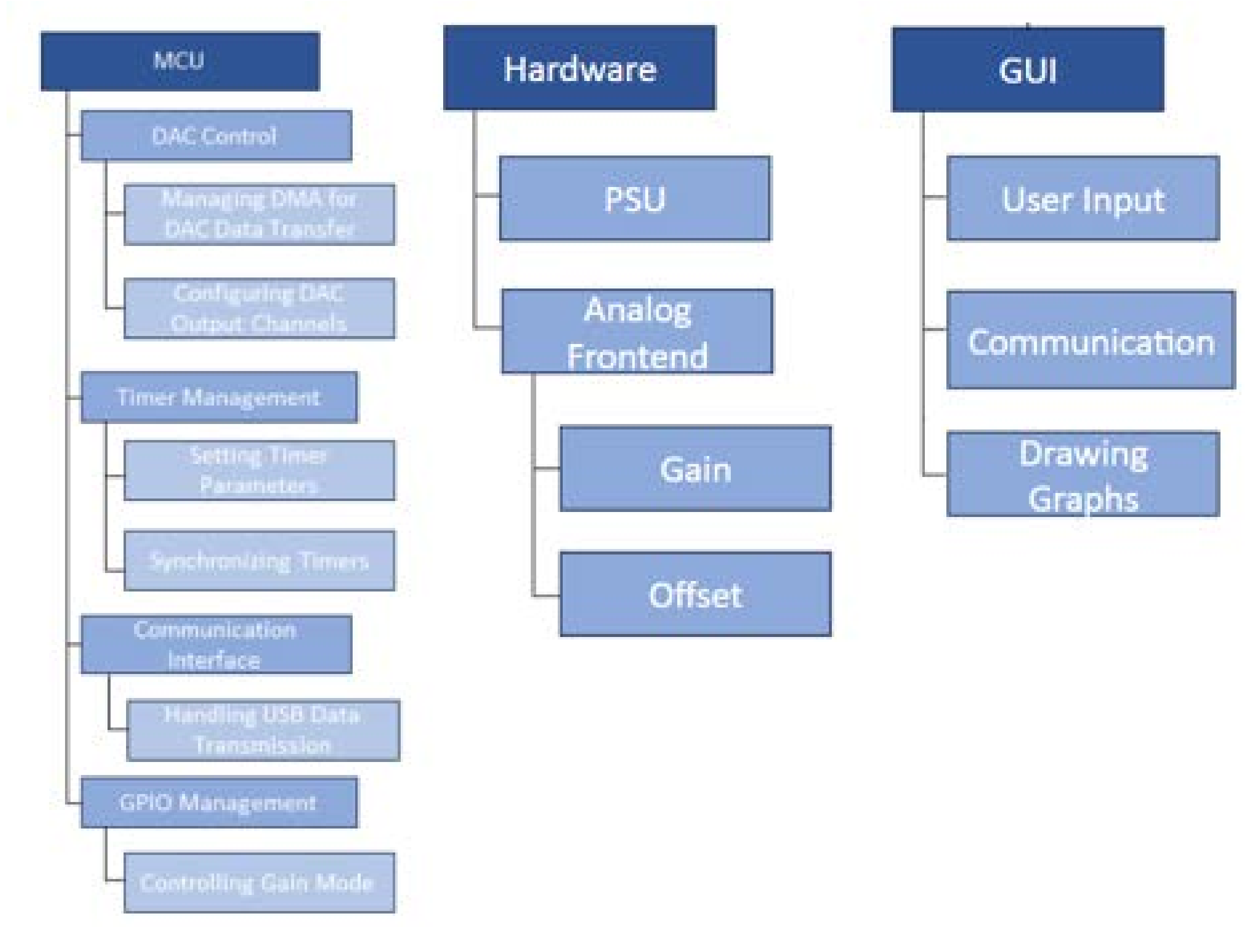


Figure 5: Functional Decomposition

4.2 Physical Architecture

The physical architecture is split into three main components. The first is the user's PC which runs the GUI application. The next component is the NUCLEO dev board which contains the STM32 MCU and the voltage regulator for the MCU. The third is our custom PCB, which contains the analog front end, itself composed of the gain and offset stages, and the power supplies. A USB Cable links the PC to the MCU.

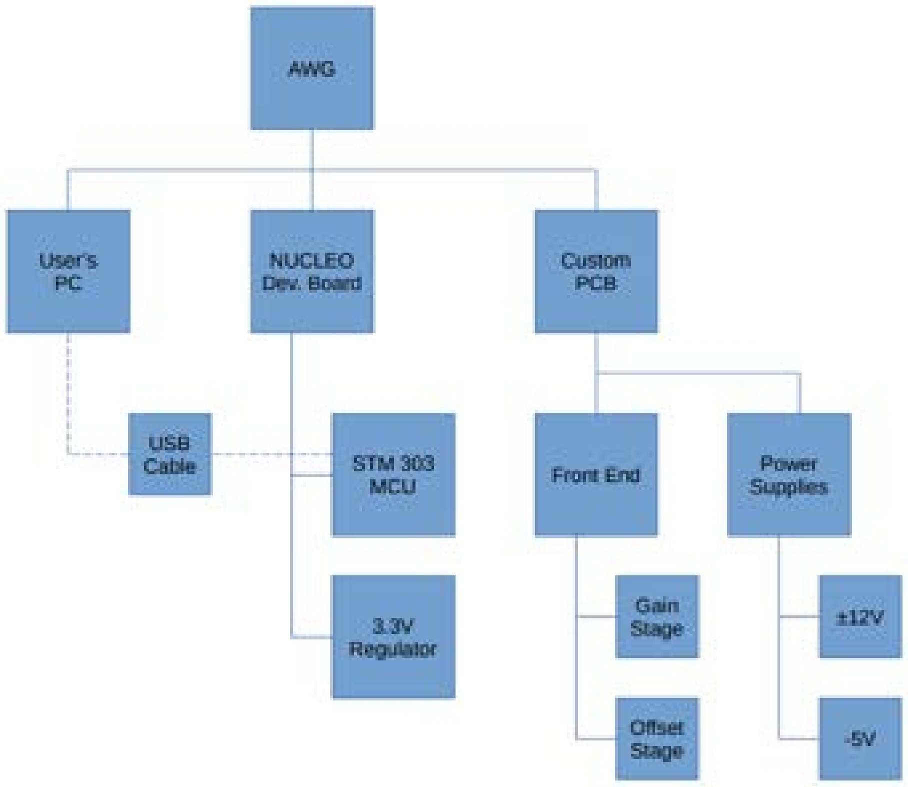


Figure 6: Physical Architecture

4.3 System Architecture

The system is composed of the GUI software, which runs on the user's PC, the MCU software, which runs on the STM MCU, and the AWG hardware, made up from the includes the NUCLEO dev board and our custom PCB. The GUI accepts input from the user. The GUI displays the user's changes and sends configuration packets to the MCU, to which the MCU responds to with acknowledgment packets. The MCU generates the wave, which passes through the AWG hardware analog front end and is output from the device. The PC which runs the GUI software supplies power to the AWG hardware.

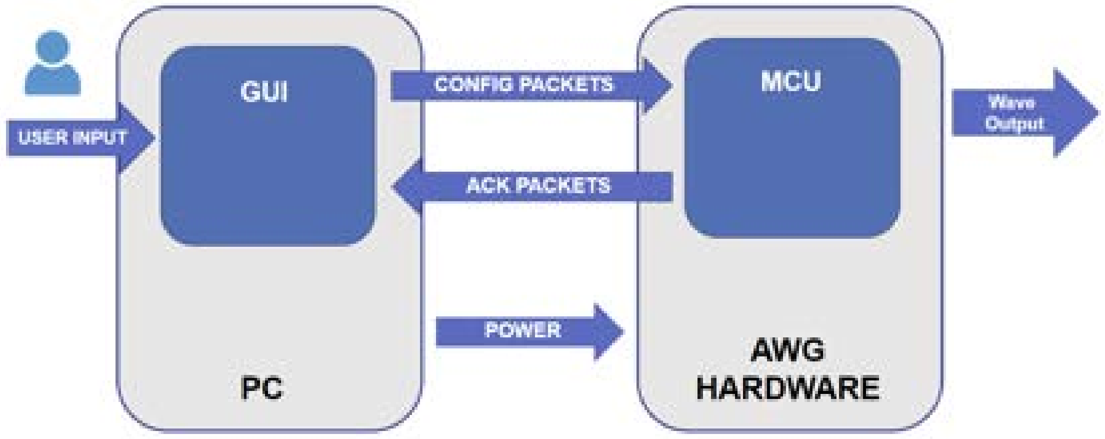


Figure 7: System Architecture

4.4 Connection

4.4.1 Connection Setup

The STM32F303 MCU has a built-in USB controller. This allows the MCU to communicate to a PC through USB without requiring external hardware like an FTDI USB-Serial converter IC. The USB protocol defines a device class commonly used to emulate serial devices called the USB communication device class. (CDC). The STM32F303 can act like a CDC Device, and send and receive data from a USB Host, such as a PC. On the PC, the MCU will appear as a serial device. An application can then open the serial port and talk to the device.Serial devices communicate using a stream of bytes. This stream is continuous, with no separation between packets. Any custom packet format has to be manually implemented by the user on top of the stream. However, usb is a packet based, not stream based, protocol. Thus, when USB CDC is used, the stream has to be split into packets, sent through the USB controllers, and reassembled back into a stream. On this USB device, the USB CDC packet can carry up to 64 bytes.

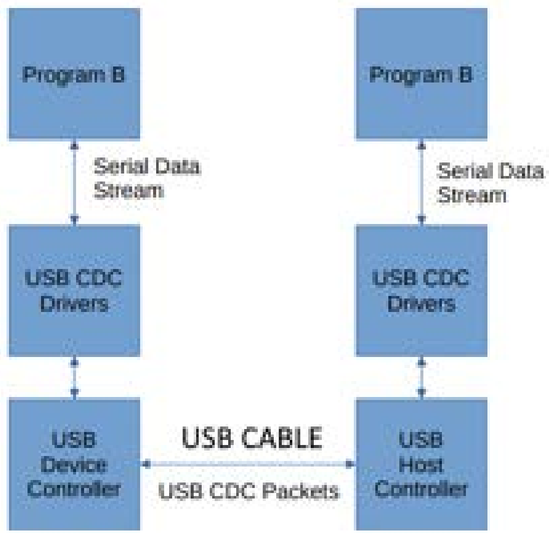


Figure 8: Connection Setup

The STM32's CDC USB device library code does not automatically convert the CDC packets back into a stream. For simplicity, we thought reassembling the USB CDC packets into a stream only to split it into our custom packet format to be redundant. Thus, we made the packet size in our custom packet format a multiple of 64 (which is the max amount of data that can be carried in a USB CDC packet) and then simply treated the data in the USB CDC packets as if they were our custom packets.

The process of sending a packet from our GUI application to the MCU is explained here:

a) The GUI creates an array of bytes representing the packets. The array size is a multiple of 64.
b) The packet bytes are sent through the socket. The OS's drivers take the stream and split it into USB CDC packets. Since at any time the amount of data in the stream should be a multiple of 64, the CDC packets created should always contain 64 bytes of data.

c) The packets are sent through the USB cable via the controllers.
d) The MCU receives the CDC packet and reads the data in the packet. The MCU can now interpret this data for commands.

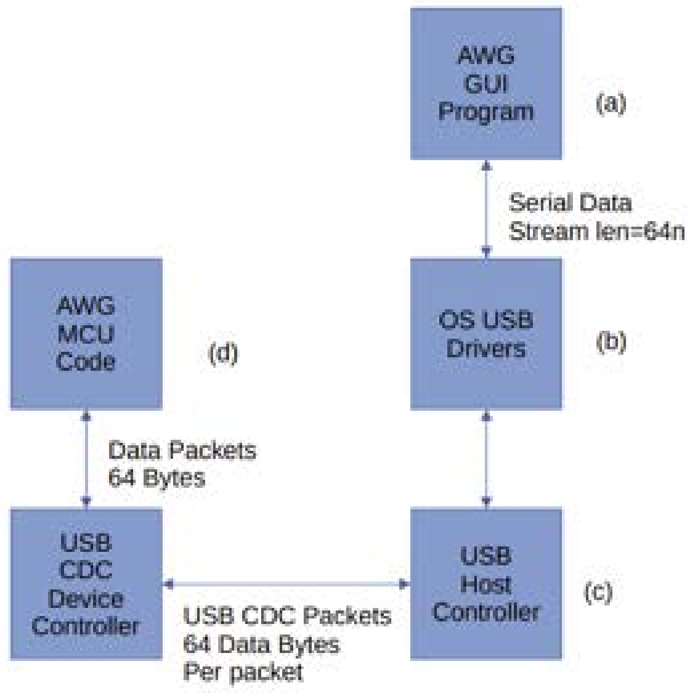


Figure 9: MCU-GUI Communication

A possible issue with this method is that nothing prevents the OS's drivers from splitting the stream into CDC packets with less than 64 bytes, however in practice, we have not seen that happen. The process for sending data from the MCU to the GUI works much the same way but in reverse

4.4.2 Packet Format

Now that the connection setup has been discussed, we move onto our custom packet format used to control the AWG device from the GUI. There are three packet types in our format. The format is easily extendable to allow for more packet types. All packets use a little-endian format. The three packet types are:

a) handshake/keep-alive (GUI -> MCU)
b) Config (GUI -> MCU)
c) ACK (MCU -> GUI)

The first packet type is the "handshake/keep-alive" packet. It is sent from the GUI to the MCU to initiate a handshake between the two. It is also sent occasionally by the GUI as a form of "keep-alive", but this is not necessary as USB already implements keep-alive and any disconnect would already be detectable. Upon receiving this packet, the MCU is expected to send an "ACK" (acknowledgment) packet. If the MCU does not respond within a given time period, or responds with an incorrectly formatted ACK packet, the GUI software knows the device has malfunctioned and attempts no further communication. The format for the "handshake/keep-alive" packet is shown in table 2:

Handshake/Keepalive Packet					
Offset	Name	Type	Value	Length	Note
0	Packet Type	u8	0	1	Indicates packet type
1	Magic String	u8[]	"INIT"	4	Magic string, no null terminator.
5	Padding	u8[]	{0, 0, …}	59	Zeroed Padding Bytes

Table 2: Handshake Packet Format

The "ACK" packet, which is shown in table 3, is sent from the MCU to the GUI is equally simple. Both send no practical data other than a short magic string that can be checked for correctness. Note how both packets are padded with zero bytes to increase the length up to 64 bytes.

Ack Packet					
Offset	Name	Type	Value	Length	Note
0	Packet Type	u8	0	1	Indicates packet type
1	Magic String	u8[]	"STMAWG2 3"	8	Magic string, no null terminator.
9	Padding	u8[]	{0, 0, …}	55	Zeroed Padding Bytes

Table 3: Acknowledgement Packet Format

The config (configuration) packet, shown in table 4, is more complex. It is sent by the GUI to the MCU everytime the GUI wants to configure the output wave. The first data field of the packet is the channel that the GUI wishes to configure, this field can have a value of 0 or 1. All other data fields contain settings that will be applied to that channel. The second field is the gain, its value is 0 to indicate high gain (10VPP output) or low gain (1VPP output). The next to fields, PSC and ARR, control the output frequency of the samples. The next field, CCR_Offset, controls the offset voltage. 0 indicates the minimum offset voltage (-5V), and 4095 indicates the maximum offset voltage (5V). NumSamples indicates the number of samples to send per period of the wave. PhaseARR is the number of clock cycles to offset the phase of the waves by. PhaseARR must not be larger than ARR, thus PhaseARR allows for precise control of the phase down to ~13.9ns, but does not allow for a large range. To get phase control with high precision and high range, the GUI must employ two methods of changing the phase: setting PhaseARR and shifting the provided samples. After the padding bytes, comes the array of samples. The length of the array (in bytes) is always numSamples*2, rounded up to the nearest multiple of 128. Each sample is 16 bit and ranges from 0 (-5V with gain=0, -1V with gain=1) to 4095 (-V with gain=0, 1V with gain=1). The GUI must scale the samples to get the correct amplitude. The packet format is shown here:

Channel Config Packet					
Offset	Name	Type	Value	Length	Note
0	Packet Type	u8	1	1	Indicates packet type
1	Channel	u8	x	1	Channel to configure, 0 or 1
2	Gain	u8	x	1	Gain to use, 0 = High, 1 = Low
3	PSC	u16	x	2	Prescaler Value for channel Timer
5	ARR	u16	x	2	Divider Value for channel Timer
7	CCR_Offset	u16	x	2	CCR Register value for PWM Offset Gen.
9	numSamples	u16	x	2	Number of Samples
11	phaseARR	u16	x	2	Phase offset of wave in clock cycles
13	Padding	u8[]	{0, 0, …}	51	Zeroed Padding Bytes
64	Samples	u16[]	x	v	The samples for the wave. Length is numSamples*2 rounded up to nearest multiple of 128

Table 4: Channel Configuration Packet Format

5. Detail Design - Analog Front End Architecture

5.1 Hardware Architecture

The primary hardware component is the STMF303 microcontroller. The microcontroller is responsible for generating the samples for the waveform, and controlling the settings of the two analog front ends, each of which is responsible for a single output channel. The analog front end circuit takes a waveform generated by the MCU and applies the desired gain and offset. The resulting waveform is fed to the output connectors.

A USB connector allows the hardware to be connected to a computer. The data pins of the connector are connected to the MCU to facilitate communication between the MCUand the user’s PC. The USB connector also supplies 5V to the board.

A number of voltages are required by the components. 3.3V is needed to power the MCU. A linear regulator is used to generate this voltage from the 5V USB voltage. The opamps need a high enough supply voltage so as to not reach saturation at the min or max output voltage. Because the min and max output voltages of the analog front ends are 10V and -10V, respectively, 12V and -12V was chosen to power the opamps. The voltages are generated by a switching mode PSU. Finally, a -5V reference voltage is needed by the analog front end for correct offsetting of the wave. This reference voltage is generated via a zener-like voltage reference IC.

This project - the waveform generator - is intended to be one of three projects that make up the final product, all three of which share the MCU. Due to this, a NUCLEO MCU development board will be used as the base of the hardware, and all other necessary functionality will be built on a “hat” daughterboard that can plug onto the connector found on the NUCLEO development board. This will allow the other two projects to be combined into one product by simply plugging their corresponding daughterboards onto the stack. The development contains the MCU as well as the 3.3V linear regulator.

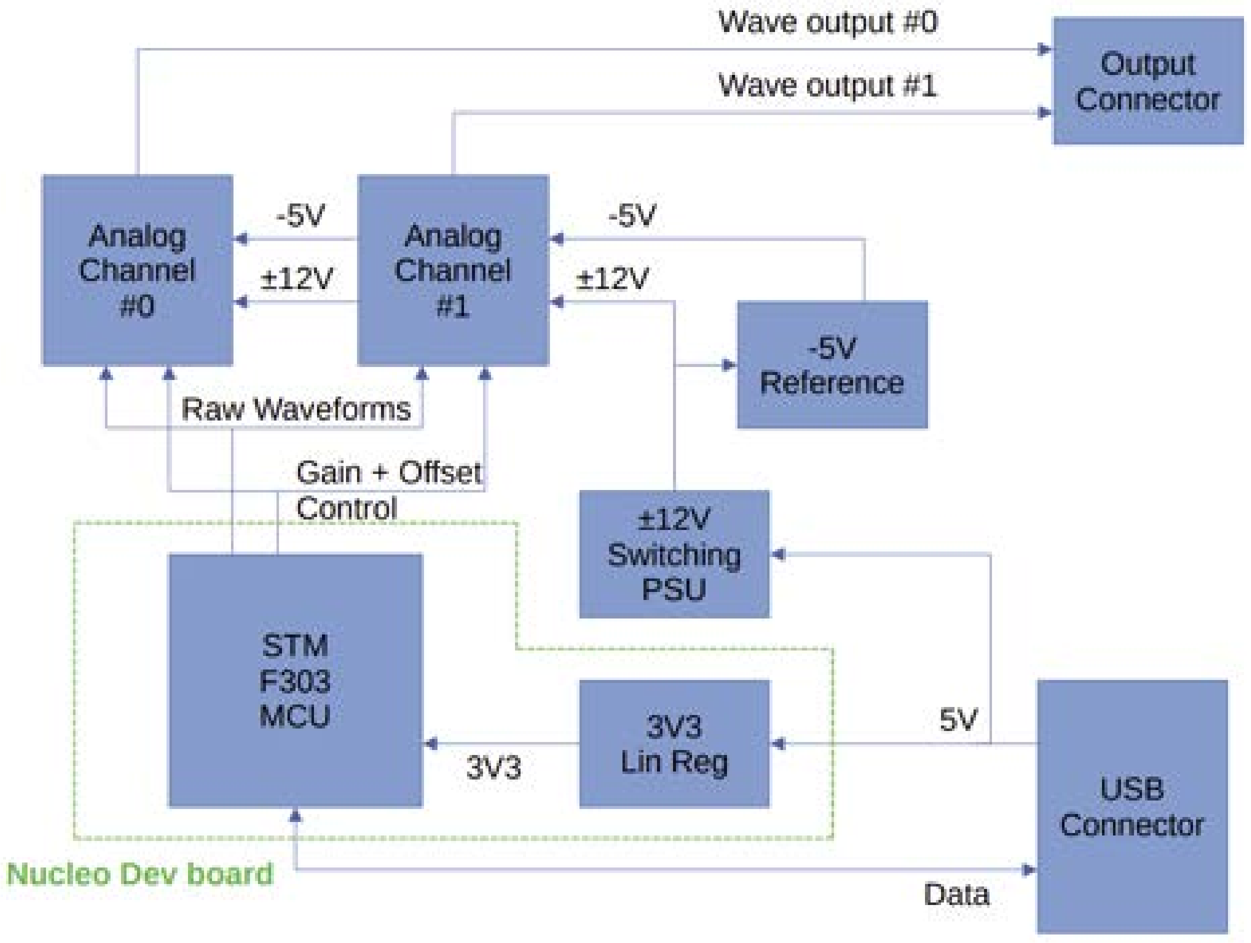


Figure 10: Analog Front End System Architecture

5.2 Analog Channel Architecture

The analog channel, of which there are two on the board, has two primary jobs. The first is to apply a gain to the waveform generated by the MCU, as the MCU DAC's output is only 3.3V peak-to-peak. And second, to correctly offset the wave. The diagram for the analog channel is shown in figure N.

The wave enters the analog front end from the MCU. It is 3.3V peak-to-peak and centered around 1.65V. Therefore it is offset by -1.65V (fig. 11a) to center it around zero before applying the gain (11b). The two possible gains are *10÷3.3* and *1÷3.3*. (3.0303… and 0.3030… respectively.) The MCU can select which gain to use (11c). These gains are fractions to turn a 3.3V peak-to-peak wave into either a 10V peak-to-peak wave (the "high gain" option) or a 1V peak-to-peak wave (the "low gain" option). The offset is configured by a PWM signal generated by the MCU. It enters a low-pass filter (11d) which turns the PWM signal into a DC signal. The DC offset signal is amplified (11e) and is combined with the waveform (11f). Because the PWM signal is centered around 1.65V, it also needs to be shifted by -1.65V in (11f).

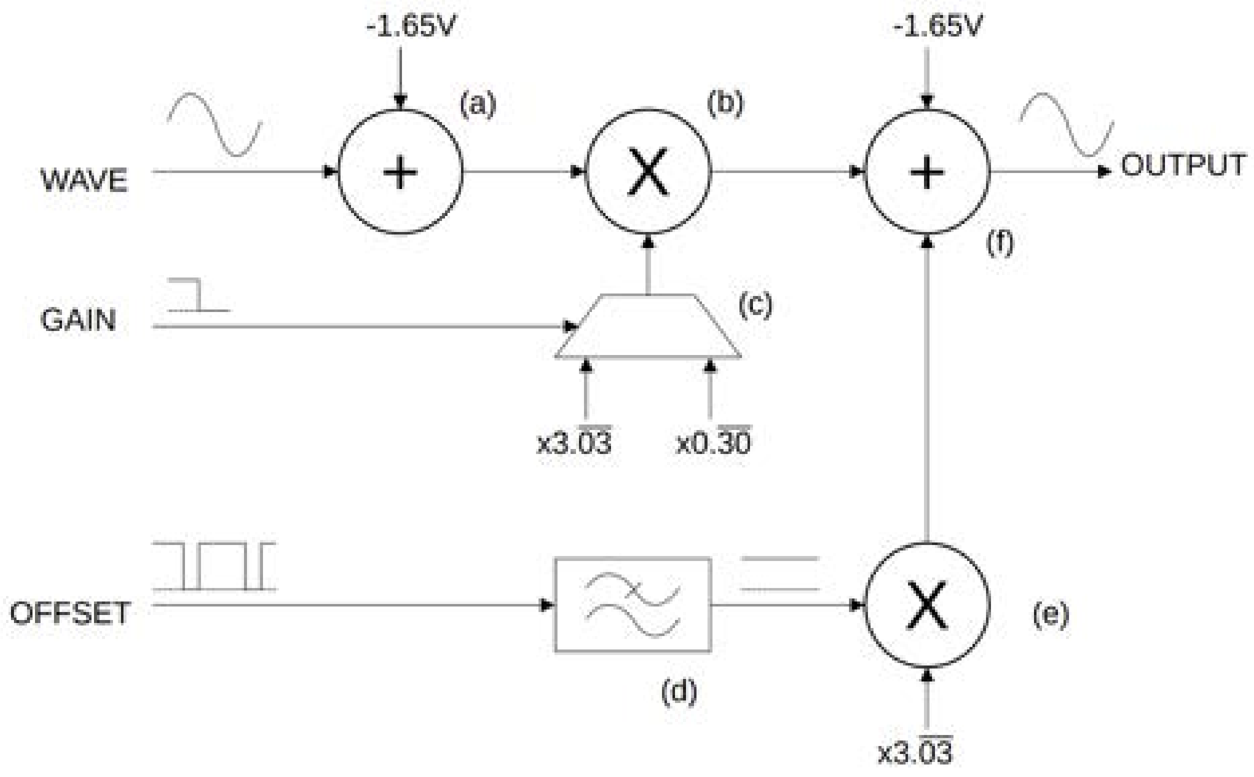


Figure 11: Analog Channel System Architecture

5.3 Schematic Design - Analog front end

The analog front end design was prototyped using Falstad circuit simulator to work out a compact design that used few components. Then the circuit was carried over to schematics in Kicad. Stages (11a)(11b) and (11c) are performed using an inverting opamp configuration. To control the gain, an analog multiplexer is used to switch the resistor in the feedback of the amplifier. Here the -5V reference voltage is used for the -1.65V offset.

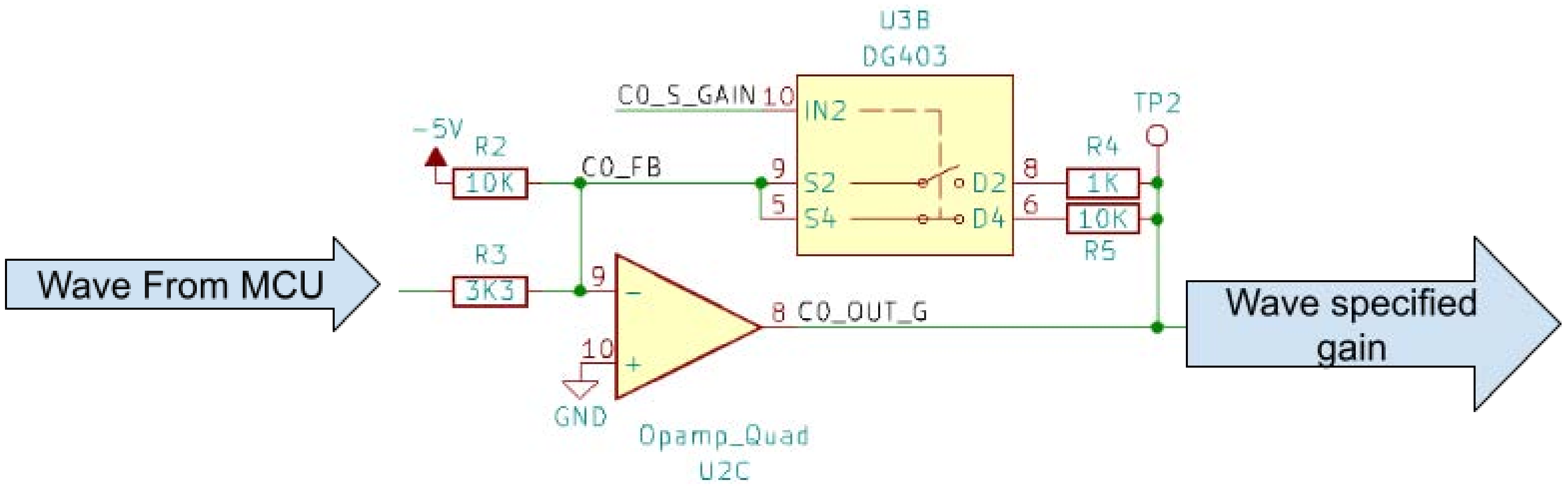


Figure 12: Schematic of MCU Supplied Wave and Gain Adjustment

Stage (11d) is done by passing the PWM offset through a RC lowpass filter, then passing the filtered signal through a opamp in a voltage follower gain configuration.

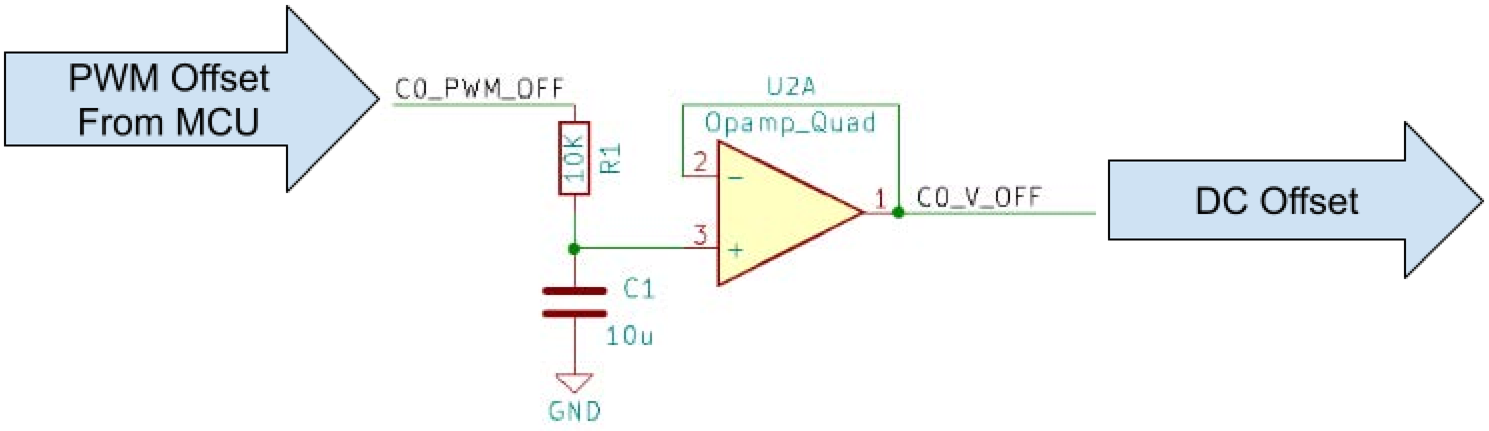


Figure 13: Schematic of PWM/DC Offset

Stage (11e) and (11f) are performed by another inverting opamp configuration. Since the front end contains two inverting opamp configurations, they will cancel out and the output signal will be the same phase as the input signal.

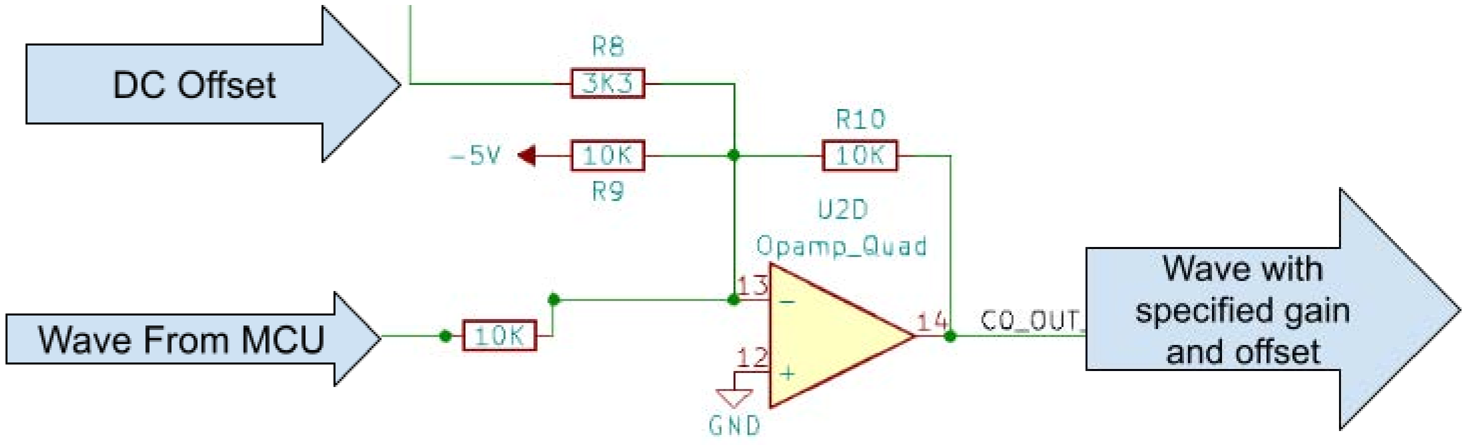


Figure 14: Schematic of PWM Offset

All together the schematic for the entire front end for a single channel is in figure where C0_PWM_OFF is the PWM offset and C0_IN is the waveform from the MCU.

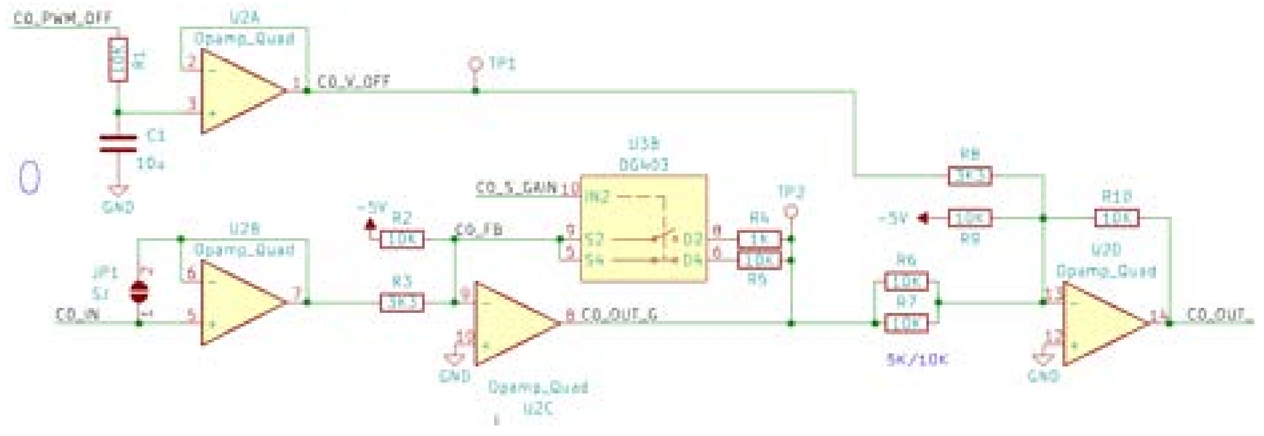


Figure 15: Complete Channel Schematic

This circuit is duplicated for the second channel. The outputs of the two analog channels are fed to a pair of BNC and a pair of male pin connectors.

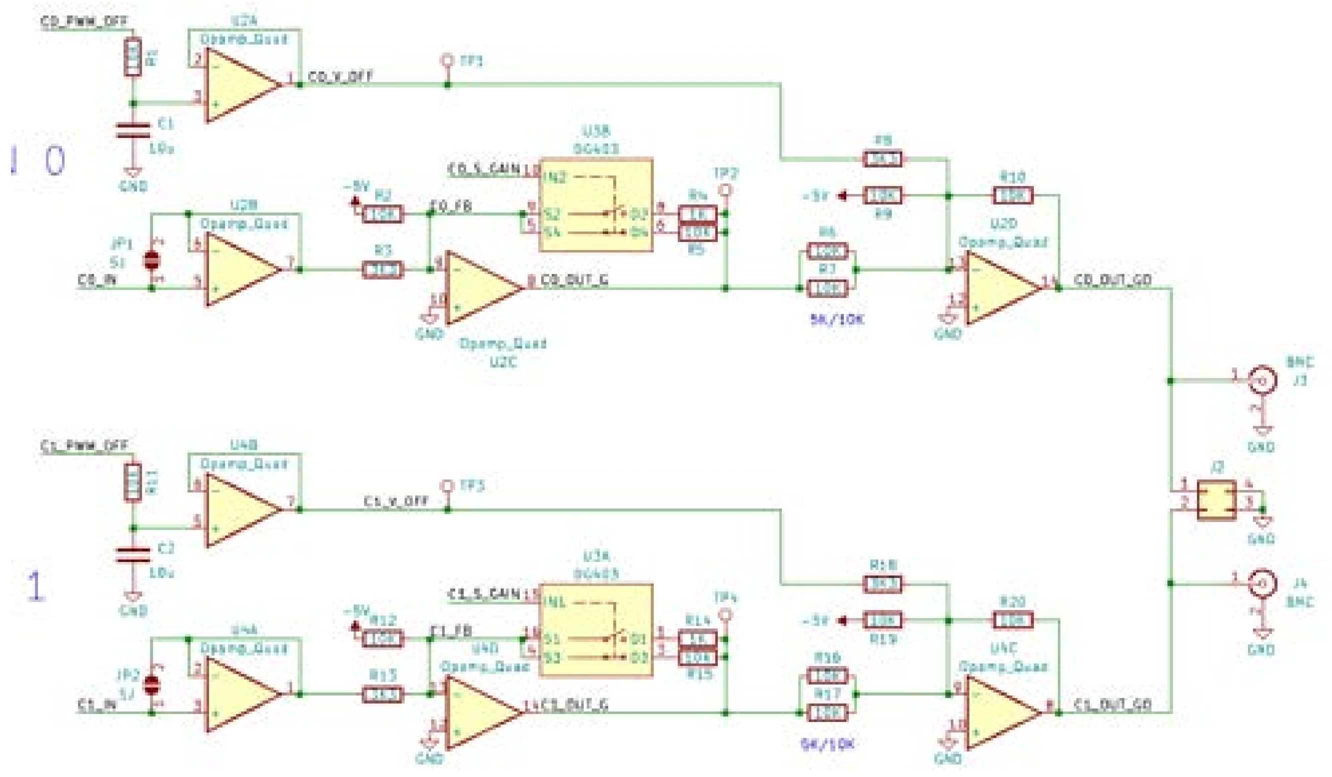


Figure 16: Two Channel Complete Schematic

5.4 Schematic Design - PSU

To generate the 12V and -12V power rails for the op amps, a circuit was designed around the R1283, a dual rail step-up/inverting DC/DC converter IC. Though it is intended for powering CCDs and LCDs in cameras, the chip fits our specifications, and was available for a low price, and was thus selected for usage. The circuit design is fairly standard. The RF/RG and RF/RR resistors set the output voltage values according to the formula provided in the datasheet. A LM4040-5 voltage reference IC is used to get a -5V reference voltage from the -12V rail. The LM4040 mimics the behavior of a zener diode, and can be used in the same circuit setup as a zener diode. However, the LM4040 actually contains analog circuitry that allows a more stable output over a wide range of current draws, whereas a zener diode's forward voltage would vary with the current drawn from the reference and would thus probably require buffering.

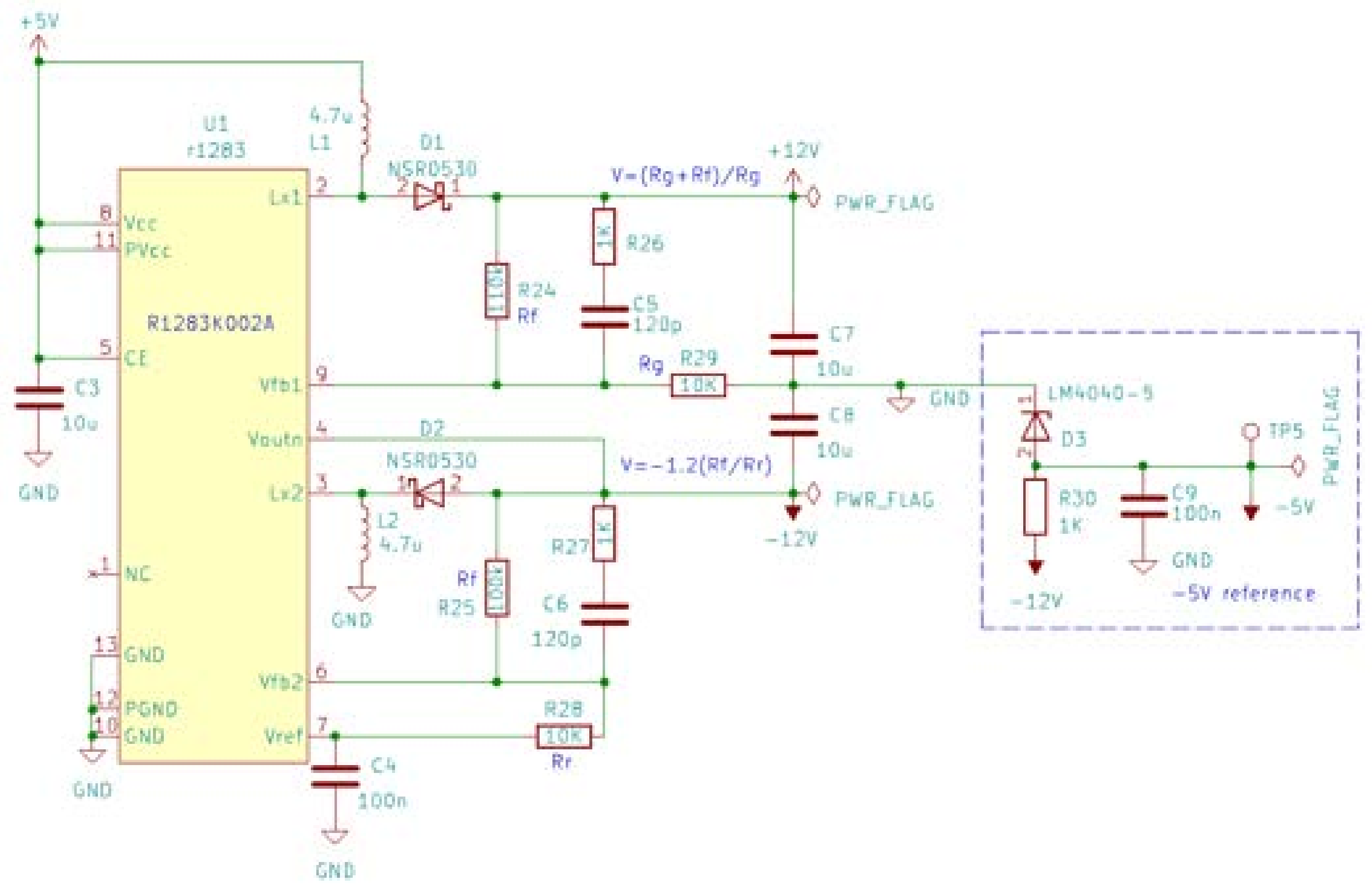


Figure 17: Power Supply Unit Schematic

5.5 Schematic Design - Connectors

Four rows of through-hole headers are used to connect the daughterboard to the NUCLEO development board. The USB connector also sits on the daughterboard and is connected to the corresponding. As per specifications, the two differential data lines have 22 ohm series resistors, and a 1.5K ohm pullup resistor on the positive data line. It should be noted that the pins used for offset were changed on the MCU after the PCB was made, so this schematic is slightly wrong. The error had to be manually bodged on the PCB with wires.

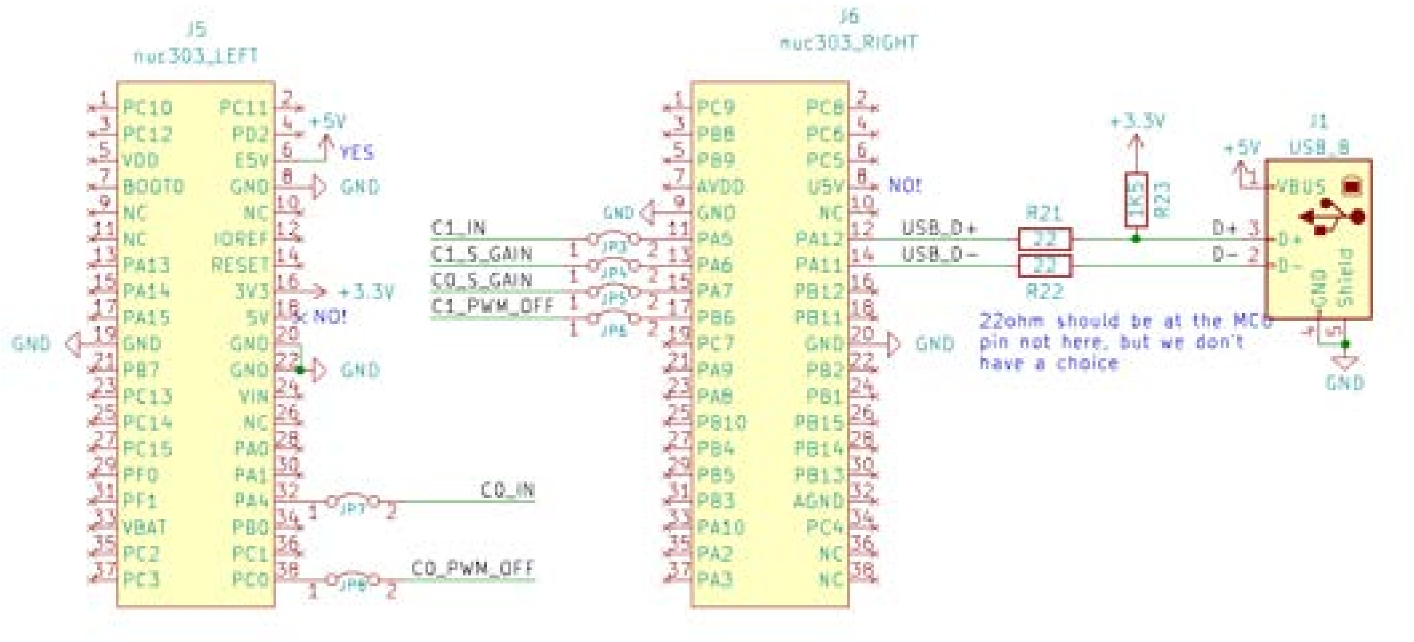


Figure 18: Connector Schematic

5.6 Layout Design

The schematics were transferred over to a PCB layout. A width of 0.25mm was used for most traces, but the power traces were thicker, at 0.4mm. The USB data traces were routed as a length matched differential pair. The PCB was routed as a two layer board, with most horizontal traces the the first layer and most vertical traces on the second layer. Because of the two layer layout, having a continuous ground plane was not possible. Instead there is a segmented ground plane stitched together with vias. A “final” version should use a 4 layer layout with a typical signal-ground-ground-signal stackup.

When doing PCB layout, cross coupling between traces is a concern. Analog traces should be kept away from other analog traces to prevent cross coupling between the channels. The USB traces, which contain sharp edges, can easily couple to other traces and should also be kept away from the analog traces. Unfortunately, due to the use of the development board, whose layout is beyond our control, the USB traces couple to the trace for the second channel’s DAC output. This causes noticeable interference on the second channel output when a packet is sent to the device. Similarly, the traces carrying the offset PWM signal should be kept away from the analog or USB traces too.

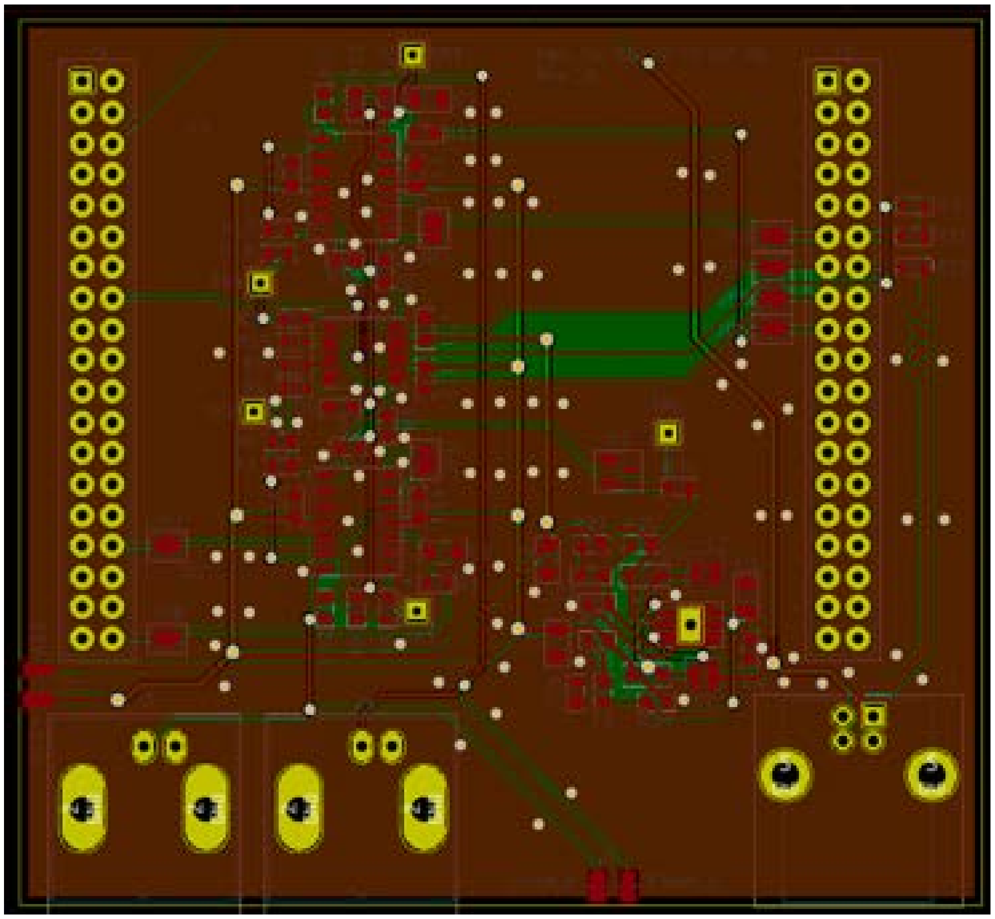


Figure 19: PCB Layout

5.7 PCB Assembly Method

The PCB was manufactured by JLCPCB and manually assembled by us. The PCB assembly was done in 4 steps. The first step is to place solder paste on the pads on the PCB. This could either be done manually, using a syringe of solder paste, or by placing a stencil over the board and spreading the paste over the stencil. The stencil is way more effortless, but has to be milled by a CND, which is done at the manufacturer, and thus increases development cost slightly (~7$).

The second step was to populate the components onto the board. The components were placed on the board with tweezers under a microscope. The third step was reflow soldering the board. The board was placed in a reflow oven, which melts the solder, soldering the components to the board once the oven cools.

In order for this process to work flawlessly, placement and quantity of solder needs to be controlled perfectly (a stencil helps with this), and the components have to be placed with tight tolerances. This is hard to do by hand, which can cause problems with the board such as tombstoned resistors/capacitors or shorted pads. The final step of the assembly process is to fix up any issues with the board.

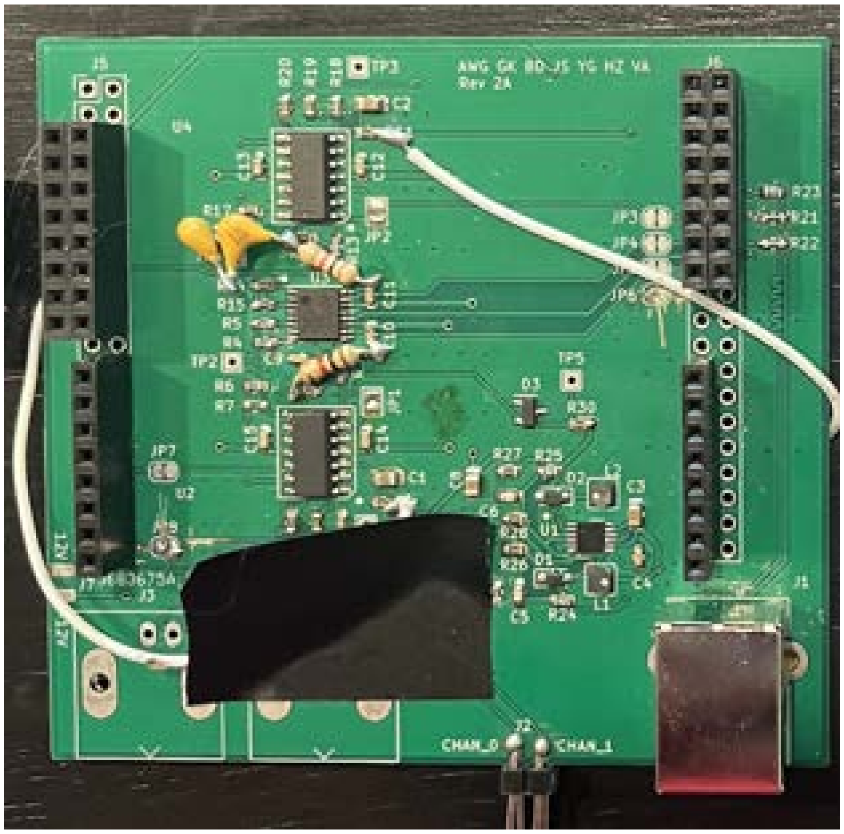


Figure 20: Final Version of the PCB

5.8 NUCLEO-F303RE Modifications

The NUCLEO development board is intended for a variety of use cases and thus contains solder bridges that should be shorted/cut as needed for your use case. First, X3 on the development board has to be populated with a 8MHZ crystal. Then, C33 and C34 have to be populated with 20pf capacitors, and R35 and R37 have to be bridged. Finally jumpers SB54, SB55, SB16, and SB50, SB21 have to be cut. These modifications allow the MCU to run off a 8MHZ crystal, which is needed for the MCU's UCB peripheral work, and allow the second DAC output channel to work correctly.

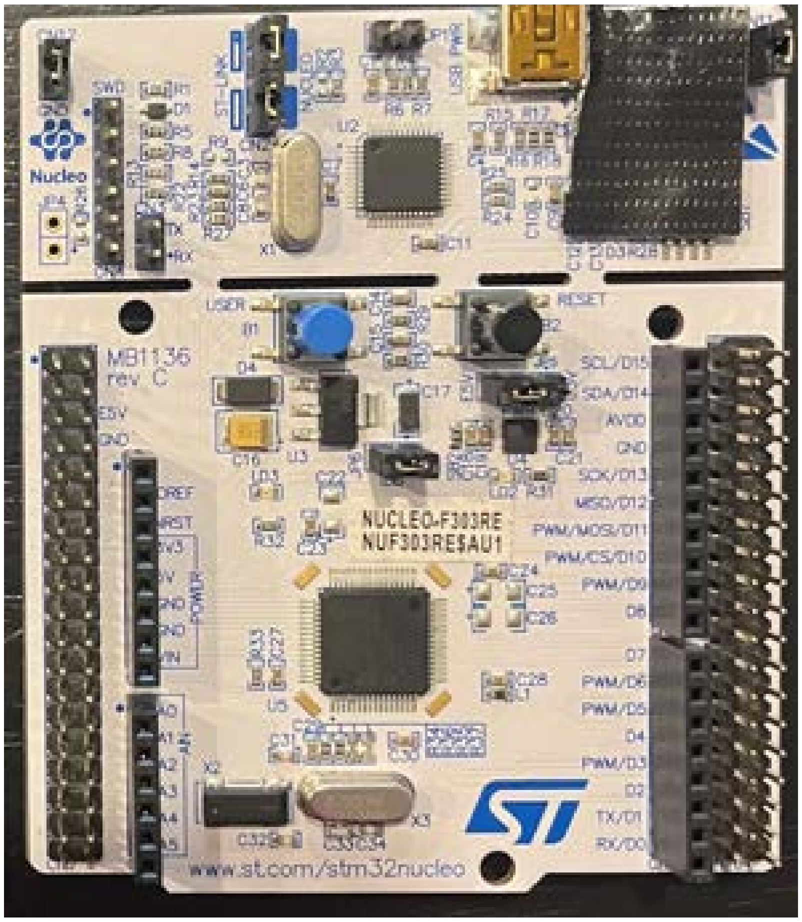


Figure 21: STM32-F303RE Microcontroller

5.9 Final Product

After assembly, the daughterboard PCB can be placed onto the pin headers of the NUCLEO development board.

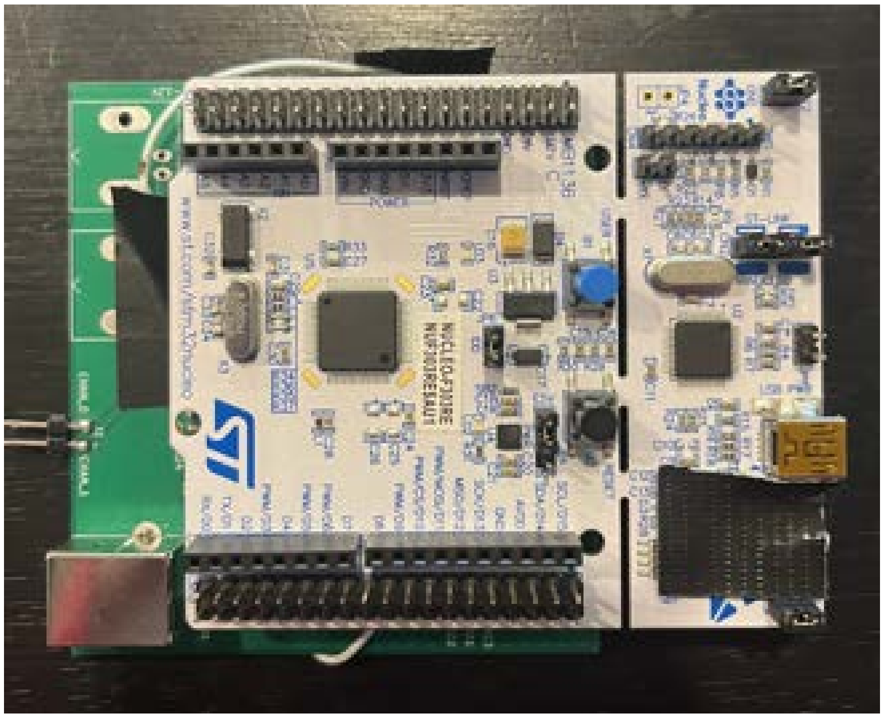
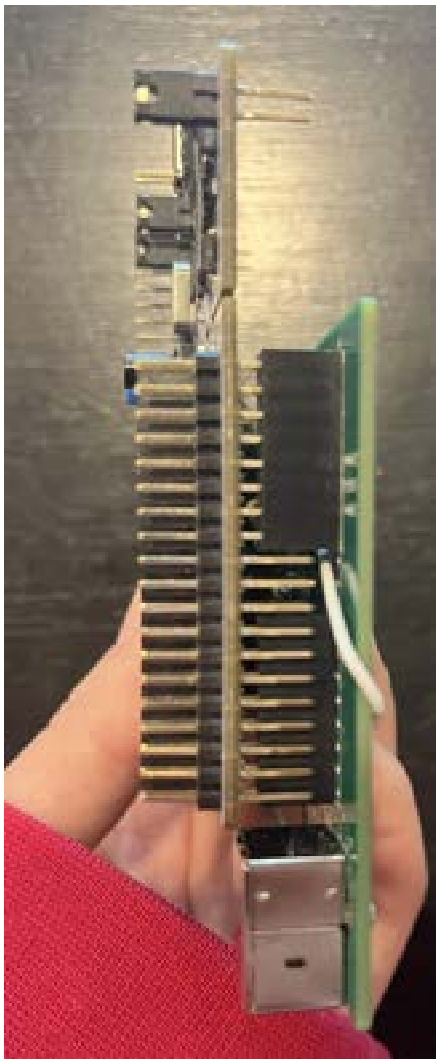



Figure 22: Fully Assembled PCB & MCU Device

6. Detail Design - Microcontroller

6.1 Interface State Machine Diagram

This section provides a systematic depiction of the control logic implemented in the STM32F303RE microcontroller, central to the operation of our waveform generator, as represented by the state machine diagram.

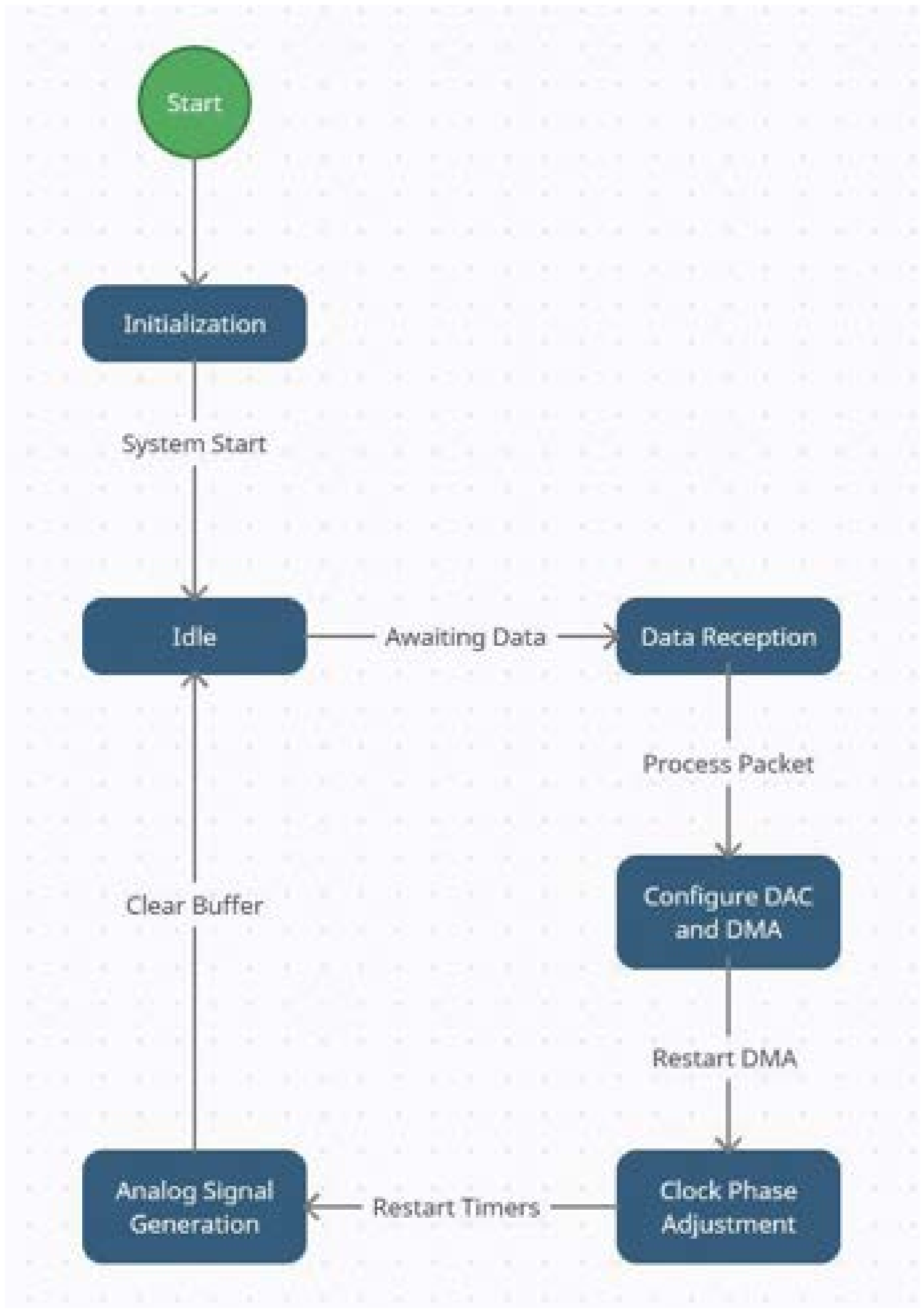


Figure 23: MCU State Machine Diagram

6.2 MCU Hardware Utilization

The STM32 microcontroller leverages an array of integrated hardware components to achieve efficient waveform generation. Timers within the MCU are employed to manage the precise timing needed for waveform modulation and to control the rate at which data is processed. The Direct Memory Access (DMA) channels are utilized to transfer the waveform data directly between memory and the Digital-to-Analog Converter (DAC) without burdening the CPU, thereby enhancing the system's real-time performance. The DAC converts the digital waveform data into an analog signal, ready for output. To facilitate these operations, the onboard RAM is used as a buffer, storing the waveform samples and ensuring a seamless flow of data. This orchestrated use of timers, DMA, DAC, and RAM enables the STM32 to function as a robust and precise waveform generator.

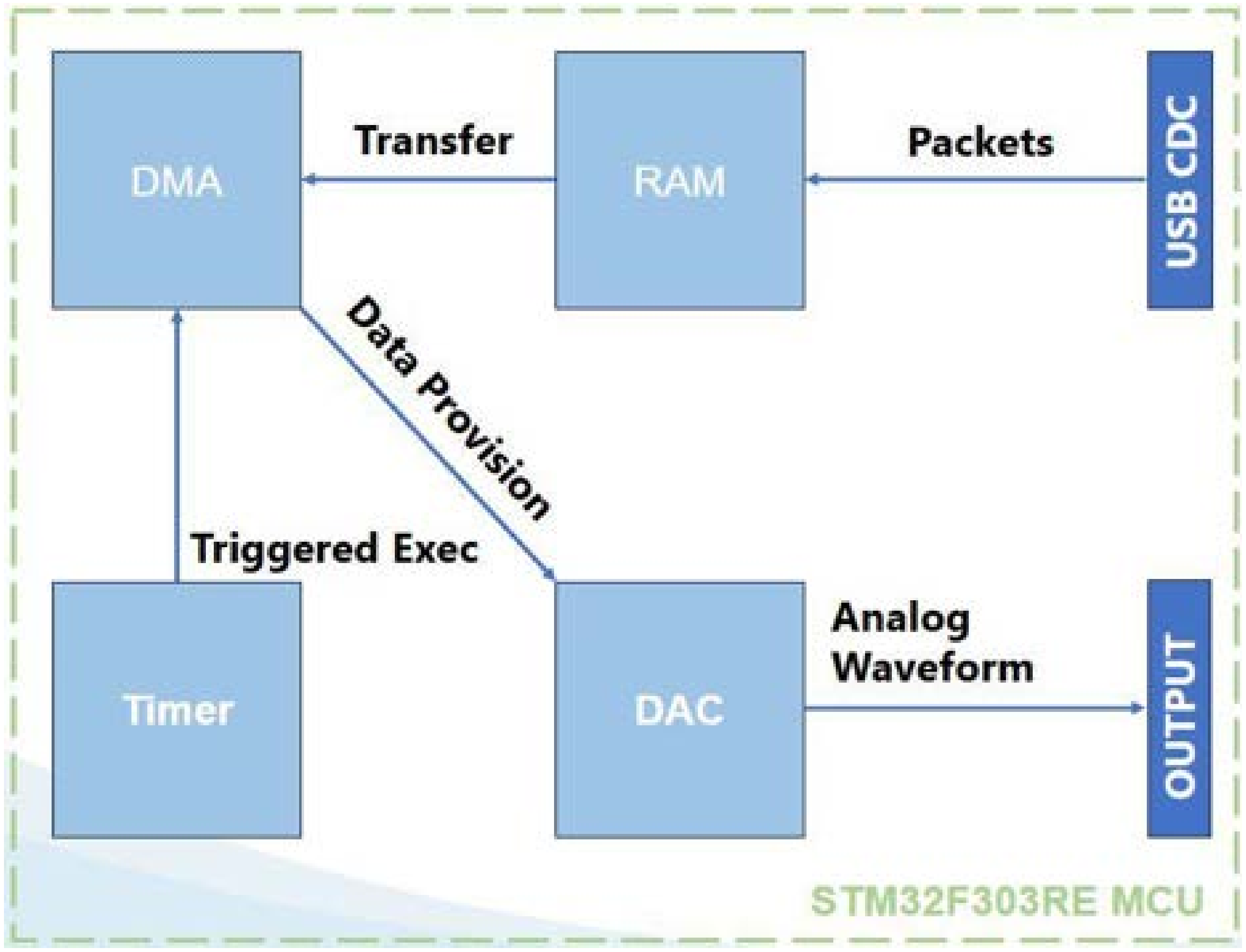


Figure 24: Hardware Utilization Chart

6.3 MCU Pinout

This section includes a straightforward pinout diagram of the STM32 microcontroller, which outlines the basic configuration used in our project. The diagram simplifies the understanding of our hardware setup, showing the essential connections and functionalities assigned to each pin.

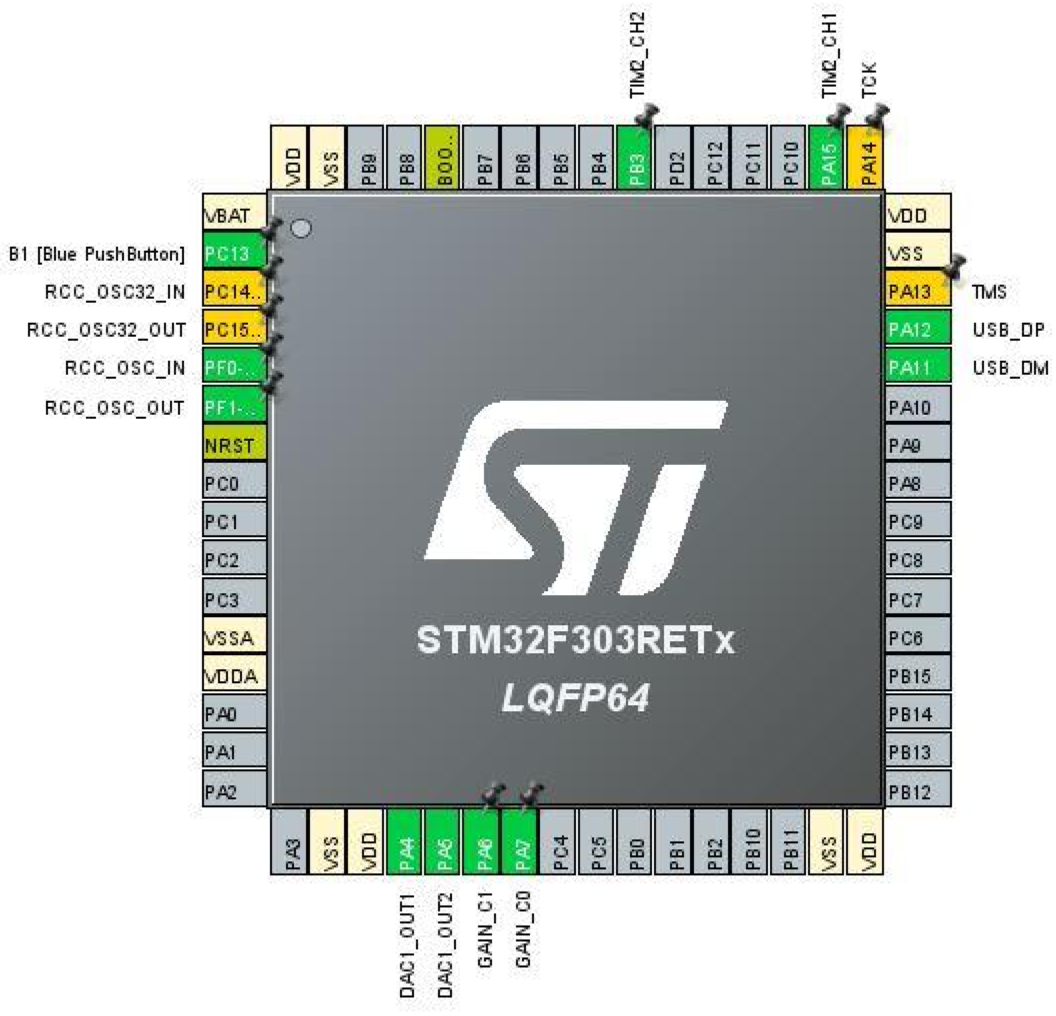


Figure 25: MCU Pinout from STM Cube Integrated Development Environment

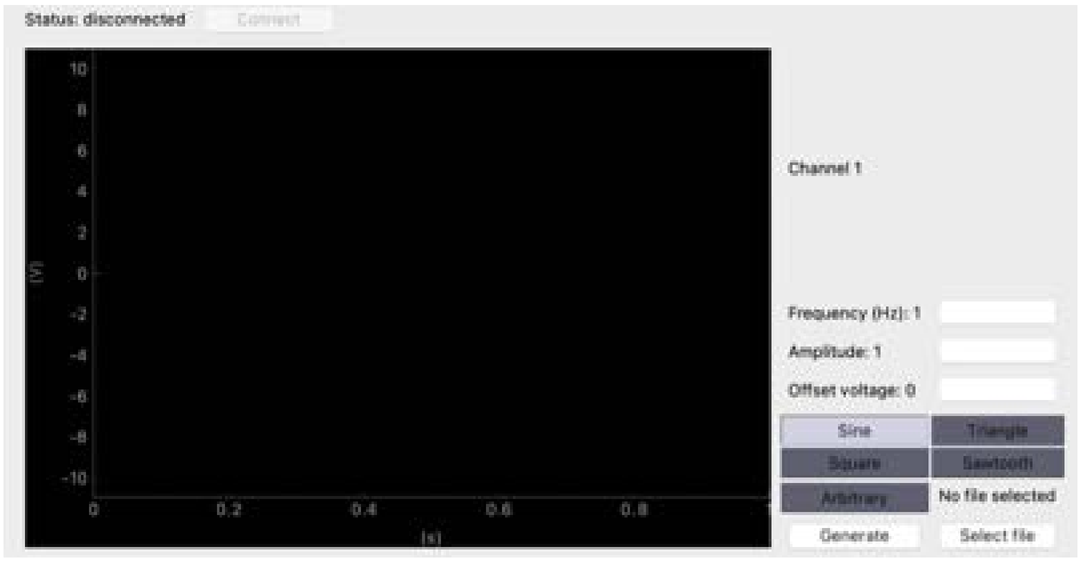
- **PA4 and PA5**: In our application, these pins serve as the output channels for the Digital-to-Analog Converter (DAC), with PA4 as DAC Channel 1 (DAC_OUT1) and PA5 as DAC Channel 2 (DAC_OUT2). These are critical for converting the digital signal into an analog waveform.
- **PA6 and PA7**: These General Purpose Output (GPIO) pins are designated for selecting the high and low gain settings in our system, which is a pivotal aspect of managing signal amplification.
- **PB3 and PA15**: As part of our design, these pins function as timer outputs to provide Pulse Width Modulation (PWM) signals. These signals are then used to adjust the offset.
- **PA11 and PA12**: These pins form the USB data interface, with PA11 as USB_DM (Data Minus) and PA12 as USB_DP (Data Plus), allowing for USB communication essential for data transfer and device control.
- **PF0 and PF1**: These pins are used for connecting an external crystal oscillator, which allows USB connectivity.

7. Detail Design - Graphical User Interface

7.1 Graphical User Interface Version 1

The initial version of our graphical user interface (GUI) was designed to deliver fundamental functionality. It enabled users to generate predefined wave shapes by specifying parameters such as frequency, amplitude, and offset. Upon entering these parameters and pressing the 'generate' button, the GUI would display the corresponding graphical representation of the wave. Simultaneously, it transmitted data packets to the microcontroller unit (MCU), ensuring a seamless integration of user inputs with the hardware functionality.

7.1.1 Main GUI


Status: disconnected

Channel 1

Frequency (Hz): 1

Amplitude: 1

Offset voltage: 0

No file selected

Figure 26: GUI Version 1 - Buttons

7.1.2 GUI AWG Wave Drawer - Matplotlib

The initial design for the arbitrary wave drawer was minimalistic and offered the basic functionality for testing arbitrary waveform generation. The prototype was a stand alone program from the main GUI that implemented Matplotlib to allow the user to draw waveforms. The samples of drawn waves could then be saved to a csv file and uploaded to be displayed in the main GUI.

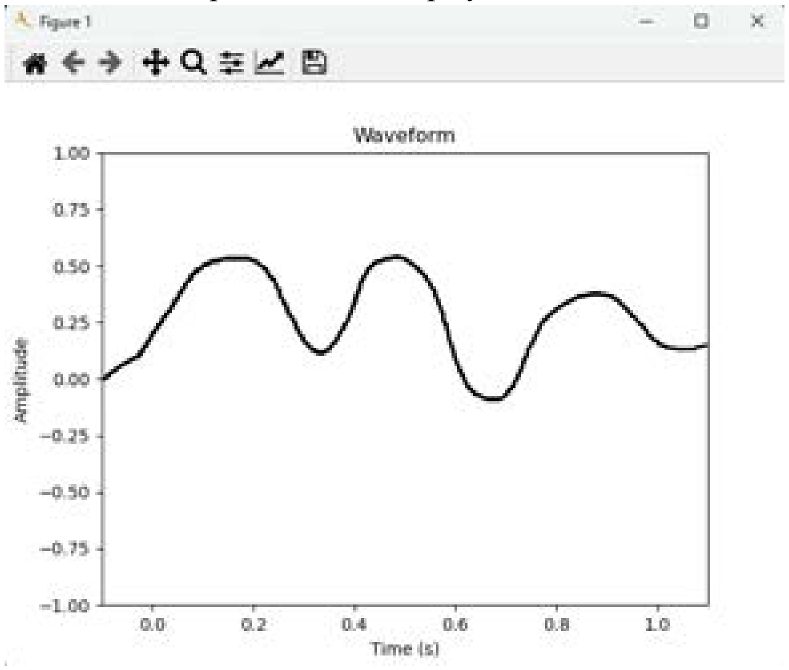


Figure 27: Wave Drawer Version 1- Matplotlib

7.2 Graphical User Interface Version 2

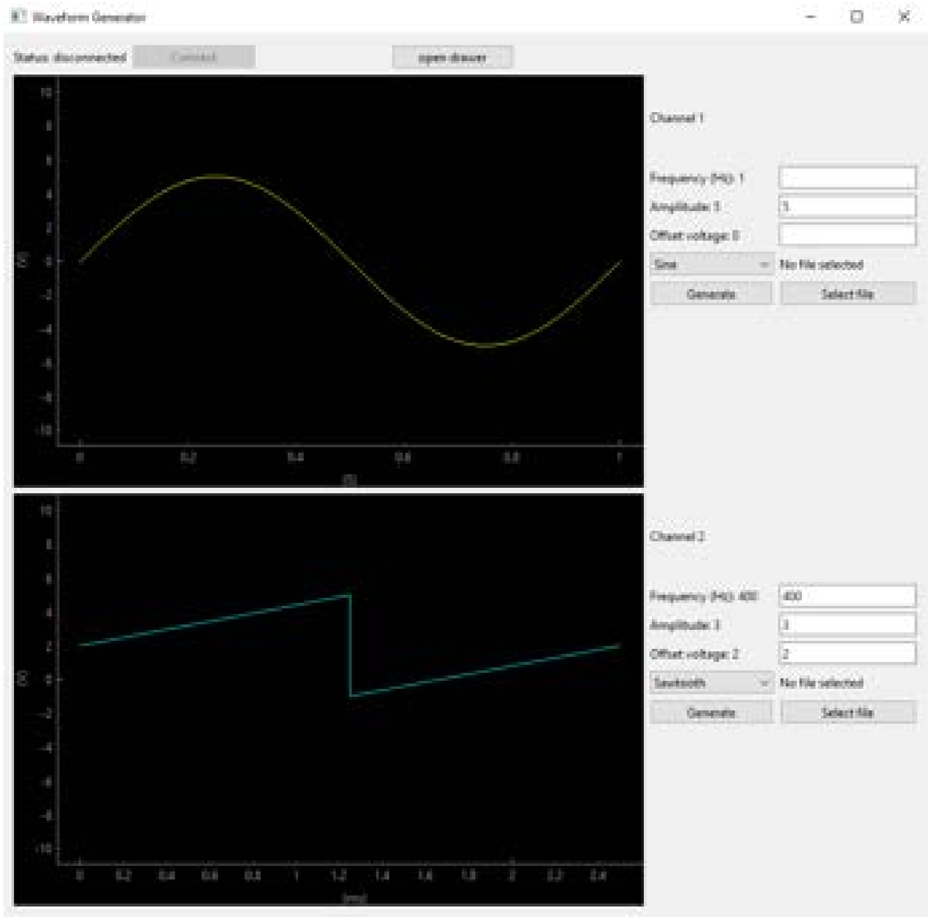
7.2.1 Main GUI

The second iteration of our graphical user interface aimed to incorporate a second channel while enhancing its intuitiveness. To achieve this, we prioritized modular coding, leading to a shift towards an object-oriented approach in our code development. This transition facilitated the addition of the second channel.

In our pursuit of a more intuitive interface, we identified several areas for improvement, particularly in the graphical representation. A key issue was the display of waveforms. High-frequency waves tended to appear compressed, whereas low-frequency waves displayed the opposite effect. To address this, we experimented with different display strategies: initially setting a range between a minimum of one and a maximum of ten waves, and later trying to consistently show only a single period. Ultimately, we found that displaying a single period offered a clearer representation.

Another challenge arose with the wave offset changes, especially noticeable when shifting between certain frequency values (e.g., from 10 to 100). The updated graph often appeared similar to its previous state, failing to clearly depict the changes. To resolve this, we focused on enhancing the graphical updates. We implemented visual shifts that accurately reflect changes in the waveform, such as an offset alteration causing the entire wave to move up, thereby providing a more accurate and user-friendly visual representation.

This version of the wave drawer included some updates to make the program more intuitive from a user perspective. Instead of needing to run a separate program the user can open the wave drawer from within the main GUI. The mechanic for drawing was updated from a user being required to input all points of the wave to the user being able to manipulate all samples or sections of pre-generated waves. With this change, sample presets were added for all of the generic wave types by the AWG project.



Status: disconnected

Channel 1

Frequency (Hz): 1
Amplitude: 5 | 5
Offset voltage: 0
Sine | No file selected

Channel 2

Frequency (Hz): 400 | 400
Amplitude: 3 | 3
Offset voltage: 2 | 2
Sawtooth | No file selected

Figure 28:GUI Version 2 - Buttons

7.2.2 GUI AWG Wave Drawer - PyQtGraph

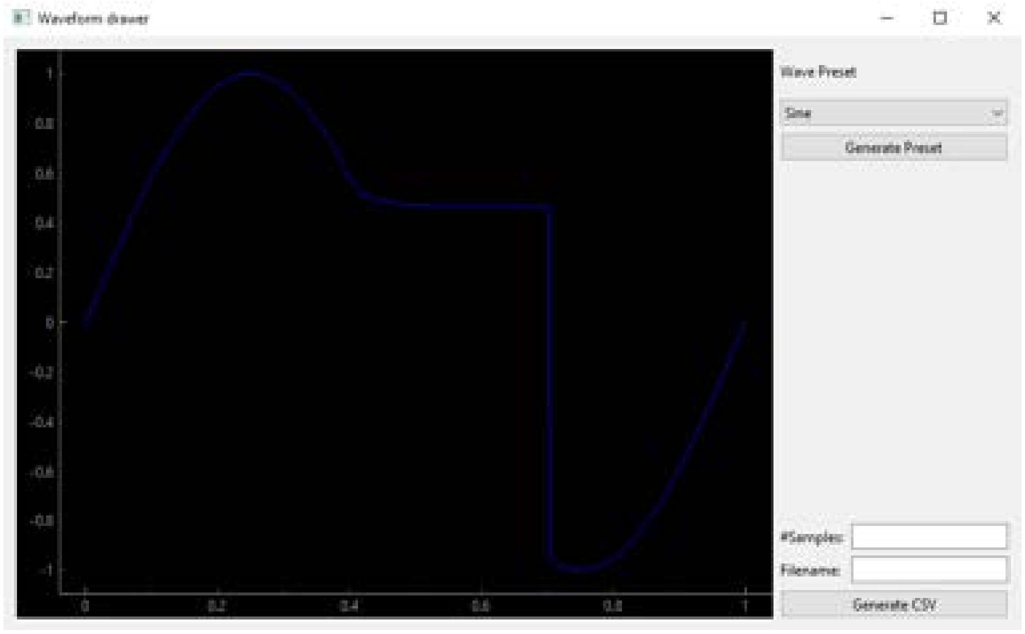


Figure 29: Wave Drawer Version 2 - PyQtGraph

7.3 Graphical User Interface Version 3 - Offset/Phase/Duty Cycle Implemented

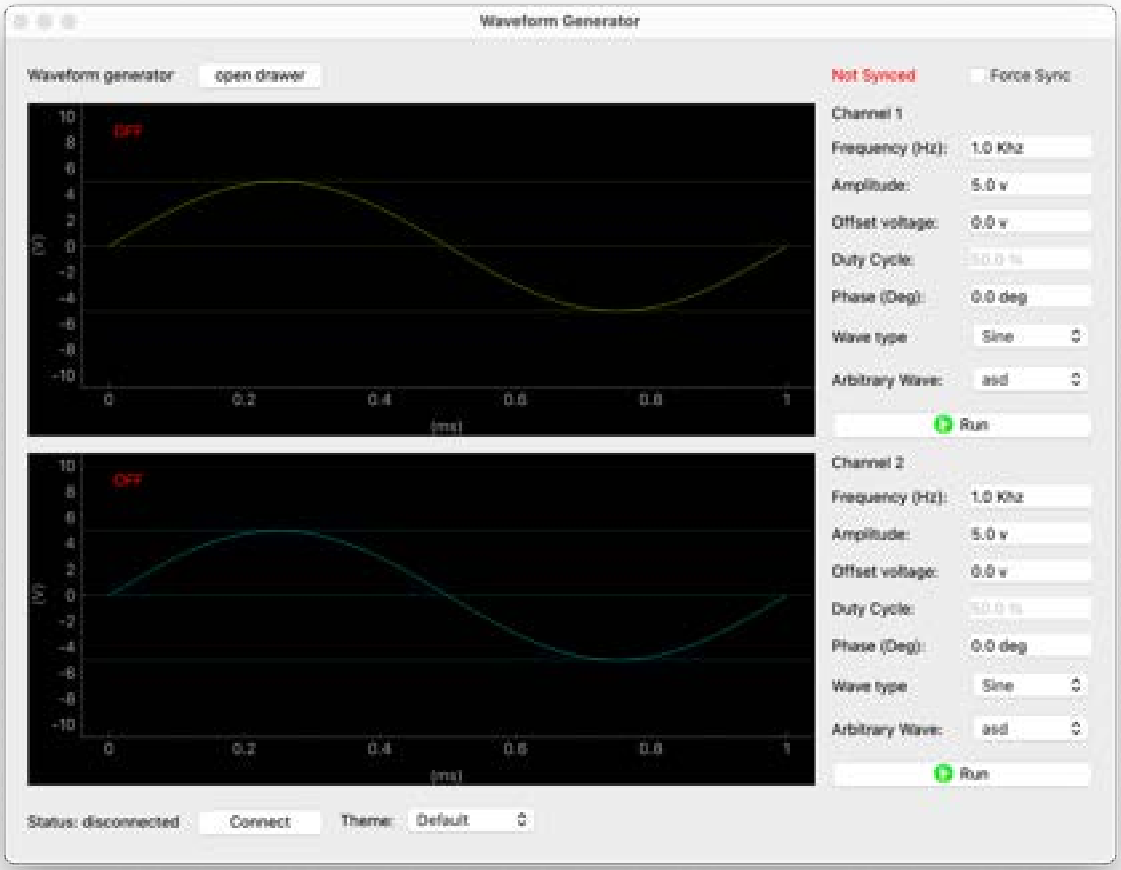
7.3.1 Main GUI

Having established a solid foundation for our graphical user interface (GUI), we focused on adding new features to enhance its functionality. We introduced additional parameters such as phase and duty cycle. To streamline the user experience, we made duty cycle adjustments and file selection options context-sensitive, activating them only when the appropriate waveform types (DC and arbitrary, respectively) are selected.

We also improved user input flexibility. Now, users can specify units directly in the GUI, and we have enhanced the system's capability to manage edge cases in inputs. Furthermore, we expanded the ways users can input data. For example, they can now use the scroll wheel to linearly increase or decrease the value in a selected input box.

A notable update is the replacement of the 'generate' button with a 'run' button. This change allows for the automatic updating of the wave being sent as long as it is toggled, facilitating a more interactive and responsive experience. With the generate button removed, the graph now automatically updates to reflect the currently entered parameters, ensuring that users always see the most current representation of their settings.

Finally, we have upgraded the visual aspects of the GUI by implementing customizable button themes and overall themes, such as dark mode and light mode. These options cater to diverse user preferences and needs, including considerations for users who are colorblind or prone to eyestrain, thereby making our GUI more accessible and user-friendly.The team implemented a new feature in the form of a 'sync' button, accompanied by an indicator. This addition was designed to enhance user interaction with the dual-channel functionality of our application. The sync button allows users to easily synchronize the two waveforms, ensuring they operate in harmony. The accompanying indicator serves a crucial role in alerting users whenever the two waveforms become desynchronized. This feature not only improves the usability of the application but also provides a visual cue for users to maintain waveform synchronization effectively.



Channel 1
Frequency (Hz): 1.0 Khz
Amplitude: 5.0 v
Offset voltage: 0.0 v
Duty Cycle: 50.0 %
Phase (Deg): 0.0 deg
Wave type: Sine
Arbitrary Wave: asd

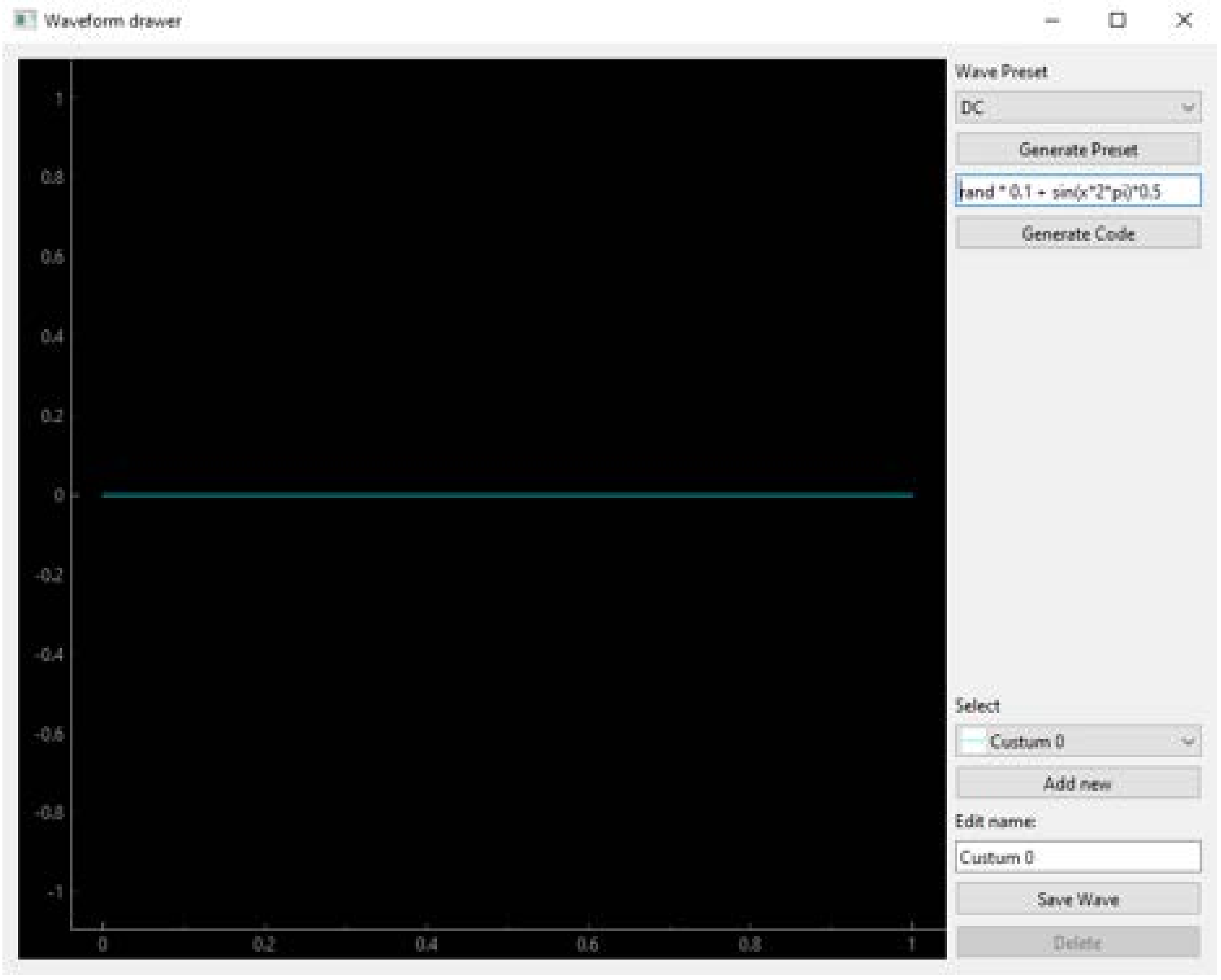
Channel 2
Frequency (Hz): 1.0 Khz
Amplitude: 5.0 v
Offset voltage: 0.0 v
Duty Cycle: 50.0 %
Phase (Deg): 0.0 deg
Wave type: Sine
Arbitrary Wave: asd

Status: disconnected
Theme: Default

Figure 30: GUI Final Version

7.3.2 Wave Drawer Final Version

The final version of the Arbitrary waveform drawer implemented an overhaul of the previous method for saving waveforms. Instead of saving each wave wave as an individual csv file, the method stores and modifies wave data in a text file. The user can now add, delete, and modify previously created waves. The wave data is also stored in a list that is passed into the main GUI, thus the user no longer needs to upload the samples for arbitrary waves themselves.
The last change is an additional preset option that allows the user to input a formula for a desired wave. This functionality allows for precision of arbitrary waves that was not previously possible.



Wave Preset: DC

rand * 0.1 + sin(x*2*pi)*0.5

Select: Custum 0

Edit name: Custum 0

Figure 31: Wave Drawer Final Version

8. Detail Design - Device Case

8.1 Device Case Overview

The primary purpose of designing a case for our device was to prevent damage to the hardware components, and make the final product look more visually appealing to the end user. Design of the case was done in Autodesk Inventor Professional 2024. The design is composed of two pieces, a sleeve that holds the attached PCB/MCU and the cover that attaches to the back. The files for the sleeve and cover were then converted to stl file types and printed using a Makerbot Replicator 3D Printer. The final version was printed in PLA plastic with a 25% infill density.

8.2 Cad Rendering of Case Sleeve

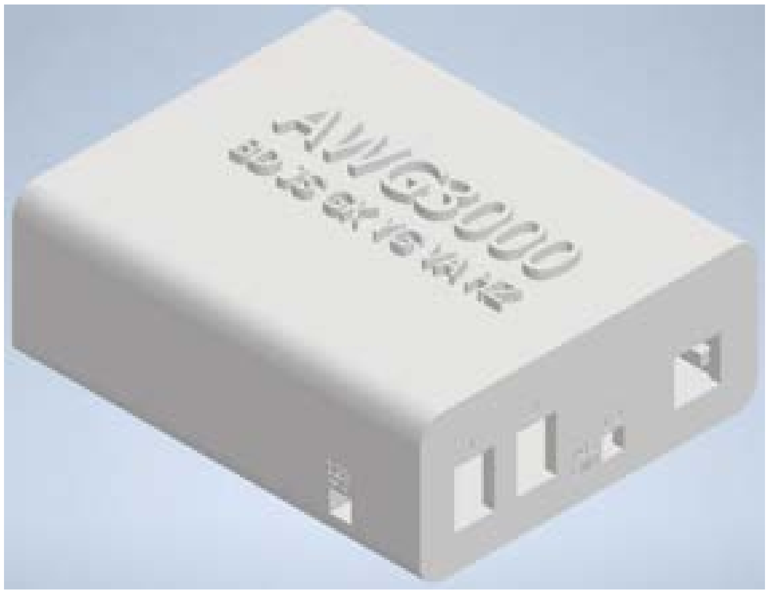


Figure 32: Case Sleeve - Isometric View

8.3 Cad Rendering of Case Cover

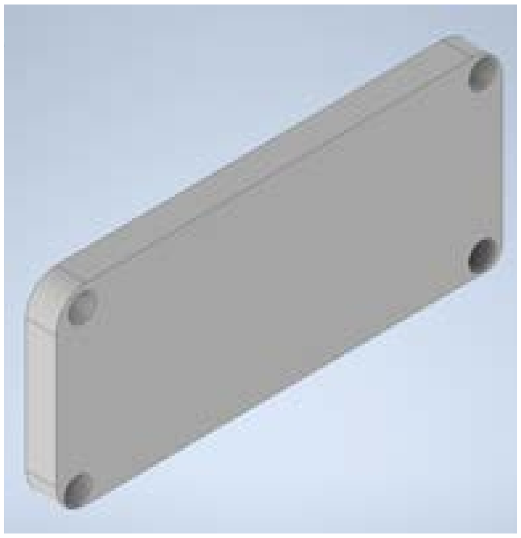


Figure 33: Case Cover - Isometric View

8.4 Cad Assembly View of Case

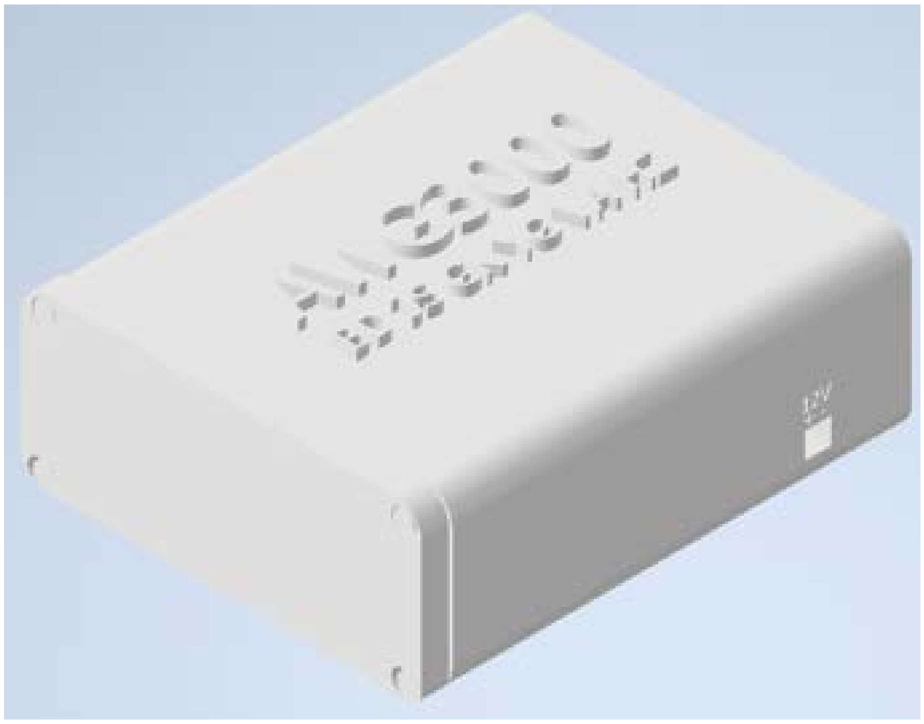


Figure 34: Assembled Case View - Isometric View

8.5 Top View of Assembled Case

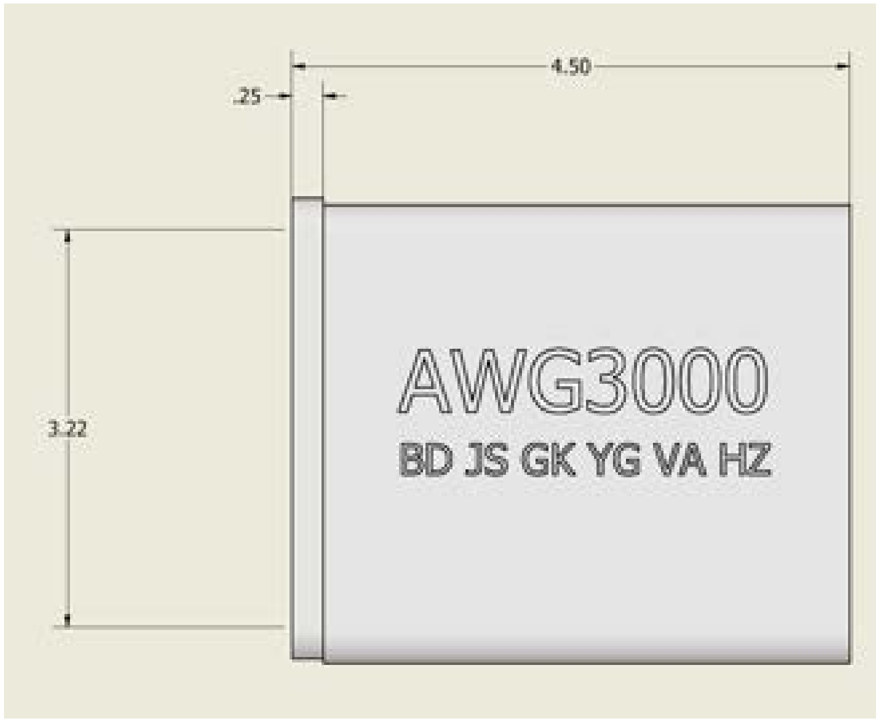


Figure 35: Top View of Assembled Case with Dimensions

8.6 Front View of Assembled Case

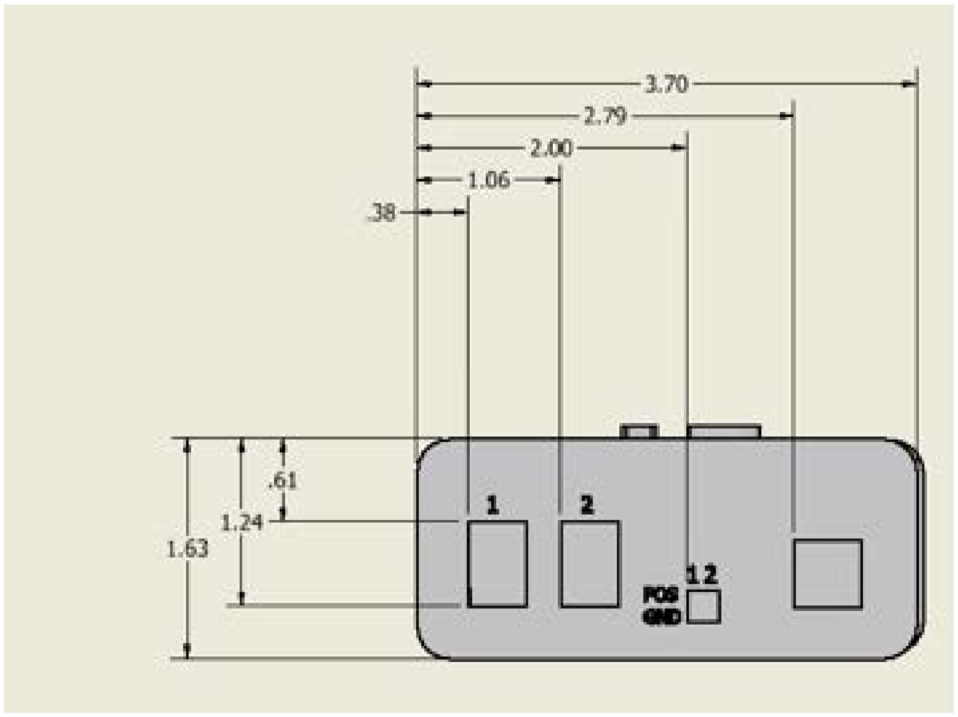


Figure 36: Front View of Assembled Case with Dimensions

8.7 Side View of Assembled Case

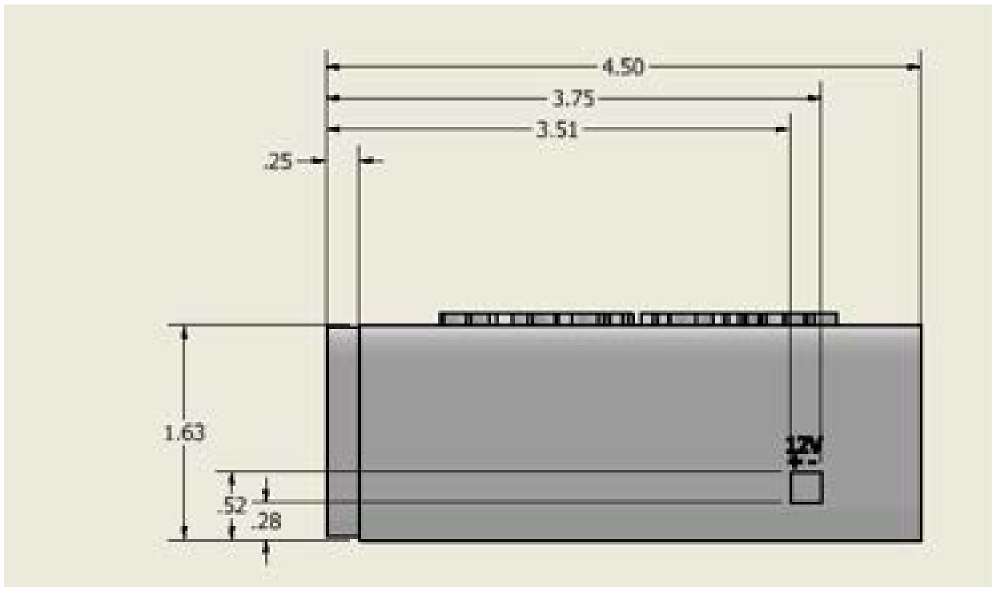


Figure 37: Side View of Assembled Case with Dimensions

8.8 Printed Case Model

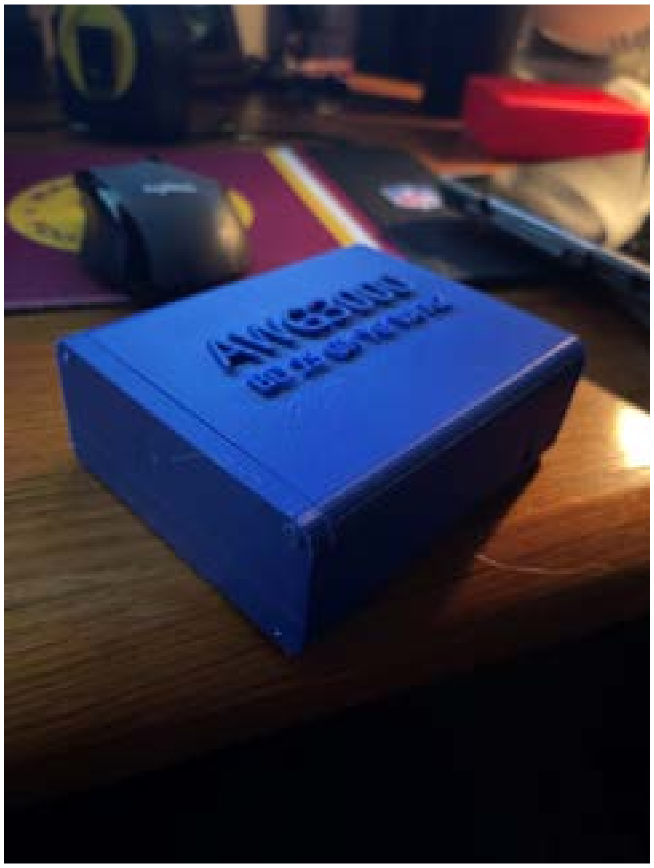


Figure 38: Printed Case Model in PLA - 25% Infill

9. Experimental Plan and Selection of Evaluation of Criteria

9.1 Overview

This project will be completed over the course of two semesters and will be split into two phases (ECE 492 and 493). The project itself can be broken down into three core categories that are integral to the design and functionality of the finished product. These divisions are the analog front end architecture and PCB design, microcontroller unit programming, and graphical user interface programming. Designated team members will be working in parallel to complete required tasks. It is expected that by the end of ECE 492 that we will have a functional prototype on a breadboard, so that PCB development can begin at the start of ECE 493 at the latest.

9.1.1 Hardware design

We will use generic op amps in the analog front end to condition the signal from the MCU. We will have to design the circuit to correctly apply the gain and offset voltage. We will also have to design the circuit for the buck and boost regulators. Finally, we will have to design and populate a PCB for this circuit.

9.1.2 MCU

For this project, we shall use the NUCLEO -F303RE development board with STM32F446RE MCU. This particular board was selected by a team working on an oscilloscope design under similar constraints. Given that integration of their oscilloscope design and our arbitrary waveform generator is an eventual possibility it makes sense to use the same board as we will still have access to the two digital to analog converters necessary.

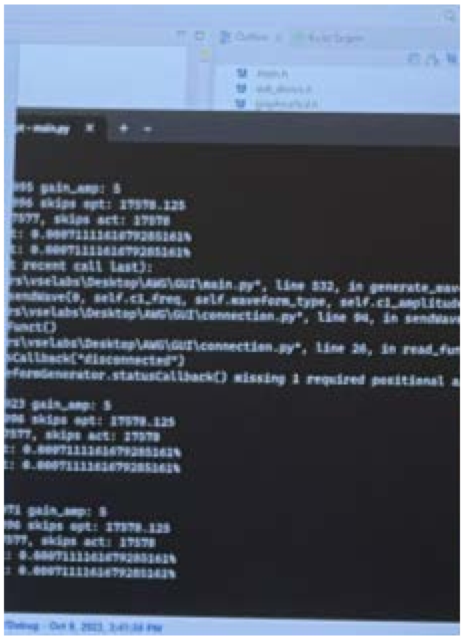
9.1.3 GUI Programming

The graphical user interface was designed using PyQtGraph, a scientific graphics and GUI library for python. This library can be used to create the GUI as well as plot waveforms with data being provided. Users must be able to enter certain data to define the waveform in order to be sent to the MCU. Upon clicking the generate button, a visual display of the waveform must be displayed on the GUI.

10. Experimentation and Success Evaluation

10.1 Experimentation of MCU

10.1.1 Communication Testing:



```
995 gain_amp: 5
996 skips opt: 17578.125
7577, skips act: 17578
: 0.0007111161679285161%
: 0.0007111161679285161%
 recent call last):
rs\vselabs\Desktop\AWG\GUI\main.py", line 532, in generate_wav
sendWave(0, self.c1_freq, self.waveform_type, self.c1_amplitud
rs\vselabs\Desktop\AWG\GUI\connection.py", line 94, in sendWave
Funct()
rs\vselabs\Desktop\AWG\GUI\connection.py", line 26, in read_fun
sCallback("disconnected")
eformGenerator.statusCallback() missing 1 required positional a

823 gain_amp: 5
996 skips opt: 17578.125
7577, skips act: 17578
: 0.0007111161679285161%
: 0.0007111161679285161%

971 gain_amp: 5
996 skips opt: 17578.125
7577, skips act: 17578
: 0.0007111161679285161%
: 0.0007111161679285161%
```

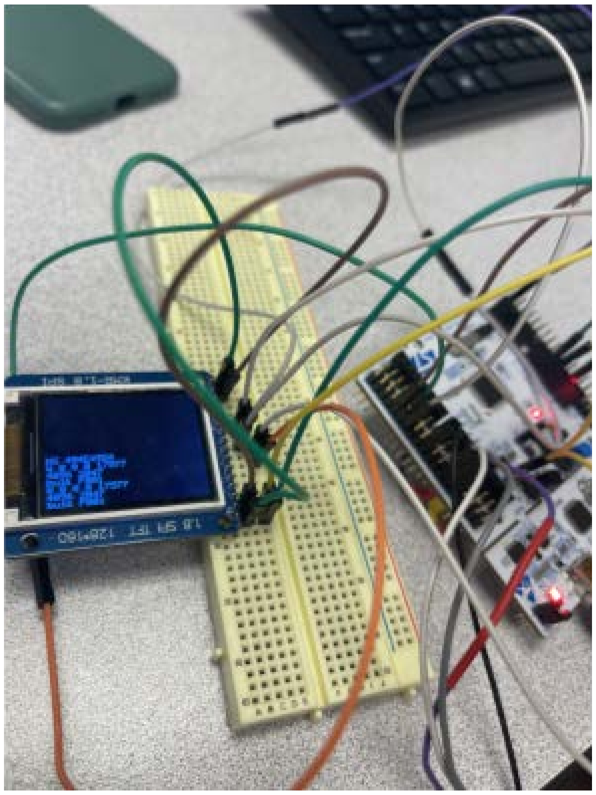


Figure 39: Communication Protocol Testing Setup

In order to test the communication between the GUI and MCU, we attached an
Adeept LCD screen to show the data being passed through. When running the
GUI from a terminal, the same data is displayed on the PC side as well. The
handshake ID, acknowledgement, and various packet data like the sampling
rate. From here, we were able to confirm that the graphical user interface and microcontroller can adequately send data between one another, with or without the hardware attached.

10.1.2 PWM Signal Testing:

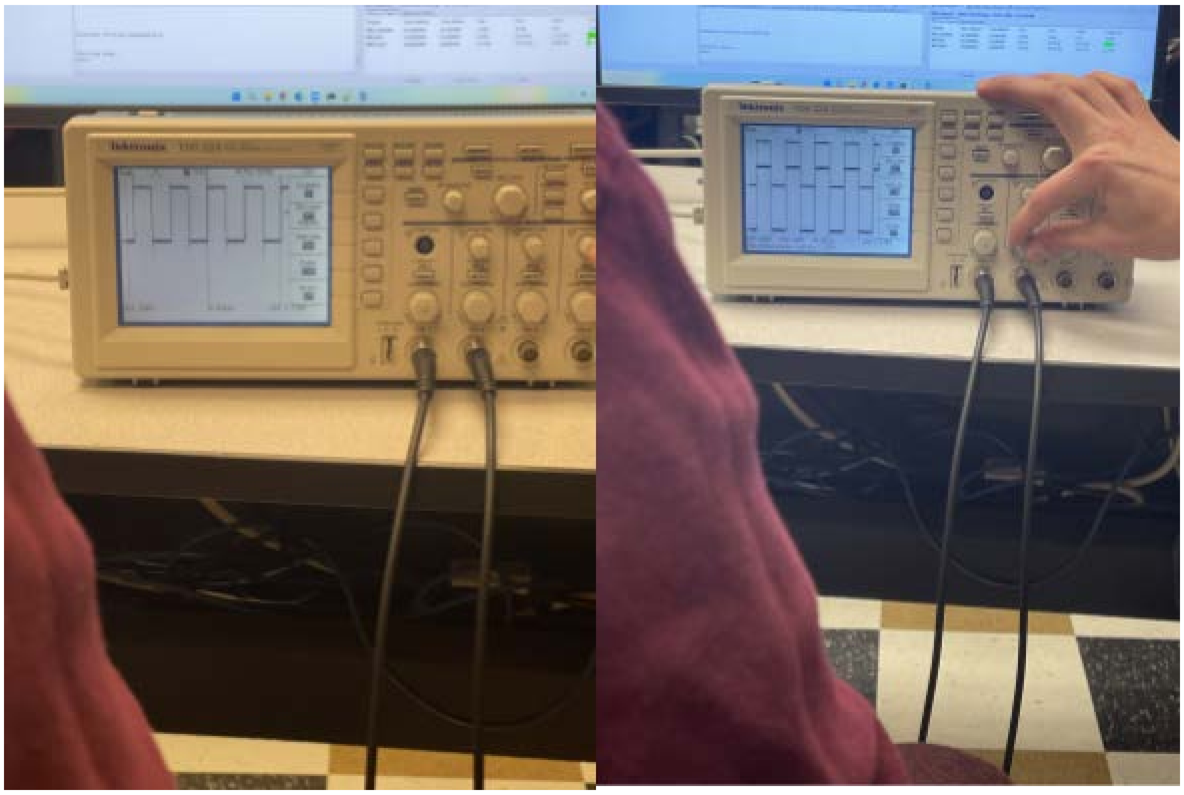


Figure 40: PWM Signal Testing Setup

The purpose of pulse-width modulation (PWM) testing is to determine if the MCU is capable of taking an analog signal and converting it to a digital signal.PWM signals are just square waves where voltage is either supplied or not supplied. This test confirmed that our written microcontroller code is able to use the digital-analog converter (DAC) and analog-digital converter (ADC) to convert between signal types.

10.1.3 Pre-Defined Wave Testing:

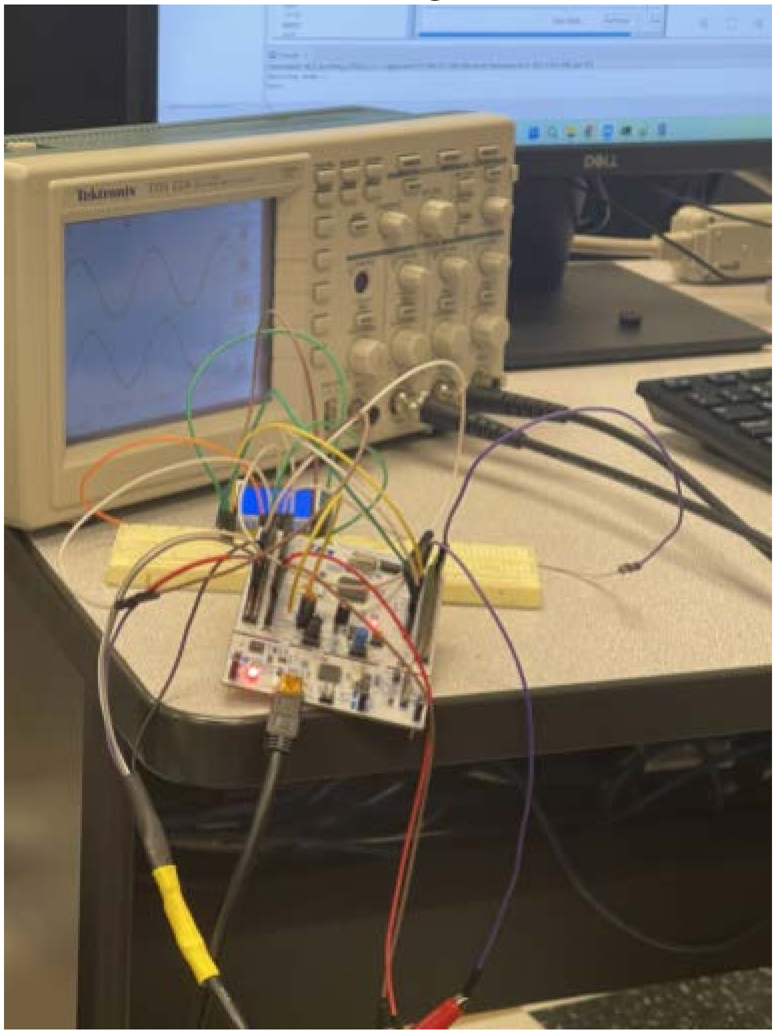


Figure 41: Pre-Defined Wave Testing Setup

In order to verify that the MCU can properly send waves, we supplied sine, triangle, and square waves to see the output before passing it across our custom analog hardware. These tests verified that our MCU is capable of taking wave data from the GUI with relative accuracy. The precision of the PCB is ultimately necessary for making sure that the proper scaling and amplification is done at the output.

10.2 Experimentation of GUI

10.2.1 Predefined Wave Testing - with Buttons - Different Operating Systems

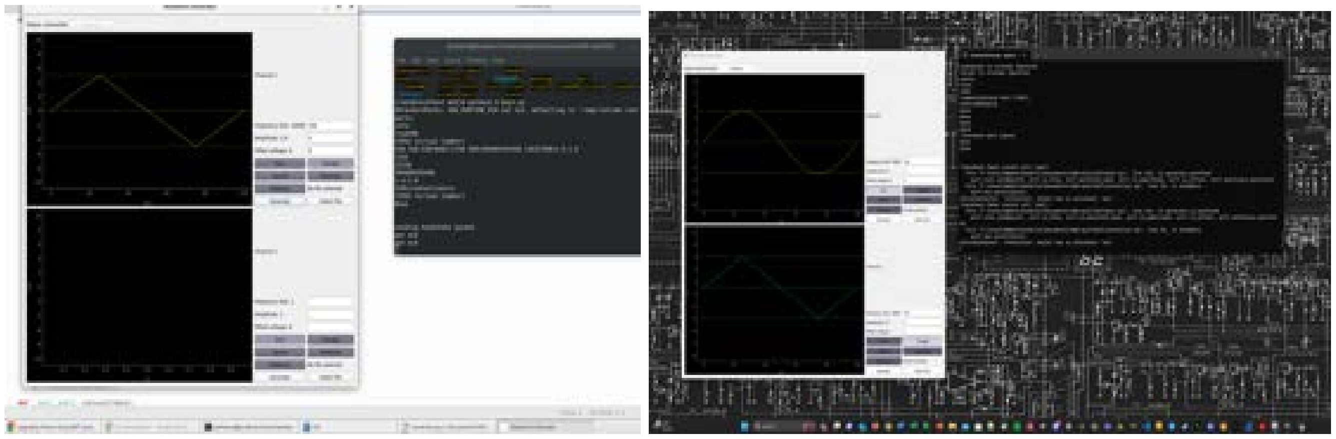


Figure 42: GUI Testing on Fedora Linux (Left) and Windows 11 (Right)

One of our key objectives was to ensure that our application functioned seamlessly across multiple operating systems. The application relies on a variety of Python packages, which it is designed to download automatically. However, each operating system has its unique method for handling this automated package downloading process. Our testing process, therefore, not only involved verifying the successful download of these packages across different platforms but also encompassed a thorough examination of the application's functionality through user application and QA testing. An additional critical aspect was testing the compatibility of different operating systems with the hardware, considering that operating systems recognize and label external devices differently.

During our multi-system testing, we encountered issues that were not related to the operating system but rather to the hardware. For instance, we observed that on devices with a built-in dark mode (notably the primary PC used for testing), the entire GUI would default to dark mode, even though it was not set to this theme. Conversely, on devices with a built-in light mode, the GUI would display in light mode. This discrepancy highlighted areas where certain colors, especially for buttons, were hardcoded. This realization prompted us to revise our approach to the visual design of the application.

Furthermore, we noted that the layout of widgets within the GUI was affected by the hardware, particularly the screen size. This variability necessitated additional adjustments to ensure a consistent and user-friendly interface across different devices and screen sizes.

10.2.2 AWG Wave Drawer Testing - Matplotlib

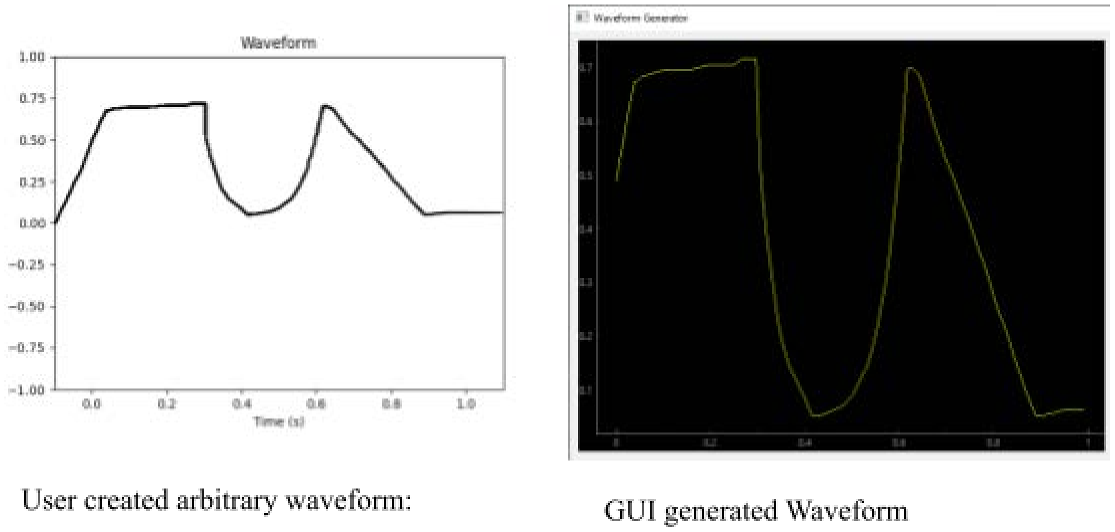


Figure 43: Wave Drawer Testing - Matplotlib
One critical aspect of the GUI is making sure that all information being displayed to the user is accurate. So it is important that user drawn waves are displayed correctly on the main GUI and the samples are not corrupted between the time that they are generated and when they are displayed. The initial prototype generated samples in a csv file to be read in the main GUI. During testing we ensured that the samples generated were the same as the sample being used in the main GUI's display function

10.2.3 AWG Wave Drawer Testing - PyQtGraph

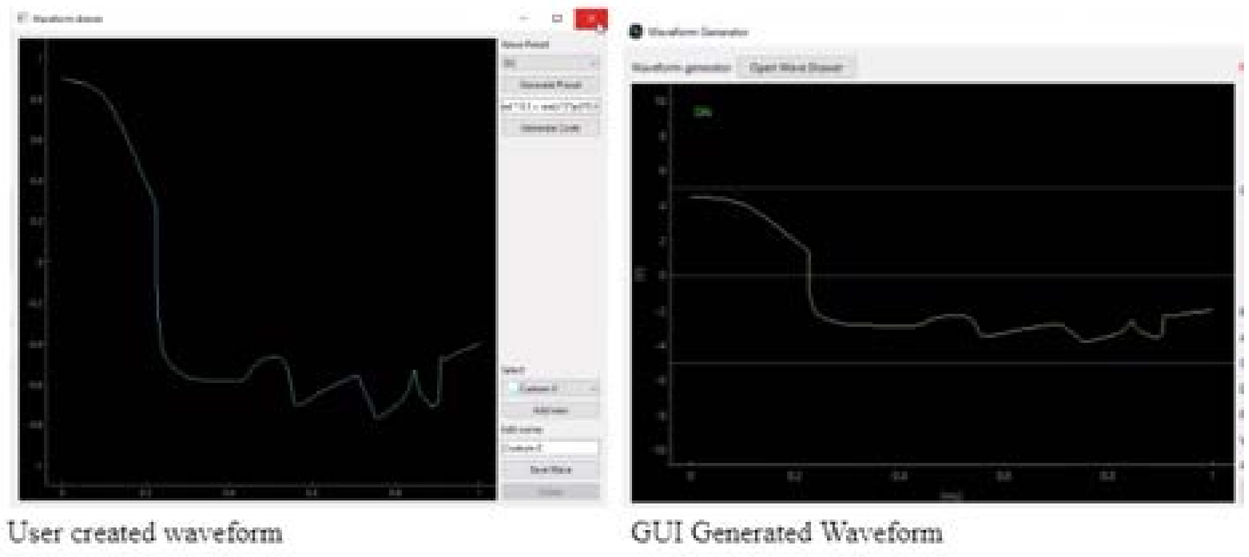


Figure 44: Wave Drawer Testing - PyQtGraph
In later stages of the project, the task of saving and loading arbitrary waveforms was done internally rather than the user needing to access separate files. Thus, the array of samples was

being passed directly to the main GUI. Similarly to previous prototype testing we could ensure that the samples were unchanged.

10.3 Experimentation of integrated MCU and PCB

The MCU and PCB were tested together. An automated script was used to configure the MCU to output a specific wave and record the signal coming out of the PCB with and AD2.
This allowed for a number of tests, such as the DAC linearity, error in offset, and amplitude falloff with higher frequency

10.3.1 DAC Sweep with constant offsets / gains

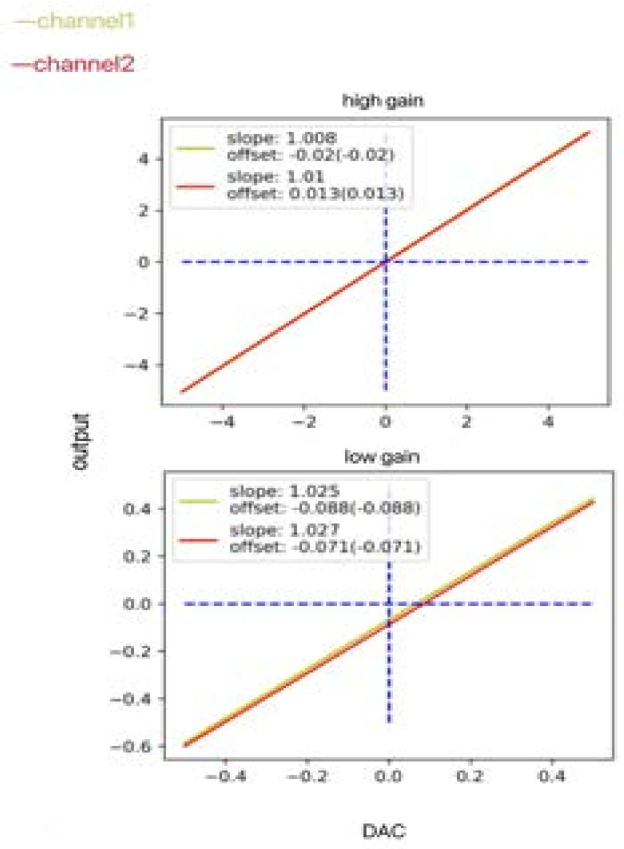


Figure 45: DAC Sweep Testing Results

These two figures represent DAC_sweep. This occurs when the offset equals 0. X-axis is DAC and Y-axis is output. As we know, the output equals DAC + offset. Therefore, the results match our expectations. We also have results measured when the offset equals other values, as shown in Appendix. Error evaluation: There's tiny error between the results and expectations. The slope is supposed to pass through (0,0), however, it has a tiny intercept, which is caused by the 2% error of the diode from the circuit we designed. Additionally, the high gain errors are all smaller than the low gain errors, regardless of the values of the DAC.

10.3.2 Offset Sweep with constant DAC values/gains

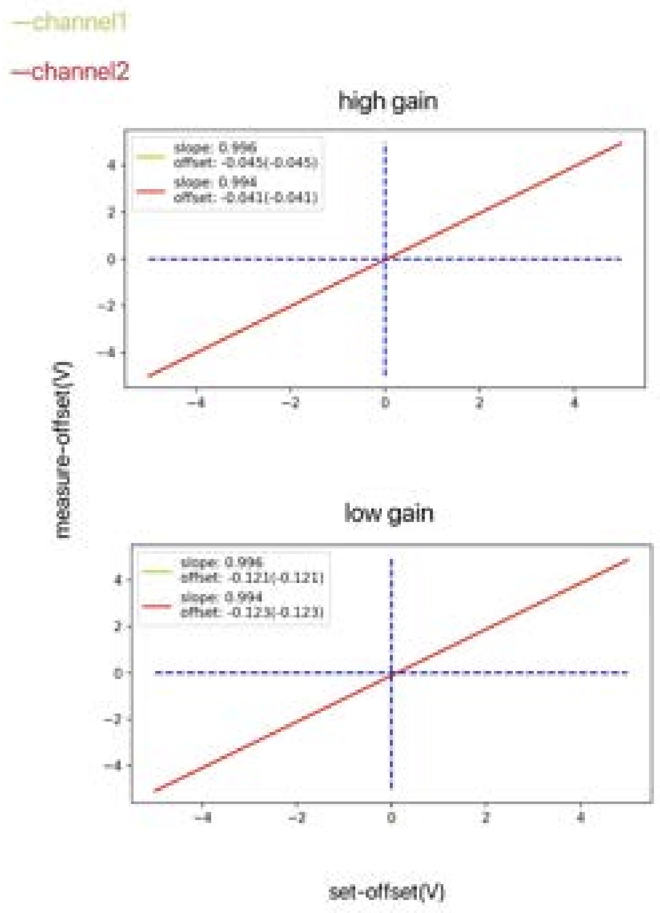


Figure 46: Offset Sweep Testing Results

Offset_sweep when DAC=0

These two figures represent offset_sweep. This occurs when DAC equals 0. X-axis represents set-offset and Y-axis represents measure-offset. As we know, measure-offset equals DAC + set-offset. Therefore, the results match our expectations. We also have results measured when DAC equals other values, as shown in Appendix.

Error evaluation: There's tiny error between the results and expectations. The slope is supposed to pass through (0,0); however, it has a tiny intercept, which is caused by the 2% error of the diode from the circuit we designed. Additionally the high gain errors are all smaller than low gain errors, regardless of the values of the set-offset.

10.3.3 Frequency Response Testing

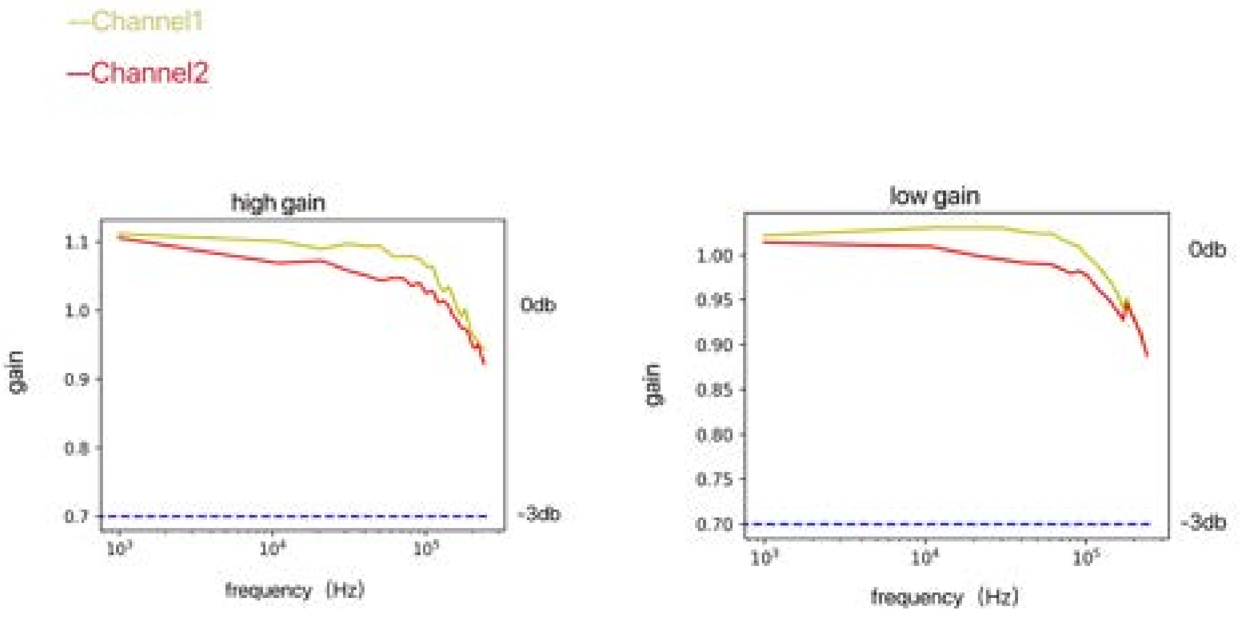


Figure 47: Frequency Response Testing Results

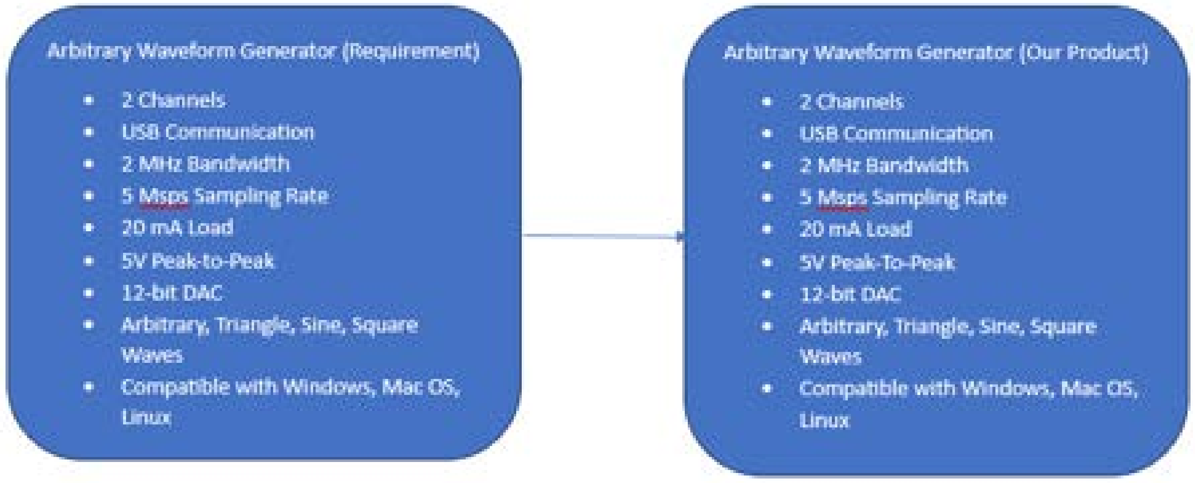
Frequency response when offset =0.

These two figures represent Frequency Response. This occurs when DAC equals 0. X-axis is frequency and Y-axis is gain. We also have results measured when the offset equals other values, as shown in Appendix.

Error evaluation: There's error between the results and expectations. The Y-axis is supposed to start at 1; however, it starts around 1.1, which is due to the error of the resistor we used in the circuit we designed. Additionally, we can't obtain the value of cutoff frequency which is because the result is too large, and we are not able to measure it.

10.4 Project Success Evaluation

10.4.1 Overall Project Evaluation



Arbitrary Waveform Generator (Requirement)

- 2 Channels
- USB Communication
- 2 MHz Bandwidth
- 5 Msps Sampling Rate
- 20 mA Load
- 5V Peak-to-Peak
- 12-bit DAC
- Arbitrary, Triangle, Sine, Square Waves
- Compatible with Windows, Mac OS, Linux

Arbitrary Waveform Generator (Our Product)

- 2 Channels
- USB Communication
- 2 MHz Bandwidth
- 5 Msps Sampling Rate
- 20 mA Load
- 5V Peak-To-Peak
- 12-bit DAC
- Arbitrary, Triangle, Sine, Square Waves
- Compatible with Windows, Mac OS, Linux

Figure 48: Project Requirement to Outcome Comparison

Overall, the project can be considered a success. The GUI team successfully created an interface that is both user friendly and complements interfaces already in use by students today like Digilent Waveforms, or Scopy. This interface is able to connect to the MCU via a communication protocol that communicates data from the GUI over a serialized USB connection to the MCU. From there the MCU is able to process those data packets into information that can be passed off to the analog front end for offset adjustment and gain amplification. The device is then able to output the predefined or custom waveform specified by the user in the GUI in its corresponding output channel.

10.4.2 Other Issues

- **Reason for the Project:** The American economy follows this natural boom and bust cycle every 8-10 years throughout its entire history. If we were to go back, then we could notice the Oil Embargo of 1990, then the DotCom crash of 2000, the housing-crisis of 2009, and more recently the COVID-19 pandemic of 2019; we see a common pattern. As Engineers, the supply and demand chain as well as the causes of these crashes; gives us new problems to solve.

  With the onset of the COVID-19 pandemic, demand for computer chips increased while production or supply decreased. The workforce and students alike were sent home to complete work from home. For electrical engineering students, their curriculum consists of a series of hardware lab classes where state of the art bench equipment is required to simulate and test hardware. However, due to the online nature of classes, the on-campus lab rooms were unavailable for use; forcing students to buy all-in-one USB devices to complete their experiments.

Devices like the Analog Discovery 2 by Digilent or the ADALM2000 were recommended by ECE professors for use in these experiments. Currently the AD2 costs around $399, and the ADALM costs around $230; unaffordable to the average college student. This project sought to develop a cheaper alternative to these devices as apart of a three-team/project effort. In May of 2023, a group of students successfully developed a two-channel oscilloscope based off the STM32-F303RE MCU, the same one we are using. Our project solved the need for AC signals processing by designing the Arbitrary waveform generator, and lastly the logic analyzer component will be built by a future senior design team. Since all three components are based on the same MCU, there will be a group that combines our three projects, making a more affordable laboratory tool at around a $50 price range; significantly cheaper than the other alternatives on the market today.

- **Use Cases of the Project:** The main use cases of the project are undergraduate electrical and computer engineering students that are required to complete coursework where the simulation and testing of hardware using integrated circuit chips and low-level discrete electronics. In addition, it can marketed to hobbyists and academic researchers where access to moderate electrical engineering bench equipment is needed.

- **Final Design Maintenance:** By creating a design case where the end user only has access to the USB connector, channel outputs, and external tap-in to the power supply; we eliminate any maintenance that will need to be done on the side of the hardware. The device will require minimal maintenance and if any is needed, they would be the replacement of the connectors from physical wear and tear. The GUI will periodically require updates of python packages, as updates to the GUI would be released via GitHub or the product's website.

- **Life Cycle of Final Design:** All parts and components in this project were sourced from common vendors like Digikey and Mouser, and replacements can be easily found online. Its unlikely that our particular operational amplifier or power supplier chips are discontinued, but in the event that they are then similar alternatives could be found. In addition, there are many alternative waveform generator device schematics online, so if replacement components can not be found; our design could be modified to have the same functionality but perhaps with a different amplifier or with additional buffer stages added, etc. When it comes to the disposal of our device, it would be considered E-Waste and can be sent to any recycling center where other devices containing computer chips are sent. This project can be constantly improved and thus does not truly have an end of lifetime.

11. Administrative Section

11.1 Project Progress

11.1.1 Front-End: After developing version one of the analog hardware pcb, we found issues with noise and bandwidth capacity of the opamps selected. In the first two months of ECE-493, we developed two new revisions of the PCB with different properties. One revision was simply revision one but with an added buffer stage to allow for better filtering of noise. The second revision switched from two-channel opamps to four-channel opamps with better resolution and less noise on the supplied waves.

11.1.2 MCU: The MCU was configured using the STMCube Integrated Development Environment (IDE). In the first few months of ECE-493, we tested the configuration of the DAC and timers to ensure transfer of data from the GUI to the MCU via our communication protocol. In addition, we also tested the pulse-width modulation (PWM) or the supply of a square wave through the MCU.

11.1.3 GUI: Our graphical user interface was developed using PyQtGraph6, but is also compatible with PyQtGraph5. The GUI allows the user to generate predefined waveforms for channels one and two via a dropdown list, or open a wave-drawer to specify a user-defined wave. In the wave-drawer window, the user can start with a preset wave shape or a wave generated using a formula and click and drag with their mouse to get the desired shape. The user can store waves as arbitrary wave objects that contain the name of the wave and samples. These arbitrary wave objects are stored in a list passed to the main GUI and the list is updated automatically when waves are created or deleted . We also generated executable files for each operating system that users can download and run as a standalone application on their desktop computers.

11.2 Project Challenges

11.2.1 Front-End:

Throughout the development of the Analog Front End, we encountered issues with two things,creation of custom footprints and compactness.

11.2.2 MCU:

One significant challenge encountered during the project was the lack of detailed online documentation and examples specific to our application, particularly in configuring and managing the DAC, DMA, and timers of the STM32 microcontroller. This scarcity of resources made it difficult to ascertain the best practices for starting, stopping, and fine-tuning these components. Another major hurdle was developing a robust technique for handling data transmission from our device, where again, relevant documentation was minimal. Additionally, we faced hardware challenges, as evidenced by the failure of two boards, which could have been attributed to either hardware faults or software glitches. Compounding these issues was the complexity and non-intuitive nature of the debugging feature in STM32CubeIDE, which presented a steep learning curve and hindered efficient troubleshooting.

11.2.3 GUI:

During the development of our GUI, we encountered two primary challenges. The first issue pertained to integrating the wave drawer, developed using Matplotlib, into the main application, which was created with PyQt. As a standalone component, the wave drawer functioned correctly and would activate upon pressing the designated button. However, when embedded into the main GUI, it failed to register mouse activities and presses. Despite extensive research and attempts to resolve this issue, we found that Matplotlib and PyQt seemed to have compatibility limitations, hindering the interaction between the wave drawer and the main GUI. This interaction led to us having to recreate the wave drawer with pyqt. The second major challenge was the variability in the GUI's behavior across different operating systems. The most evident differences were in the GUI's appearance, including layout and color scheme, which varied significantly between operating systems. Additionally, each operating system had its distinct method for the automatic installation of the required packages for the application, adding to the complexity of cross-platform functionality. Another significant issue was the detection of the microcontroller unit (MCU). Each operating system had a different approach to recognizing the MCU, and we noted that particularly in the macOS version of the application, it would not function correctly if the MCU was not detected, even during development stages. These challenges underscored the need for a more adaptable and robust design approach to ensure consistent performance and appearance across various operating systems.

11.3 Man-Hours Devoted to the Project

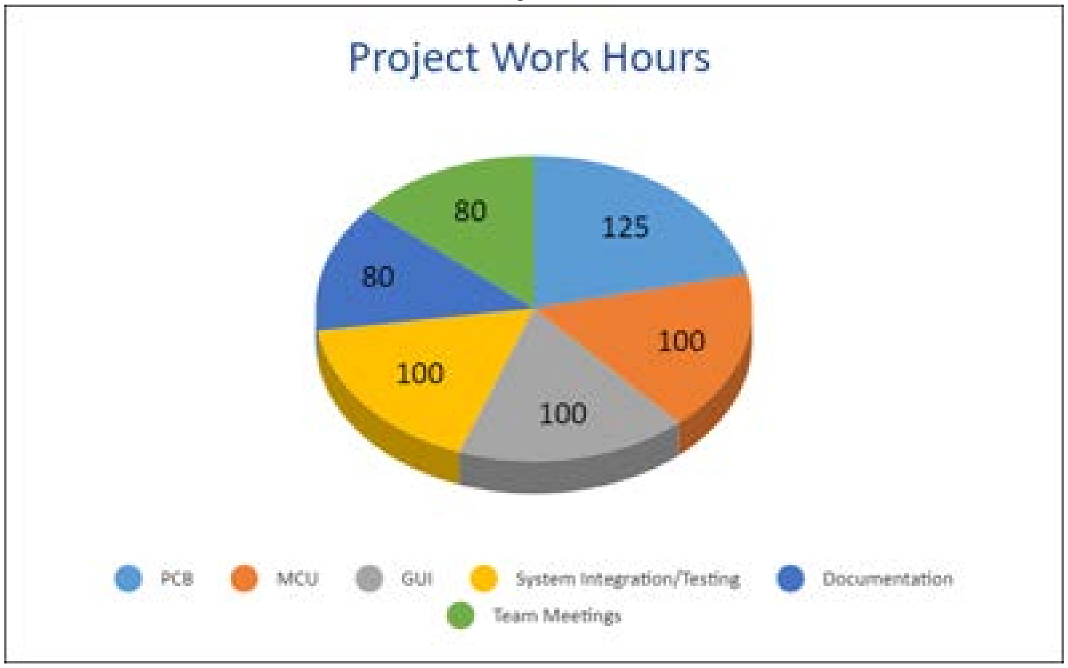
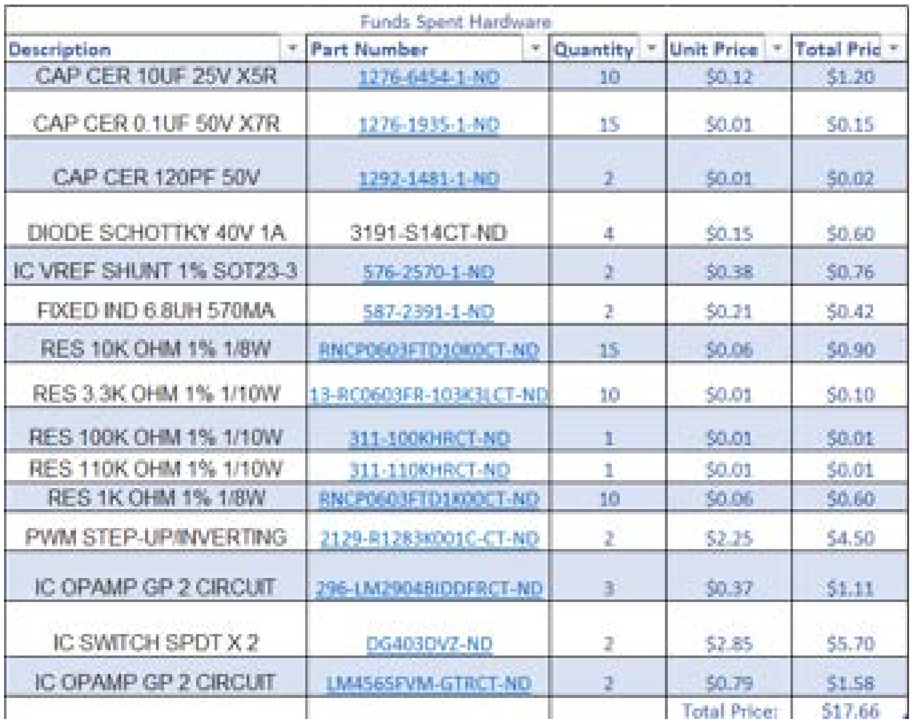


Figure 49: Project Work Hours Breakdown

Throughout the project, we spent about 585 total hours of time spread across the different areas of the project, but also on clerical functions like documentation and routine team meetings. If we were to evenly divide this amount among the six of us, each of us worked on the project roughly 90-100 hours each. This is consistent with the project being treated like a three credit class and the notion of putting in an hour of work outside of class for every hour spent inside of the class.

11.4 Funds Spent

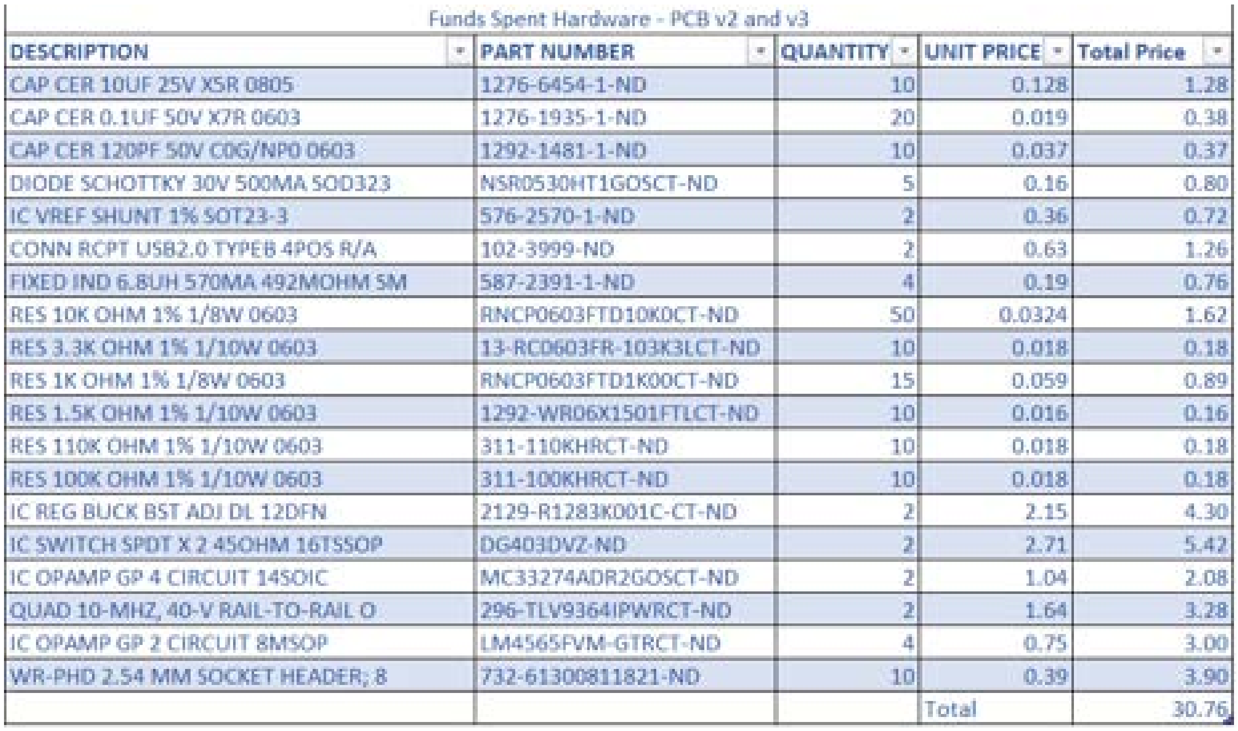
11.4.1 Front End - Version 1



Funds Spent Hardware				
Description	**Part Number**	**Quantity**	**Unit Price**	**Total Pric**
CAP CER 10UF 25V X5R	1276-6454-1-ND	10	$0.12	$1.20
CAP CER 0.1UF 50V X7R	1276-1935-1-ND	15	$0.01	$0.15
CAP CER 120PF 50V	1292-1481-1-ND	2	$0.01	$0.02
DIODE SCHOTTKY 40V 1A	3191-S14CT-ND	4	$0.15	$0.60
IC VREF SHUNT 1% SOT23-3	576-2570-1-ND	2	$0.38	$0.76
FIXED IND 6.8UH 570MA	587-2391-1-ND	2	$0.21	$0.42
RES 10K OHM 1% 1/8W	RNCP0603FTD10K0CT-ND	15	$0.06	$0.90
RES 3.3K OHM 1% 1/10W	13-RC0603FR-103K3LCT-ND	10	$0.01	$0.10
RES 100K OHM 1% 1/10W	311-100KHRCT-ND	1	$0.01	$0.01
RES 110K OHM 1% 1/10W	311-110KHRCT-ND	1	$0.01	$0.01
RES 1K OHM 1% 1/8W	RNCP0603FTD1K00CT-ND	10	$0.06	$0.60
PWM STEP-UP/INVERTING	2129-R1283K001C-CT-ND	2	$2.25	$4.50
IC OPAMP GP 2 CIRCUIT	296-LM2904BIDDFRCT-ND	3	$0.37	$1.11
IC SWITCH SPDT X 2	DG403DVZ-ND	2	$2.85	$5.70
IC OPAMP GP 2 CIRCUIT	LM4565FVM-GTRCT-ND	2	$0.79	$1.58
			Total Price:	$17.66

Figure 50: Front End Prototype Funds Spent Breakdown

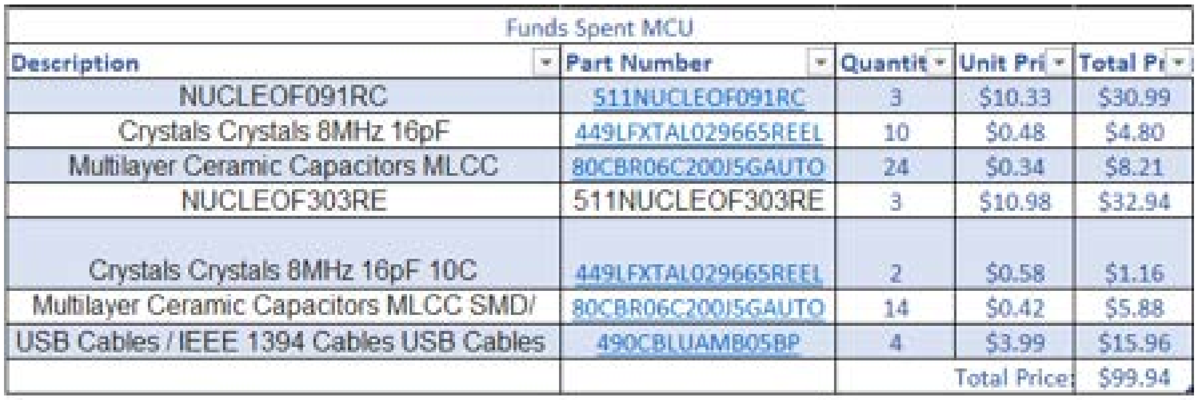
11.4.2 Front End - Versions 2 and 3



Funds Spent Hardware - PCB v2 and v3				
DESCRIPTION	**PART NUMBER**	**QUANTITY**	**UNIT PRICE**	**Total Price**
CAP CER 10UF 25V X5R 0805	1276-6454-1-ND	10	0.128	1.28
CAP CER 0.1UF 50V X7R 0603	1276-1935-1-ND	20	0.019	0.38
CAP CER 120PF 50V C0G/NP0 0603	1292-1481-1-ND	10	0.037	0.37
DIODE SCHOTTKY 30V 500MA SOD323	NSR0530HT1GOSCT-ND	5	0.16	0.80
IC VREF SHUNT 1% SOT23-3	576-2570-1-ND	2	0.36	0.72
CONN RCPT USB2.0 TYPEB 4POS R/A	102-3999-ND	2	0.63	1.26
FIXED IND 6.8UH 570MA 492MOHM SM	587-2391-1-ND	4	0.19	0.76
RES 10K OHM 1% 1/8W 0603	RNCP0603FTD10K0CT-ND	50	0.0324	1.62
RES 3.3K OHM 1% 1/10W 0603	13-RC0603FR-103K3LCT-ND	10	0.018	0.18
RES 1K OHM 1% 1/8W 0603	RNCP0603FTD1K00CT-ND	15	0.059	0.89
RES 1.5K OHM 1% 1/10W 0603	1292-WR06X1501FTLCT-ND	10	0.016	0.16
RES 110K OHM 1% 1/10W 0603	311-110KHRCT-ND	10	0.018	0.18
RES 100K OHM 1% 1/10W 0603	311-100KHRCT-ND	10	0.018	0.18
IC REG BUCK BST ADJ DL 12DFN	2129-R1283K001C-CT-ND	2	2.15	4.30
IC SWITCH SPDT X 2 45OHM 16TSSOP	DG403DVZ-ND	2	2.71	5.42
IC OPAMP GP 4 CIRCUIT 14SOIC	MC33274ADR2GOSCT-ND	2	1.04	2.08
QUAD 10-MHZ, 40-V RAIL-TO-RAIL O	296-TLV9364IPWRCT-ND	2	1.64	3.28
IC OPAMP GP 2 CIRCUIT 8MSOP	LM4565FVM-GTRCT-ND	4	0.75	3.00
WR-PHD 2.54 MM SOCKET HEADER; 8	732-61300811821-ND	10	0.39	3.90
			Total	30.76

Figure 51: Front End Versions 2 and 3 Funds Spent Breakdown

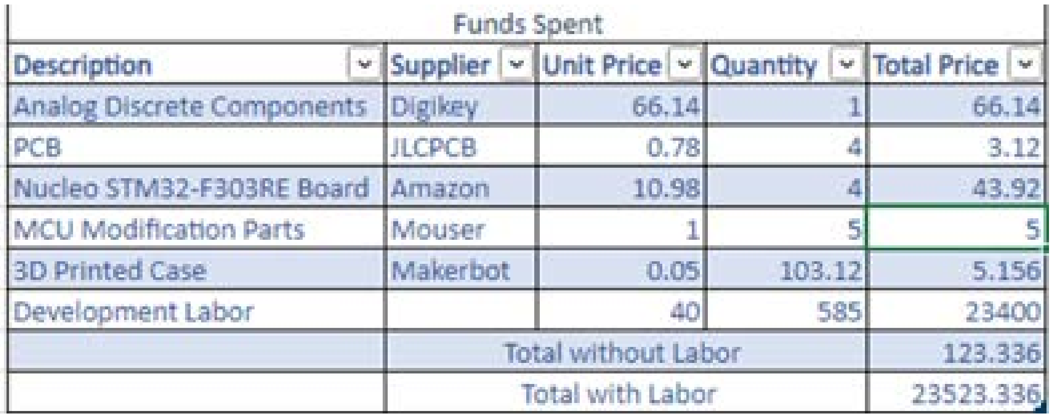
11.4.3 MCU



Funds Spent MCU				
Description	Part Number	Quantit	Unit Pri	Total Pr
NUCLEOF091RC	511NUCLEOF091RC	3	$10.33	$30.99
Crystals Crystals 8MHz 16pF	449LFXTAL029665REEL	10	$0.48	$4.80
Multilayer Ceramic Capacitors MLCC	80CBR06C200J5GAUTO	24	$0.34	$8.21
NUCLEOF303RE	511NUCLEOF303RE	3	$10.98	$32.94
Crystals Crystals 8MHz 16pF 10C	449LFXTAL029665REEL	2	$0.58	$1.16
Multilayer Ceramic Capacitors MLCC SMD/	80CBR06C200J5GAUTO	14	$0.42	$5.88
USB Cables / IEEE 1394 Cables USB Cables	490CBLUAMB05BP	4	$3.99	$15.96
			Total Price:	$99.94

Figure 52: MCU Funds Spent Breakdown

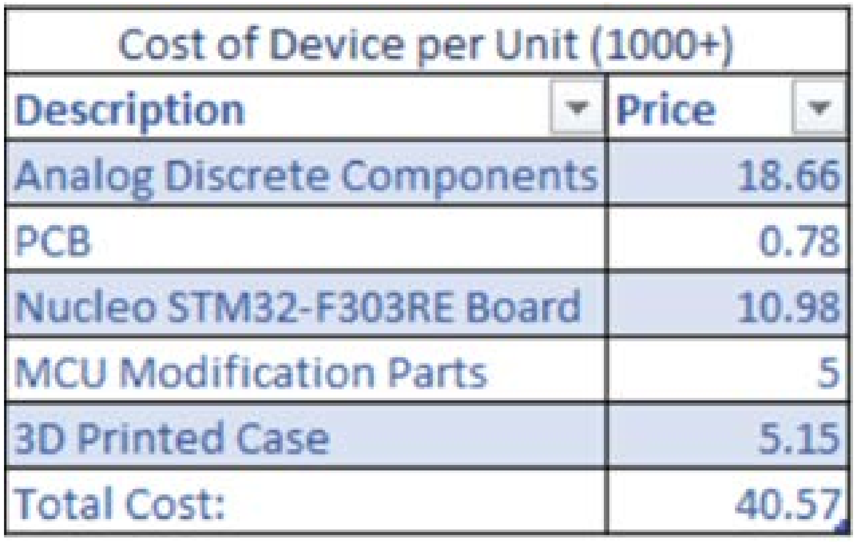
11.4.4 Total Funds Spent



Funds Spent				
Description	Supplier	Unit Price	Quantity	Total Price
Analog Discrete Components	Digikey	66.14	1	66.14
PCB	JLCPCB	0.78	4	3.12
Nucleo STM32-F303RE Board	Amazon	10.98	4	43.92
MCU Modification Parts	Mouser	1	5	5
3D Printed Case	Makerbot	0.05	103.12	5.156
Development Labor		40	585	23400
	Total without Labor			123.336
	Total with Labor			23523.336

Figure 53: Total Funds Spent Breakdown

11.4.5 Cost Per Unit



Cost of Device per Unit (1000+)	
Description	Price
Analog Discrete Components	18.66
PCB	0.78
Nucleo STM32-F303RE Board	10.98
MCU Modification Parts	5
3D Printed Case	5.15
Total Cost:	40.57

Figure 54: Cost Per Unit

11.5 Individual Team Member Contributions

11.5.1 Front End

- German Kuznetsov worked on designing the PCBs and subsequent revisions, also soldered the boards
- Bill Denham worked on designing the PCBs and the revisions, also soldered the boards

11.5.2 MCU

- James Schaeffler: Created the setup for pyserial USB communication between the PC and MCU, using a virtual com port.
- Yifei Gao: Wrote python code coverage test cases

11.5.3 GUI

- Hussain Zainal: Wrote the main interface that generates pre-defined and arbitrary waves for channels one and two in Python using the PyQt library
- Vikram Arunachalam: Wrote the wave drawer program for generating arbitrary, user-defined waveforms in Python using PyQtGraph

11.5.4 Project Management

- Bill Denham: Served as project manager, responsible for seeing all aspects of team dynamics, communication, and human resources

12. Lessons Learned

12.1 Additional Knowledge and Skills Learned

12.1.1 Front-End:

When developing custom printed circuit boards, we found that the creation of the footprints and the sizes of the components affected assembly of the boards. In some cases, where defined footprints were not defined, we had to create our own and that introduced a bit of sizing issues due to human error in measurement. In addition, we sought to compact our pcb as much as possible while trying to match the size of the MCU protoboard. The issue with this is that the more compact the PCB is, the harder it is to solder and place components without having to go back and manually straighten or fix components. Taking a more scientific approach to creating custom footprints and the tolerances of those footprints would go a lot further in the success of manufacturing future boards. However, when it comes to mass production, we would probably outsource PCB manufacturing and have a company provide the fabrication service, where there would likely be a quality assurance check before putting our device out to the end consumer.

12.1.2 MCU

The NUCLEO-F091RC microcontroller unit initially did not fulfill all the project requirements. The primary challenge encountered was the inadequate bandwidth of the onboard USB port. This limitation was effectively addressed by leveraging the GPIO pins to power an externally soldered USB port, thereby enhancing the board's capabilities. This experience imparted a valuable lesson about the potential for hardware modifications in microcontrollers to meet specific project needs.

Additionally, a significant learning point was the process of establishing effective communication and cooperation between the microcontroller unit and the front-end graphical user interface (GUI). Through proactive discussions and clear communication of expectations and requirements among the teams, we were able to streamline the project workflow. This collaboration not only facilitated a more efficient project development but also underscored the importance of inter-team communication in achieving project goals.

12.1.3 GUI

In the course of our project, we gained a wealth of knowledge, with lessons that extended far beyond the technical aspects of software development. One of the primary areas of learning was navigating and effectively utilizing the software development ecosystem. This involved mastering tools like GitHub and Git, which are pivotal for version control and collaborative development. We learned not only the technical skills to manage code changes and project versions but also the art of efficient communication within the team. This included setting clear expectations, systematically addressing requests, and maintaining a continuous and transparent dialogue about our progress and challenges.

The most significant lesson, however, revolved around the importance of modular coding and the intricacies of developing system-dependent software. We learned that modular coding ensures each part of the code can function independently yet integrate seamlessly. This approach was crucial in allowing us to adapt our application to different operating systems and hardware configurations. It taught us about the flexibility and foresight needed in design and development to anticipate and effectively handle potential variations and dependencies in software environments.

12.1.4 CAD Design

In the area of CAD design, we were able to take skills that were taught to us in ENGR-107, Introduction to engineering. In that class, we were taught the basics of engineering and the agile engineering design process, and also how to model 3D objects in Autodesk Inventor. Objects that can then be taken and turned into actual products via 3D printing. These skills were able to be translated into creating a custom case enclosure that makes our device look like it can be marketed like our competitor devices, the AD2 and ADALM2000. We learned that 3D modeling via CAD and 3D printing is a cheap and efficient way to realize a product before sending it off for the final manufacturing of the device.

12.2 Teaming Experience

Throughout the project, our team met twice a week to update the faculty supervisor and each other on progress made. We typically met with the faculty supervisor earlier in the week to update him on the previous week's accomplishments, while having working sessions together as a team, later in the week.

12.2.1 Project Sub-Teams:

For the project we split up into three smaller sub-teams, GUI, MCU, and PCB. Each sub-team or pair of students was responsible for development for its portion of the project, but could additionally help out in other areas depending on the need that week.

12.2.2 Team Communication/Dynamics:
One of the most difficult parts of this project was working through team communication and the dynamics of the different members of the team. It's well known that the stereotypical engineer is really adept at solving challenging problems, but not always the most skilled at working in teams, nor communicating their ideas in a constructive and polite way. Our team was no different, as we are all very well versed in various disciplines of the project, but for some of us it was a challenge communicating back to the others in a way that was both respectful and constructive. Ultimately, this proved to be one of the largest challenges that our team experienced as it caused resentment between different team members towards others. We probably spent an equal amount of time with team member disputes as we did doing technical development.

12.2.3 Project Management/Schedule
In order to keep the team and project on schedule, we utilized a few different forms of communication and project management tools. The first was our team Discord server which had separate channels for each area of the project, where discussion could occur asynchronously. It also contained means of uploading pictures and data for use in documentation. The next tool was our GitHub, which contained all source code for the GUI, MCU, and KiCad schematics of the PCB designs. We utilized Smartsheets for our Gantt chart and overall timeline for the project that could be referenced by any member of the team at any time. The Google Drive contained all deliverables like the design document, progress reports, and presentations. Lastly, were our weekly face-to-face meetings in the Engineering building with Dr. Kaps. With the face to face meetings, we could update Dr. Kaps on the project progress, obtain any constructive feedback from him, as well as work through any technical obstacles, or team dynamics issues.

13. References

[1] “ADALM2000,” *Evaluation Board | Analog Devices*, 09-Jan-2023. [Online]. Available: https://www.analog.com/en/design-center/evaluation-hardware-and-software/evaluation-boards-kits/adalm2000.html#eb-documentation. [Accessed: 26-Feb-2023].

[2] “Campus covid-19 data archive,” *George Mason University*. [Online]. Available: https://www.gmu.edu/campus-covid-19-data-archive. [Accessed: 26-Feb-2023].

[3] “Dataman 531 arbitrary waveform generator,” *Device Programmers & ISP Programming by Dataman Programmers*. [Online]. Available: https://www.dataman.com/dataman-531-arbitrary-waveform-generator.html. [Accessed: 26-Feb-2023].

[4] *Hantek2000 series - hantek electronic & your testing solution provider*. [Online]. Available: http://www.hantek.com/products/detail/13174. [Accessed: 26-Feb-2023].

[5] “Picoscope 2000 specifications,” *PicoScope 2000 Specifications | Pico Technology*. [Online]. Available: https://www.picotech.com/oscilloscope/2000/picoscope-2000-specifications. [Accessed: 26-Feb-2023].

[6] S. K, “Analog discovery 2,” *Analog Discovery 2 - Digilent Reference*. [Online]. Available: https://digilent.com/reference/test-and-measurement/analog-discovery-2/start. [Accessed: 26-Feb-2023].

[7] “Wiki,” *Reference Manual (Understanding the Internals) [Analog Devices Wiki]*. [Online]. Available: https://wiki.analog.com/university/tools/m2k/users/reference_manual. [Accessed: 26-Feb-2023].

14. Appendix A: Project Proposal (ECE 492)

ECE 492 Senior Advanced Design Project

Arbitrary Waveform Generator
Project Proposal

Team members: James Schaeffer
German Kuznetsov
Hussain Zainal
Vikram Arunachalam
Yifei Gao
William Denham

Faculty Advisor: Dr. Jens-Peter Kaps

ECE 492
Date of Submission: March 29, 2023

Contents:

1. Executive Summary

This project involves the design and construction of a USB Waveform generator device that meets the academic requirements of Electrical and Computer Engineering undergraduate students. The primary purpose of this is to design an affordable USB Waveform generator that can be readily available and at low cost, even in chip shortage conditions. The secondary purpose of this device should be to have high accuracy and precision. The design should have similar features to other portable electrical engineering test tools such as the ability to supply typical sine, square, and triangular waveforms; as well as an arbitrary waveform specified by the user.

The planning stages of the project will occur in ECE 492, where the team will be split up into three separate teams of the front-end circuit design, microcontroller (MCU), and Graphical User Interface. Individual team members will be tasked with researching and coming up with potential designs to better understand and develop a solution to their part of the project. They will come up with conceptual design sketches for the front-end, graphical user interface, and the microcontroller; then compare each design against the design criteria to select the best design. Design and implementation of a beta GUI will be developed in these stages, as well as the breadboarded version of the analog front-end. The implementation and testing stages will occur in ECE 493, where the individual components will come together into the completed system. The testing and implementation will determine whether design criteria has been met and thus the problem considered solved.

2. Problem Statement

2.1 Motivation and Identification of Need

For many years, undergraduate Electrical and Computer Engineering students have been required to purchase all-in-one oscilloscope tools such as the Analog Discovery 2 (AD2) and the Advanced Active Learning Module (ADALM 2000) to complete laboratory experiments. This was due to the inability to access state of the art bench equipment in the lab, during the COVID-19 pandemic. These devices allow students to conduct lab experiments anywhere, without the need to rely on the availability of the hardware labs. The pandemic has led to a resurgence in demand for these devices as students around the world are trying to get one for their academic studies. As a result of this increased demand, there is a chip shortage and prices continue to rise as chip supply fails to meet the demand.

Tools such as the ALAM2000 and Analog Discovery 2 cost around $236 and $399 respectively, which is not affordable to the average college student. In order to reduce the amount required to purchase one of these tools, this project aims to create an affordable usb waveform generator. This project will have a USB communication port with a two channel, 12-bit resolution Digital-Analog converter. The sampling rate will operate at 2 MHz and the sampling rate will be around 5 MSPS. This device will come with a custom design Graphical User Interface (GUI) where students can supply pre-programmed waveforms as well as user-specified waveforms to circuits. This new device will pair with the oscilloscope being designed and developed by another senior design team to create a new multifunction laboratory tool for use by undergraduate Electrical and Computer Engineering students.

2.2 Market Review

Device	Device Picture	Channels	Resolution (bit)	Sample Rate (MS/s)	Voltage Range (V)	Buffer Size (Sample/Channel)	Price ($)
Analog Discovery 2 (AD2)		2	14	100	5	16384	399
Advanced Active Learning Module (ADALM2000)		2	12	75	5	64k	236.25
Dataman 531 Arbitrary Waveform Generator		1	12	100	4.5	16384	695
Handtek2000 Handheld Oscilloscope + Waveform Generator + Multimeter		2	8	250	5	6k	174
PicoScope 2204A Benchtop Oscilloscope		2	12	100	2	4k	165

Table 1: Current Alternative Waveform Generators and Oscilloscope Multifunction Tools

3. Approach:

3.1 Problem Analysis

The problem requires the creation of a low cost AWG able to communicate with a GUI program running on the PC, and generate the waveforms requested by the user.

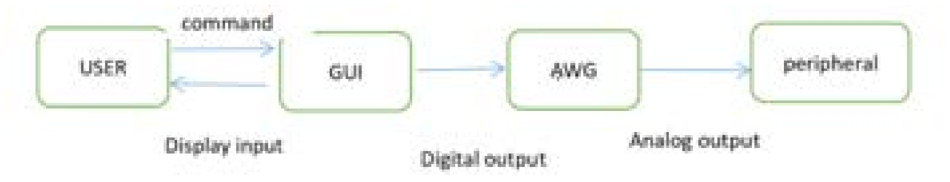


Diagram 1 : Problem Analysis Flowchart

The problem will require the usage of DACs to create the waveforms, as well as op-amps to condition the waveforms to the correct output voltage levels. Furthermore, this problem will require two pieces of software that can communicate via USB - the GUI software running on the PC, and the MCU firmware responsible for running the device and generating the waves. Also required will be a pair of regulators to generate a dual polarity voltage rail for the op-amps.

3.2 Our Approach

One way to reduce cost is to reduce complexity. By selecting a MCU with built in peripherals like DACs and USB, we can reduce the cost of our device. However, the MCU should still be powerful enough so that the AWG remains a viable option when it comes to being used in an academic setting.

3.2.1 Hardware Approach

We settled on the STM32-F303RE MCU as our microcontroller of use. Despite its price tag of 10$, the STM32 includes two DACs, DMA, and USB, meaning it is very well suited for our use. The two DACs can be used to produce the two waveforms channels, and the DMA can be used to supply the DACs with data without load on the CPU. The USB interface could allow the MCU to communicate with a computer without the need for an external USB<->UART converter, which is needed for microcontrollers like the AVR (used in Arduino).

The MCU can only generate a wave in the 0-3.3V range. In order to condition the signal to the right levels, we will need to use op-amps to apply a gain to the signal and shift the signal to the right levels. By carefully designing the circuit, we can minimize parts count and keep cost down.

To supply the aforementioned op-amps, we will need to generate a positive and negative voltage rail to power the op-amps. This will require a boost and a buck converter, to generate the positive and negative voltage rails respectively. Considering that our output signal should be in the -5 to 5V range, and that typical (non rail-to-rail) op-amps cannot reach their supply voltage, the negative supply rail should be less than -5V and the positive supply rail should be over 5V.

3.2.2 Graphical User Interface Approach

Our graphical user interface must be capable of allowing the user to specify the waveform they want, And transmitting these settings to the user. There are 4 main specifications for the waveform:

- Frequency of the waveform.
- Amplitude.
- Offset.
- Shape (triangle, square, sign, or user defined)

Furthermore, we could extend the functionality to allow the user to enter duty cycle for square waves, and phase when there are more than two waves. These options will be configured via graphical user interface elements such as numerical text boxes or drop down menus, and a preview of the output waveform will be displayed in the GUI. To simplify coding, we chose to use the GUI library PyQt. It can be used via python, and works with any of the major three operating systems.

3.3 Alternative Approaches

There are not a lot of alternative approaches when it comes to the opamp and power supply of the AWG hardware. However, there are some alternative MCUs to use. One example is the RP2040. It has a ARM Cortex CPU with higher clock speeds than the STM32. It is also cheaper. However, it does not come with an internal DAC and would require an external DAC. Another approach would be to use a Direct Digital Synthesis IC such as the AD9837 instead of a DAC. This approach would completely alleviate the task of generating the waveform from the MCU, allowing us to use a very anemic MCU. However, a chip like the AD9837 is expensive at $6.77, and only has one channel.

3.4 Background Knowledge:

3.4.1 Microcontroller Unit (MCU)

MCU is an intelligent semiconductor IC that consists of Arithmetic and Logic Unit(ALU), Register set, Control Unit, Internal bus, and Interface to System bus which is to connect to memory and I/O ports. The MCU is used in lots of different types of applications which include washing machines, radio, and controllers. MCU is similar but less sophisticated than System on a Chip. The first MCU was developed in 1971 which is called TMS1000. For MCU characteristics, it has a low price for high-volume applications. It has lower clock frequencies when compared to DSPS and it is up to 100MHz. Also, low power consumption and limited memory. For our project, we use the NUCLEO-F091RC as our boardand the microprocessor on it is STM32F091RC. For the MCU we use, its frequency is up to 48 MHz. It has 128 to 256 Kbytes of Flash memory. 32K bytes of SRAM with HW parity. It has a 12-channel DMA controller. One 12-bit, 1.0us ADC, and its conversion range is 0 to 3.6V.

Figure 1: STM32 – Nucleo – F303RE

3.4.2 Graphical User Interface (GUI)

We will use PyQtGraph(Figure 2) which is a GUI module to design the GUI for this project. PyQt supports both C++ and python and it is widely used for creating large-scale GUI-based programs. It provides creators with lots of different pre-built designs which helps creators save time. The QtWidgets has graphical components and related classes, such as buttons, windows, status bars, bitmaps, colors or fonts. Also, PyQt can run on Windows, Linux, Mac OS, and various UNIX platforms. For our project, on the user

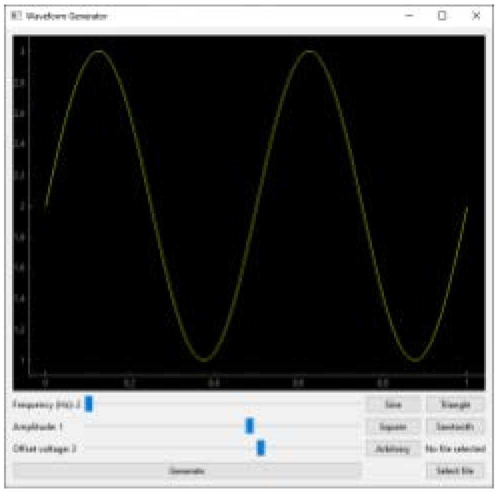


Figure 2: Prototype PyQtGraph

side, they can choose a hand drawn arbitrary waveform, sine, square, triangle and sawtooth.

3.4.3 Digital to Analog Converter

A D/A converter takes the precise number and converts it into a physical quantity. Usually, the digital signal is a finite-precision time series data and the analog signal is a continually varying physical signal.

$$v_o = \frac{v_R}{(2^{N-1})} * D$$

v_o = voltage output

v_R = reference voltage

N = number of bits

D = Digital Number Input

Figure 3: DAC Output Voltage

3.4.4 Operational Amplifier (OpAmp)

An operational amplifier is an integrated circuit that can amplify electrical signals. It has 2 input pins and 1 output pin. Usually, an operational amplifier isn't used alone but connected to other circuits' components. For 1 op-amp circuit, it can be a non-inverting amplifier circuit, inverting amplifier circuit, voltage follower, etc. For this project, the circuit we design needs to let the output signal be large enough and also can drive a 20mA load.

Figure 4 : LMH6506 Variable Gain Amplifier

3.5 Project Requirements Specification:

3.5.1 Mission Requirements:

The project shall develop an affordable USB-powered arbitrary waveform generator that   has a low cost requirement for undergraduate electrical and computer engineering students to perform laboratory experiments anywhere. The device shall utilize a custom PCB design for the analog front end and microcontroller that can interface to a Graphical User Interface to display the results.

3.5.2 Operational Requirements:

Input/output requirements:

- The device should have 2 analog output channels.

External Interface Requirements:

- Communication of the device is from the USB

Function requirements

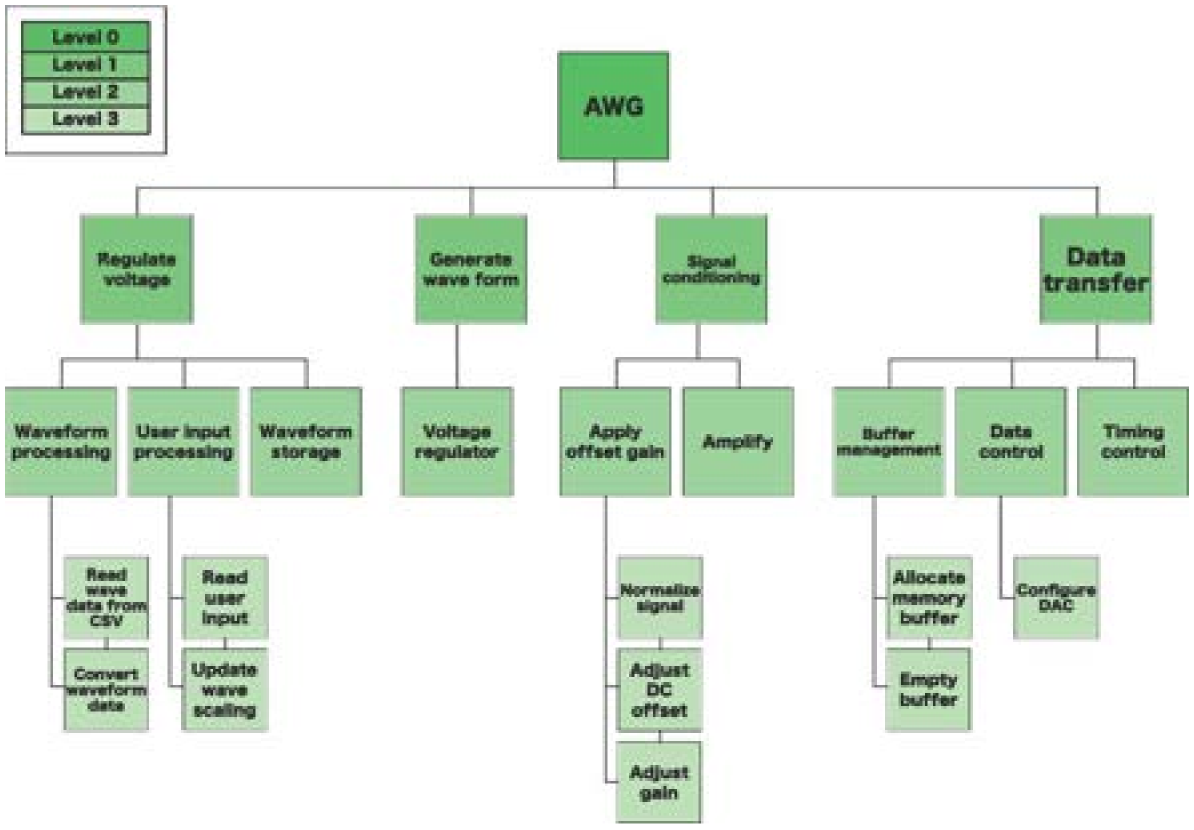
- The device **shall** be controlled via SCPI
- The device **shall** create arbitrary waveforms, triangular, rectangular, sine
- The output bandwidth **should** be around 2MHz.
- The Sample Rate **will** be around 5 MSPS.
- The output **shall** be peak-to-peak 10V, adjustable +-2.5V.
- The output **will** be able to drive a 20 mA load and 12 bits resolution.
- The output **shall** display the wave, frequency, amplitude and offset selected.

3.5.3 Technology and System-Wide Requirements:

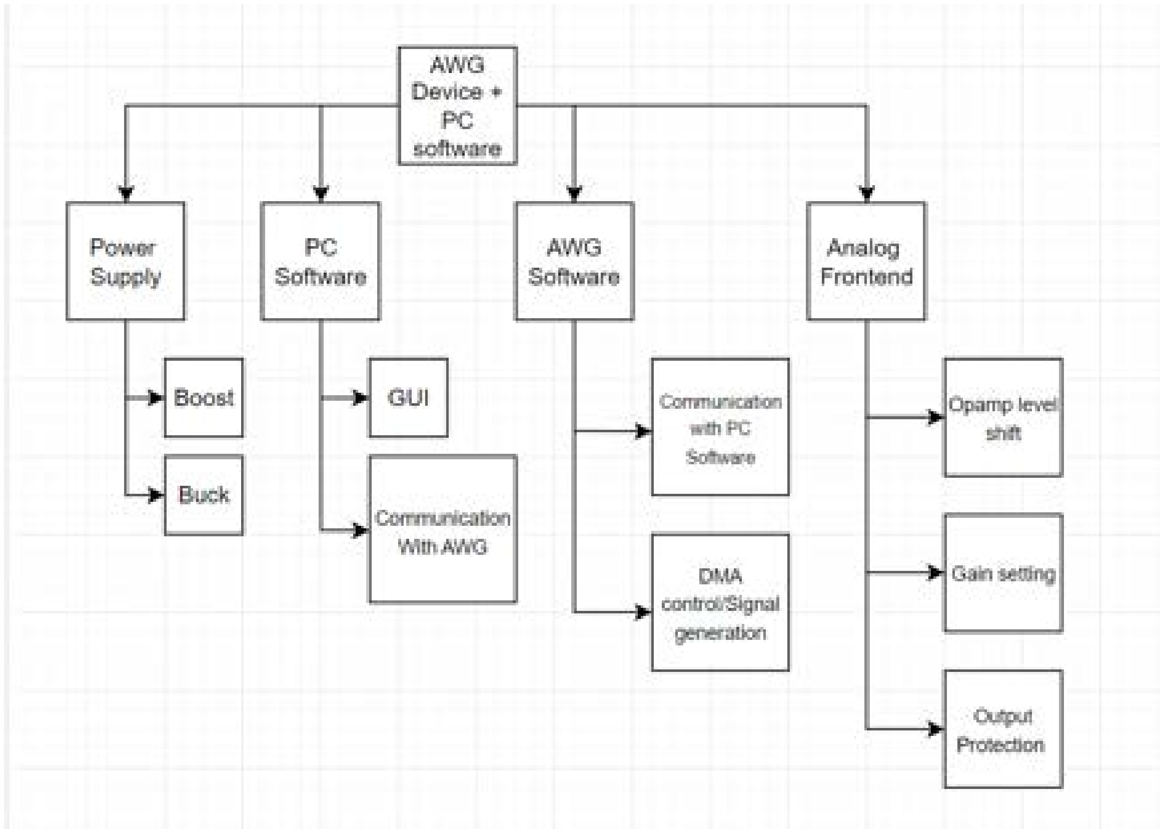
In order to run the device, the SMT32 IDE and Python based GUI **will** be able to work on most operating systems, i.e. Windows, Mac OS, and Linux.

4. System Design:

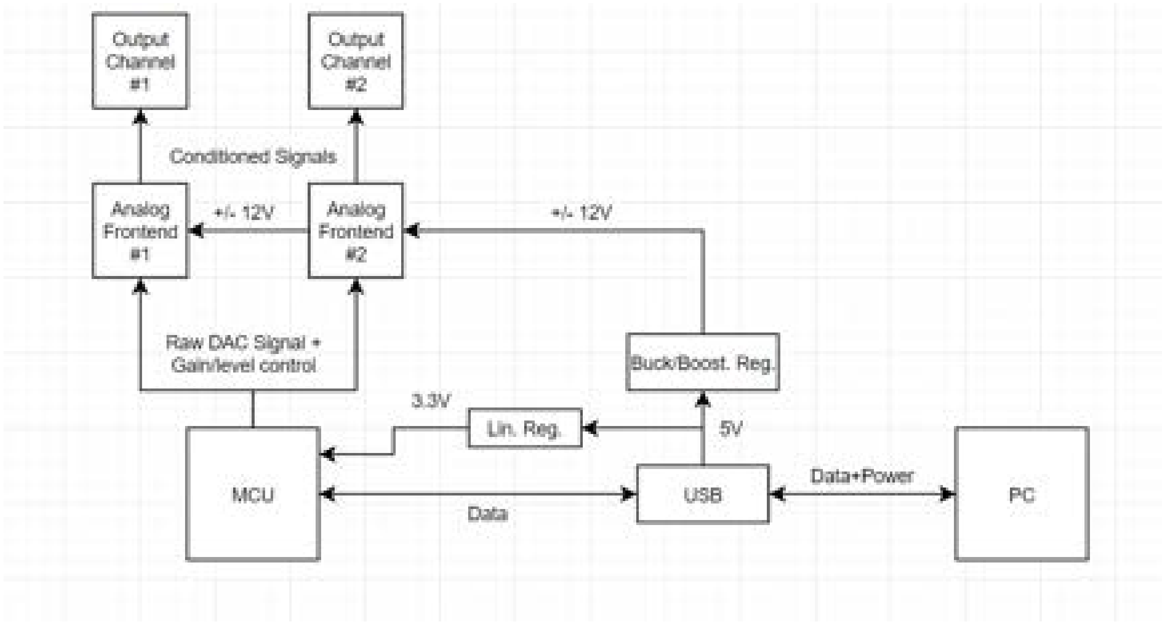
4.1 Functional Decomposition



4.2 Physical Architecture



4.3 System Architecture



5. Preliminary Experimental Plan

5.1 Preliminary Experiment and Testing Plan (ECE 492)

Testing for the prototype and final designs will be split based on the three core components of the function generator: analog front end, MCU programming, and GUI. Each of which will need to be tested individually as well as collectively once the components are integrated.

5.1.1 Analog Front End

The analog front end should be thoroughly tested for correct operation of the gain and offset functionality.

Test #1: Gain

1. Input a sine wave to the analog front end with an amplitude of 1V.
2. Go through all possible gain settings.
3. Measure the output amplitude with each setting.

   This test will ensure the gain functionality is working correctly, and allow us to measure the error.

Test #2: Offset

4. Input a DC voltage to the analog front end.
5. Sweep the offset from the max to min value.
6. Measure the output DC voltage.
7. Repeat at different Gain levels.

   This test will ensure the offset is correctly added to the input signal, and the output range goes from the max to min value while the offset goes from the max to min value. This test will also allow us to measure the error.

Test #3: Bandwidth

This test will ensure the front end has enough bandwidth to pass signals at our max output frequency without losing significant amplitude.

1. Perform a frequency sweep on the input of the analog front end.
2. Measure the output level
3. Repeat at different Gain levels or with different offsets.

5.1.2 MCU

The Nucleo-F303RE board is a popular development board used for prototyping and testing embedded systems. When designing a waveform generator, it is important to thoroughly test the hardware and software before moving to production. Testing with the Nucleo-F091RC board is an essential step in this process as it allows the team to verify that the waveform generator is functioning as intended and to identify and address any potential issues. This IDE is new to all of the members, so preliminary testing is required.

Test #1:
The first test will be used as preliminary testing for the STM32 Nucleo board. Verify the jumper positions, install the USB driver, and connect the board to a PC to ensure that the LED lights are functioning correctly when the button is pressed. Additionally, the team will use the available demonstration and software examples to test the software functionality and compatibility. Any issues with jumper positions, LED lights, or software functionality could be identified and corrected before moving to production.

Test #2:

1. Hardware setup: Connect an LED and a current-limiting resistor to the GPIO output pin of the DAC. Connect the ground of the LED to the ground pin of the DAC.
2. Code setup: Write a simple code that sets the GPIO output pin of the DAC to a known logic level (e.g., high or low), and continuously updates the output value.
3. Verification: Run the code on the microcontroller and observe the LED. The LED should turn on or off, depending on the logic level set in the code.
4. Repeat: Repeat the test with different output values to verify that the GPIO output of the DAC can be set accurately and consistently.

This test can help ensure that the GPIO output of the DAC is functioning correctly and can be set as expected, which is important for a range of applications, such as controlling external devices or triggering other circuit components. Any issues with the GPIO output of the DAC could be identified and corrected before moving to production.

Test #3:

1. Hardware setup: Connect the Nucleo board to a PC via USB using a Type-A to Mini-B cable. Connect via ST-LINK to the Nucleo board using the SWD (Serial Wire Debug) interface.
2. Software setup: In STM32CubeIDE, create a project that sets up a simple data transfer routine, such as sending a string of characters from the microcontroller to the PC over the ST-Link interface. In Python, use the Pyserial library to set up a serial communication object and read the data being sent by the microcontroller over the ST-Link interface.
3. Verification: Build and upload the project to the Nucleo board using STM32CubeIDE. Verify that the data is being transmitted and received correctly, and that there are no errors or data loss during the transfer.
4. Repeat: Repeat the test with different data transfer rates and different data formats to verify that the ST-Link communication is reliable and consistent.

This test can help ensure that the ST-Link communication between the PC and Nucleo board is functioning correctly and can transfer data reliably, which is important for many applications. Any issues with data transfer or communication can be identified and corrected before moving to production.

5.1.3 GUI

The GUI is how the user interacts with the system specifically when it comes to selecting types of waveforms and setting values for criteria such as frequency, offset, and amplitude. Thus, it is important to confirm that the GUI is both displaying desired changes as well as transmitting the proper information to the MCU.

Test #1: The first test is meant to verify that the GUI is properly plotting functions based on user input to ensure that the user has accurate visuals at all times.

1. Design several examples of settings for each of the standard waveform shapes.
2. Input values into both the GUI and the waveform generator on the AD2
3. Verify that the waveforms match relative to the respective axis.

Test #2:

1. Design several waveform examples for testing or use the waveforms from test 1.
2. For each waveform record and display the data that is to be transmitted to the Nucleo board.
3. Verify that the data received matches the sent data for each of the example waveforms.

5.2 Testing Plan for Prototype with integration(ECE 492/493)

The purpose of this testing is to verify that individual components are functionally operational when integrated together.

5.2.1 Analog Front End

Test:

1. Connect the Nucleo-F303RE board to the PC using a USB cable.
2. Using either Waveforms Live or PyQtGraph, generate a digital signal with a specified amplitude and frequency.
3. Adjust the gain and offset settings on the analog front end of the board using the GUI.
4. Measure the output signal using an oscilloscope.
5. Record the output signal amplitude and frequency for each gain and offset setting.
6. Analyze the results to determine the gain and offset error and ensure that the gain and offset functionality is working correctly.
7. Repeat the test with different digital signal amplitudes and frequencies to ensure the accuracy and reliability of the analog front end.

Overall, this test would allow you to use a GUI to adjust the gain and offset settings on the Nucleo-F303RE board and test the functionality of the analog front end with a digital signal generated by the PC. This would ensure that the board is functioning correctly and provide accurate and reliable measurements for your project.

5.2.2 MCU/GUI

Test:

1. Use Pyserial to send the digital signal from the PC's GUI to the Nucleo-F303RE board.
2. Adjust the gain and offset settings on the analog front end of the board using the MCU firmware.
3. Measure the output signal using an oscilloscope or other measurement equipment.
4. Record the output signal amplitude and frequency for each gain and offset setting.
5. Analyze the results to determine the gain and offset error and ensure that the gain and offset functionality is working correctly.
6. Repeat the test with different digital signal amplitudes and frequencies to ensure the accuracy and reliability of the analog front end.

The purpose of this test is to verify Pyserial's ability to accurately send/receive data between the user controlled GUI interface and MCU's IDE. Subsequent testing will include removing STM32CubeIDE's involvement with the User to create a simpler User experience.

Using DMA to transfer data between memory and I/O devices, the CPU can perform other tasks while the data transfer is taking place. With the DAC, DMA can be used to offload the task of transferring data from the MCU to the DAC, reducing the MCU's resource load and improving system performance.The main testing criteria for the MCU is to ensure system resources are kept low while also producing accurate results.

5.2.3 PCB

Designing a custom PCB can be challenging and requires careful consideration of component placement, trace routing, trace widths, and trace lengths to prevent common issues during testing. The trace width affects the resistance and current carrying capacity of the circuit, while the trace length affects the propagation delay of the signal and can introduce issues such as signal skew or reflection. During testing, it's important to inspect the board for physical defects, verify the power supply, and check the resistance between different points on the PCB. Taking the time to optimize the circuit and ensure proper component placement, trace routing, and appropriate trace widths and lengths can help prevent issues during testing and ensure the successful operation of the board.

6. Preliminary Project Plan

6.1 Overview

This project will be completed over the course of two semesters and will be split into two phases (ECE 492 and 493). The project itself can be broken down into three core categories that are integral to the design and functionality of the finished product. These divisions are the analog front end architecture and PCB design, microcontroller unit programming, and graphical user interface programming. Designated team members will be working in parallel to complete required tasks. It is expected that by the end of ECE 492 that we will have a functional prototype on a breadboard, so that PCB development can begin at the start of ECE 493 at the latest.

6.1.1 Hardware design

We will use generic op amps in the analog front end to condition the signal from the MCU. We will have to design the circuit to correctly apply the gain and offset voltage. We will also have to design the circuit for the buck and boost regulators. Finally, we will have to design and populate a PCB for this circuit.

6.1.2 MCU

For this project, we shall use the NUCLEO -F303RE development board with STM32F446RE MCU. This particular board was selected by a team working on an oscilloscope design under similar constraints. Given that integration of their oscilloscope design and our arbitrary waveform generator is an eventual possibility it makes sense to use the same board as we will still have access to the two digital to analog converters necessary.

6.1.3 GUI Programming

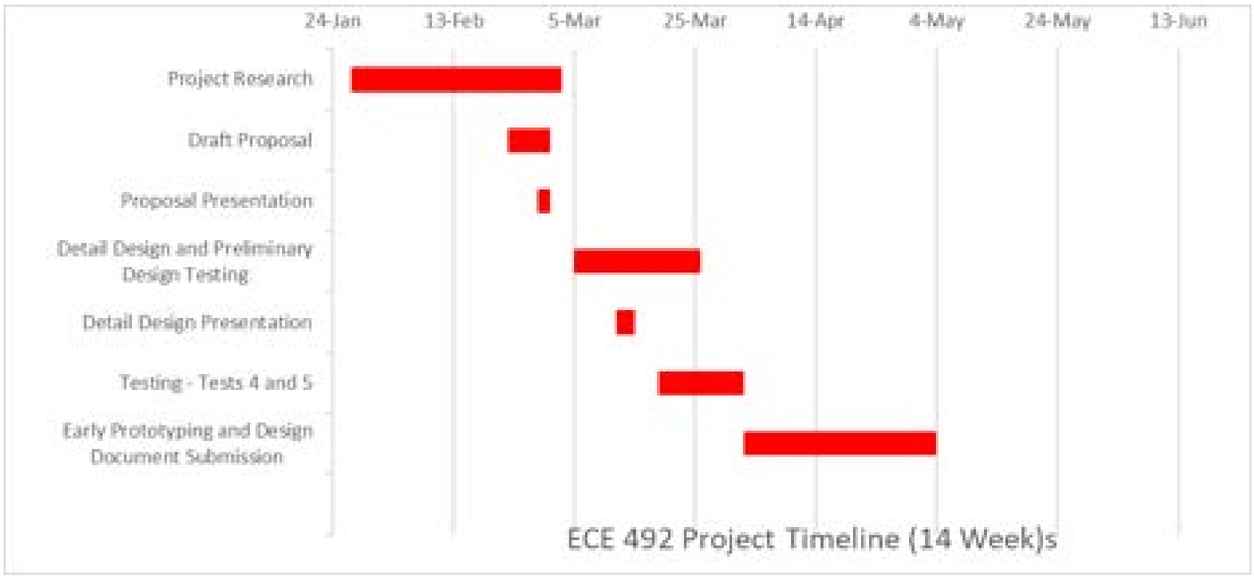
The graphical user interface will be designed using PyQtGraph, a scientific graphics and GUI library for python. This library can be used to create the GUI as well as plot waveforms with data being provided.

6.2: Allocation of Responsibilities

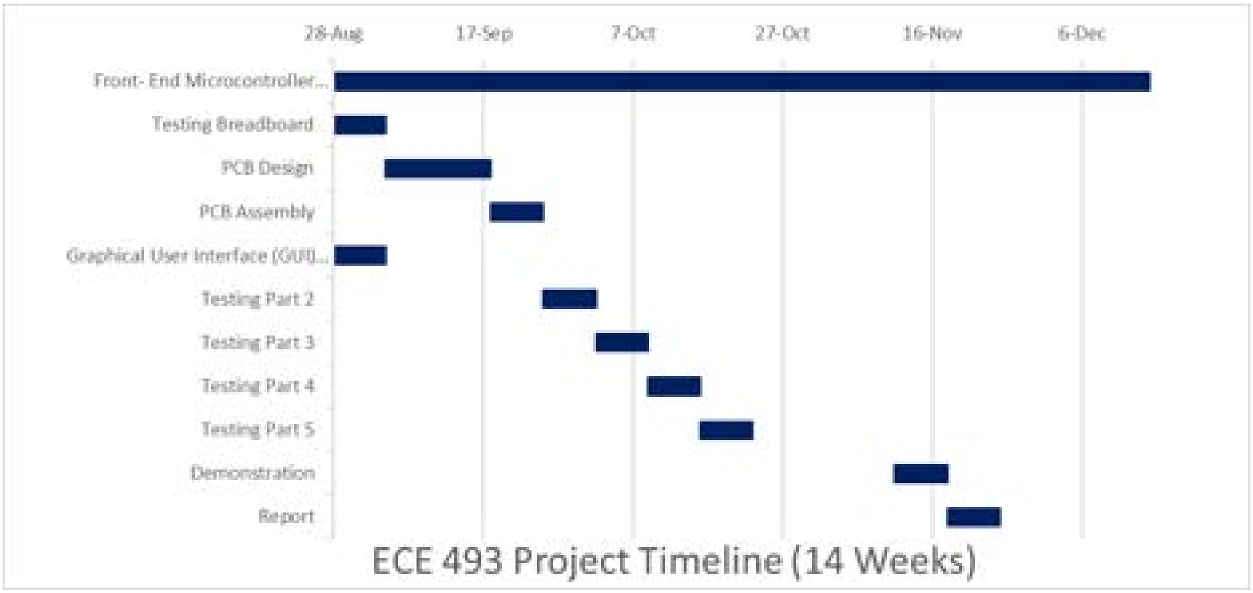
The individual tasks are divided based upon background, skills, and general interest in the project area.

- Yifei Gao / James Schaeffer : Microcontroller Unit (MCU) Programming
- Vikram Arunachalam / Hussain Zainal : Graphical User Interface (GUI) Programming
- German Kuznetsov / William Denham : Hardware Design

6.3 Detail Project Plan Timeline ECE 492 (14 Weeks)



6.4 Detail Project Plan Timeline ECE 493 (14 Weeks)



7. Potential Problems

7.1 Required Technical Skill Set

After identifying the core aspects of this project we are aware that there are required technical skills that we will need to develop a functioning prototype and final design. The following set of skills and knowledge areas will be critical during this project:

- PCB Design, Analog Design
- Programming Languages: Python , C, Matlab
- Device and programming debugging
- GUI Design using PyQtGraph

7.2 Hardware design

When designing a PCB, mistakes may cause the need for revisions, which may waste precious time. Furthermore, no one in our group has SMD soldering experience so we may have trouble assembling the board at first. Another potential problem is that in order to achieve accurate results, we may need to use more precise resistors, which may drive up the cost.

7.3 Microcontroller Unit (MCU)

Given that we are unfamiliar with the MCU being used, there is a lot of documentation to go through in order to resolve programming issues. Fewer members of the team are fully proficient in C as well, so we can expect that much more time will need to be allocated to programming and debugging code for the MCU. The hardware specifications of the MCU satisfy the current operational requirements of the project, but should we need to resort to an alternative we may face potential issues with stock and delivery timing. Furthermore, we are unsure if the MCU will be able to generate the waveform at the specified frequency while also managing the USB connection and other tasks.

7.4 Graphical User Interface (GUI)

The PyQtGraph library being used in the design of the GUI supports both Python and C++. While no members of the team have used this library before we all have experience with python. Those working primarily on the GUI have experience with working with documented python libraries.

8. References

[1] “ADALM2000,” *Evaluation Board | Analog Devices*, 09-Jan-2023. [Online]. Available: https://www.analog.com/en/design-center/evaluation-hardware-and-software/evaluation-boards-kits/adalm2000.html#eb-documentation. [Accessed: 26-Feb-2023].

[2] “Campus covid-19 data archive,” *George Mason University*. [Online]. Available: https://www.gmu.edu/campus-covid-19-data-archive. [Accessed: 26-Feb-2023].

[3] “Dataman 531 arbitrary waveform generator,” *Device Programmers & ISP Programming by Dataman Programmers*. [Online]. Available: https://www.dataman.com/dataman-531-arbitrary-waveform-generator.html. [Accessed: 26-Feb-2023].

[4] *Hantek2000 series - hantek electronic & your testing solution provider*. [Online]. Available: http://www.hantek.com/products/detail/13174. [Accessed: 26-Feb-2023].

[5] “Picoscope 2000 specifications,” *PicoScope 2000 Specifications | Pico Technology*. [Online]. Available: https://www.picotech.com/oscilloscope/2000/picoscope-2000-specifications. [Accessed: 26-Feb-2023].

[6] S. K, “Analog discovery 2,” *Analog Discovery 2 - Digilent Reference*. [Online]. Available: https://digilent.com/reference/test-and-measurement/analog-discovery-2/start. [Accessed: 26-Feb-2023].

[7] “Wiki,” *Reference Manual (Understanding the Internals) [Analog Devices Wiki]*. [Online]. Available: https://wiki.analog.com/university/tools/m2k/users/reference_manual. [Accessed: 26-Feb-2023].

15. Appendix B: Design Document (ECE 493)

ECE 492 Senior Advanced Design Project

Arbitrary Waveform Generator

Design Document

Team members: James Schaeffler
German Kuznetsov
Hussain Zainal
Vikram Arunachalam
Yifei Gao
William Denham

Faculty Advisor: Dr. Jens-Peter Kaps

ECE 492
Date of Submission: May 5, 2023

Contents:

1. Executive Summary

This project involves the design and construction of a USB Waveform generator device that meets the academic requirements of Electrical and Computer Engineering undergraduate students. The primary purpose of this is to design an affordable USB Waveform generator that can be readily available and at low cost, even in chip shortage conditions. The secondary purpose of this device should be to have high accuracy and precision. The design should have similar features to other portable electrical engineering test tools such as the ability to supply typical sine, square, and triangular waveforms; as well as an arbitrary waveform specified by the user.

The planning stages of the project will occur in ECE 492, where the team will be split up into three separate teams of the front-end circuit design, microcontroller (MCU), and Graphical User Interface. Individual team members will be tasked with researching and coming up with potential designs to better understand and develop a solution to their part of the project. They will come up with conceptual design sketches for the front-end, graphical user interface, and the microcontroller; then compare each design against the design criteria to select the best design. Design and implementation of a beta GUI will be developed in these stages, as well as the breadboarded version of the analog front-end. The implementation and testing stages will occur in ECE 493, where the individual components will come together into the completed system. The testing and implementation will determine whether design criteria has been met and thus the problem considered solved.

2. Problem Statement

2.1 Motivation and Identification of Need

For many years, undergraduate Electrical and Computer Engineering students have been required to purchase all-in-one oscilloscope tools such as the Analog Discovery 2 (AD2) and the Advanced Active Learning Module (ADALM 2000) to complete laboratory experiments. This was due to the inability to access state of the art bench equipment in the lab, during the COVID-19 pandemic. These devices allow students to conduct lab experiments anywhere, without the need to rely on the availability of the hardware labs. The pandemic has led to a resurgence in demand for these devices as students around the world are trying to get one for their academic studies. As a result of this increased demand, there is a chip shortage and prices continue to rise as chip supply fails to meet the demand.

Tools such as the ALAM2000 and Analog Discovery 2 cost around $236 and $399 respectively, which is not affordable to the average college student. In order to reduce the amount required to purchase one of these tools, this project aims to create an affordable usb waveform generator. This project will have a USB communication port with a two channel, 12-bit resolution Digital-Analog converter. The sampling rate will operate at 2 MHz and the sampling rate will be around 5 MSPS. This device will come with a custom design Graphical User Interface (GUI) where students can supply pre-programmed waveforms as well as user-specified waveforms to circuits. This new device will pair with the oscilloscope being designed and developed by another senior design team to create a new multifunction laboratory tool for use by undergraduate Electrical and Computer Engineering students.

2.2 Market Review

Device	Device Picture	Channels	Resolution (bit)	Sample Rate (MS/s)	Voltage Range (V)	Buffer Size (Sample/Channel)	Price ($)
Analog Discovery 2 (AD2)		2	14	100	5	16384	399
Advanced Active Learning Module (ADALM2000)		2	12	75	5	64k	236.25
Dataman 531 Arbitrary Waveform Generator		1	12	100	4.5	16384	695
Handtek2000 Handheld Oscilloscope + Waveform Generator + Multimeter		2	8	250	5	6k	174
PicoScope 2204A Benchtop Oscilloscope		2	12	100	2	4k	165

Table 1: Current Alternative Waveform Generators and Oscilloscope Multifunction Tools

3. Approach:

3.1 Problem Analysis

The problem requires the creation of a low cost AWG able to communicate with a GUI program running on the PC, and generate the waveforms requested by the user.

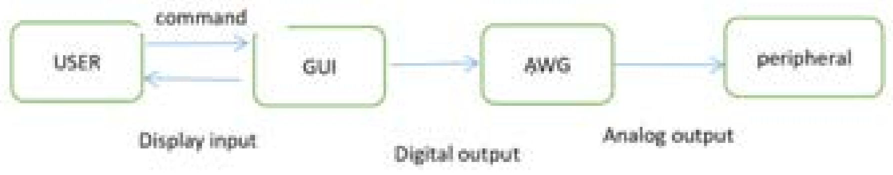


Diagram 1 : Problem Analysis Flowchart

The problem will require the usage of DACs to create the waveforms, as well as op-amps to condition the waveforms to the correct output voltage levels. Furthermore, this problem will require two pieces of software that can communicate via USB - the GUI software running on the PC, and the MCU firmware responsible for running the device and generating the waves. Also required will be a pair of regulators to generate a dual polarity voltage rail for the op-amps.

3.2 Our Approach

One way to reduce cost is to reduce complexity. By selecting a MCU with built in peripherals like DACs and USB, we can reduce the cost of our device. However, the MCU should still be powerful enough so that the AWG remains a viable option when it comes to being used in an academic setting.

3.2.1 Hardware Approach

We settled on the STM32-F303RE MCU as our microcontroller of use. Despite its price tag of 10$, the STM32 includes two DACs, DMA, and USB, meaning it is very well suited for our use. The two DACs can be used to produce the two waveforms channels, and the DMA can be used to supply the DACs with data without load on the CPU. The USB interface could allow the MCU to communicate with a computer without the need for an external USB<->UART converter, which is needed for microcontrollers like the AVR (used in Arduino).

The MCU can only generate a wave in the 0-3.3V range. In order to condition the signal to the right levels, we will need to use op-amps to apply a gain to the signal and shift the signal to the right levels. By carefully designing the circuit, we can minimize parts count and keep cost down.

To supply the aforementioned op-amps, we will need to generate a positive and negative voltage rail to power the op-amps. This will require a boost and a buck converter, to generate the positive and negative voltage rails respectively. Considering that our output signal should be in the -5 to 5V range, and that typical (non rail-to-rail) op-amps cannot reach their supply voltage, the negative supply rail should be less than -5V and the positive supply rail should be over 5V.

3.2.2 Graphical User Interface Approach

Our graphical user interface must be capable of allowing the user to specify the waveform they want, And transmitting these settings to the user. There are 4 main specifications for the waveform:

- Frequency of the waveform.
- Amplitude.
- Offset.
- Shape (triangle, square, sign, or user defined)

Furthermore, we could extend the functionality to allow the user to enter duty cycle for square waves, and phase when there are more than two waves. These options will be configured via graphical user interface elements such as numerical text boxes or drop down menus, and a preview of the output waveform will be displayed in the GUI. To simplify coding, we chose to use the GUI library PyQt. It can be used via python, and works with any of the major three operating systems.

3.3 Alternative Approaches

There are not a lot of alternative approaches when it comes to the opamp and power supply of the AWG hardware. However, there are some alternative MCUs to use. One example is the RP2040. It has a ARM Cortex CPU with higher clock speeds than the STM32. It is also cheaper. However, it does not come with an internal DAC and would require an external DAC. Another approach would be to use a Direct Digital Synthesis IC such as the AD9837 instead of a DAC. This approach would completely alleviate the task of generating the waveform from the MCU, allowing us to use a very anemic MCU. However, a chip like the AD9837 is expensive at $6.77, and only has one channel.

3.4 Background Knowledge:

3.4.1 Microcontroller Unit (MCU)

MCU is an intelligent semiconductor IC that consists of Arithmetic and Logic Unit(ALU), Register set, Control Unit, Internal bus, and Interface to System bus which is to connect to memory and I/O ports. The MCU is used in lots of different types of applications which include washing machines, radio, and controllers. MCU is similar but less sophisticated than System on a Chip. The first MCU was developed in 1971 which is called TMS1000. For MCU characteristics, it has a low price for high-volume applications. It has lower clock frequencies when compared to DSPS and it is up to 100MHz. Also, low power consumption and limited memory. For our project, we use the NUCLEO-F091RC as our boardand the microprocessor on it is STM32F091RC. For the MCU we use, its frequency is up to 48 MHz. It has 128 to 256 Kbytes of Flash memory. 32K bytes of SRAM with HW parity. It has a 12-channel DMA controller. One 12-bit, 1.0us ADC, and its conversion range is 0 to 3.6V.

Figure 1: STM32 – Nucleo – F303RE

3.4.2 Graphical User Interface (GUI)

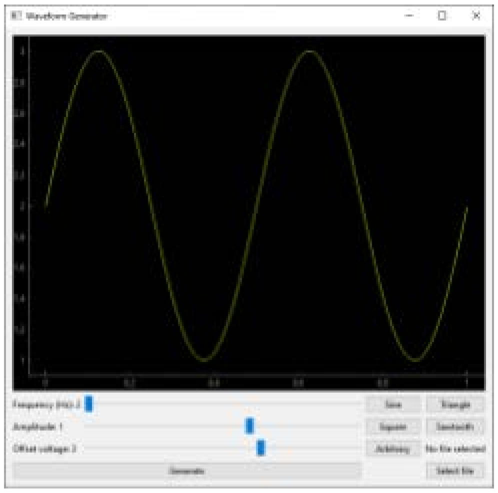


Figure 2: Prototype PyQtGraph

We will use PyQtGraph(Figure 2) which is a GUI module to design the GUI for this project. PyQt supports both C++ and python and it is widely used for creating large-scale GUI-based programs. It provides creators with lots of different pre-built designs which helps creators save time. The QtWidgets has graphical components and related classes, such as buttons, windows, status bars, bitmaps, colors or fonts. Also, PyQt can run on Windows, Linux, Mac OS, and various

UNIX platforms. For our project, on the user side, they can choose a hand drawn arbitrary waveform, sine, square, triangle and sawtooth.

3.4.3 Digital to Analog Converter

A D/A converter takes the precise number and converts it into a physical quantity. Usually, the digital signal is a finite-precision time series data and the analog signal is a continually varying physical signal.

$$v_o = \frac{v_R}{(2^{N-1})} * D$$

v_o = voltage output

v_R = reference voltage

N = number of bits

D = Digital Number Input

Figure 3: DAC Output Voltage

3.4.4 Operational Amplifier (OpAmp)

An operational amplifier is an integrated circuit that can amplify electrical signals. It has 2 input pins and 1 output pin. Usually, an operational amplifier isn't used alone but connected to other circuits' components. For 1 op-amp circuit, it can be a non-inverting amplifier circuit, inverting amplifier circuit, voltage follower, etc. For this project, the circuit we design needs to let the output signal be large enough and also can drive a 20mA load.

Figure 4 : LMH6506 Variable Gain Amplifier

3.5 Project Requirements Specification:

3.5.1 Mission Requirements:

The project shall develop an affordable USB-powered arbitrary waveform generator that   has a low cost requirement for undergraduate electrical and computer engineering students to perform laboratory experiments anywhere. The device shall utilize a custom PCB design for the analog front end and microcontroller that can interface to a Graphical User Interface to display the results.

3.5.2 Operational Requirements:

Input/output requirements:

- The device should have 2 analog output channels.

External Interface Requirements:

- Communication of the device is from the USB

Function requirements

- The device **shall** be controlled via SCPI
- The device **shall** create arbitrary waveforms, triangular, rectangular, sine
- The output bandwidth **should** be around 2MHz.
- The Sample Rate **will** be around 5 MSPS.
- The output **shall** be peak-to-peak 10V, adjustable +-2.5V.
- The output **will** be able to drive a 20 mA load and 12 bits resolution.
- The output **shall** display the wave, frequency, amplitude and offset selected.

3.5.3 Technology and System-Wide Requirements:

In order to run the device, the SMT32 IDE and Python based GUI **will** be able to work on most operating systems, i.e. Windows, Mac OS, and Linux.

4. Experimental Plan and Selection of Evaluation of Criteria

4.1 Overview

This project will be completed over the course of two semesters and will be split into two phases (ECE 492 and 493). The project itself can be broken down into three core categories that are integral to the design and functionality of the finished product. These divisions are the analog front end architecture and PCB design, microcontroller unit programming, and graphical user interface programming. Designated team members will be working in parallel to complete required tasks. It is expected that by the end of ECE 492 that we will have a functional prototype on a breadboard, so that PCB development can begin at the start of ECE 493 at the latest.

4.1.1 Hardware design

We will use generic op amps in the analog front end to condition the signal from the MCU. We will have to design the circuit to correctly apply the gain and offset voltage. We will also have to design the circuit for the buck and boost regulators. Finally, we will have to design and populate a PCB for this circuit.

4.1.2 MCU

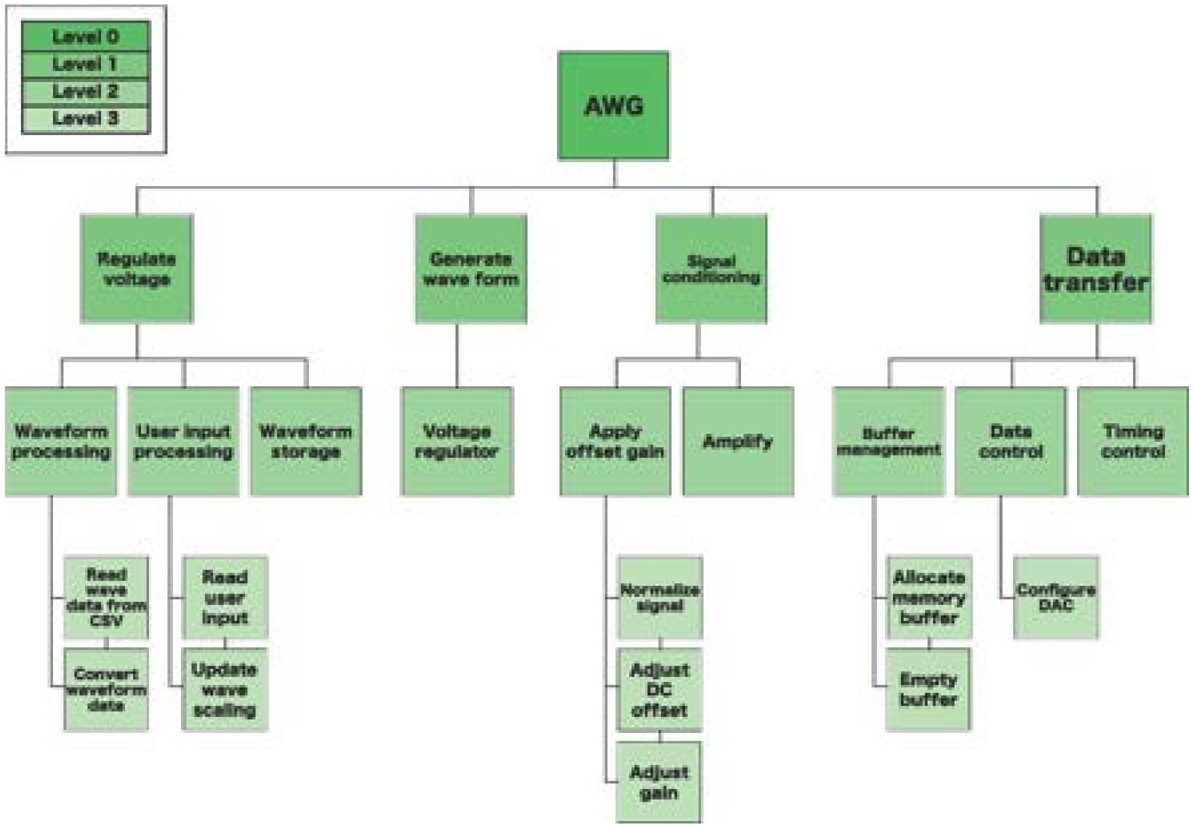
For this project, we shall use the NUCLEO -F303RE development board with STM32F446RE MCU. This particular board was selected by a team working on an oscilloscope design under similar constraints. Given that integration of their oscilloscope design and our arbitrary waveform generator is an eventual possibility it makes sense to use the same board as we will still have access to the two digital to analog converters necessary.

4.1.3 GUI Programming

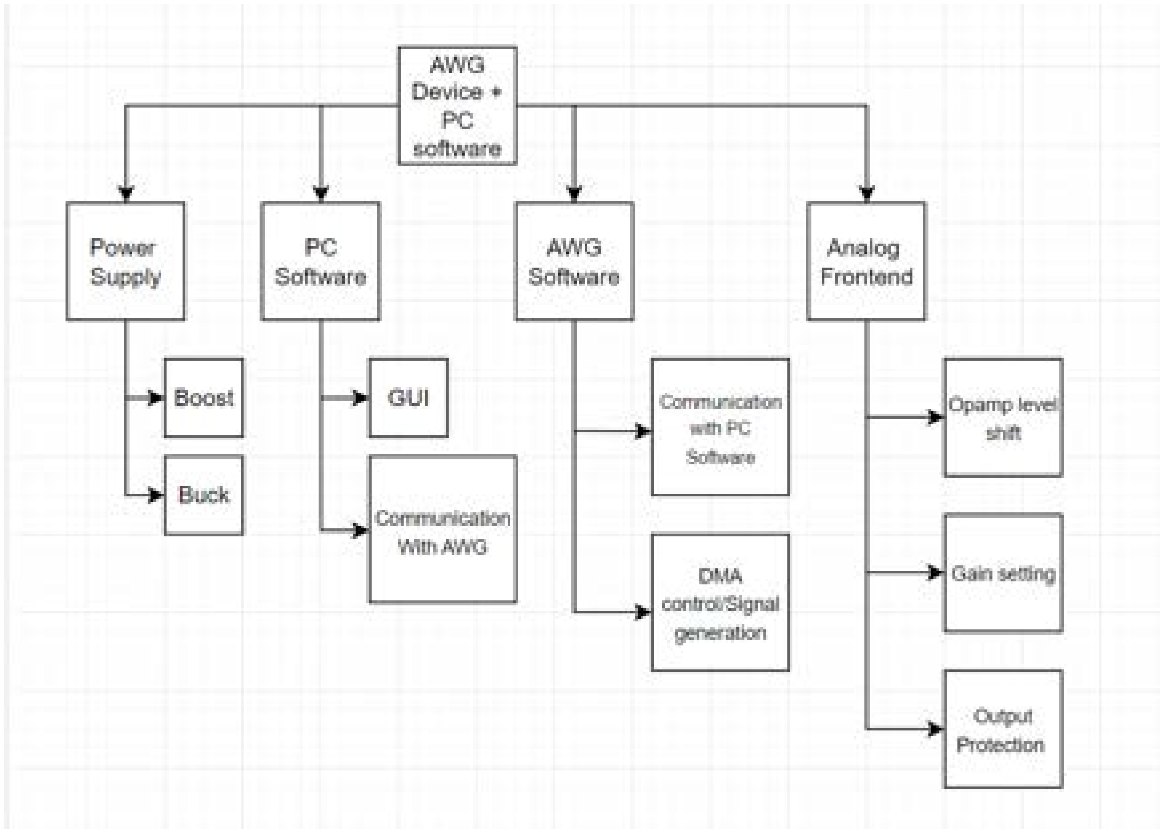
The graphical user interface will be designed using PyQtGraph, a scientific graphics and GUI library for python. This library can be used to create the GUI as well as plot waveforms with data being provided. Users must be able to enter certain data to define the waveform in order to be sent to the MCU. Upon clicking the generate button, a visual display of the waveform must be displayed on the GUI application.

5. System Design:

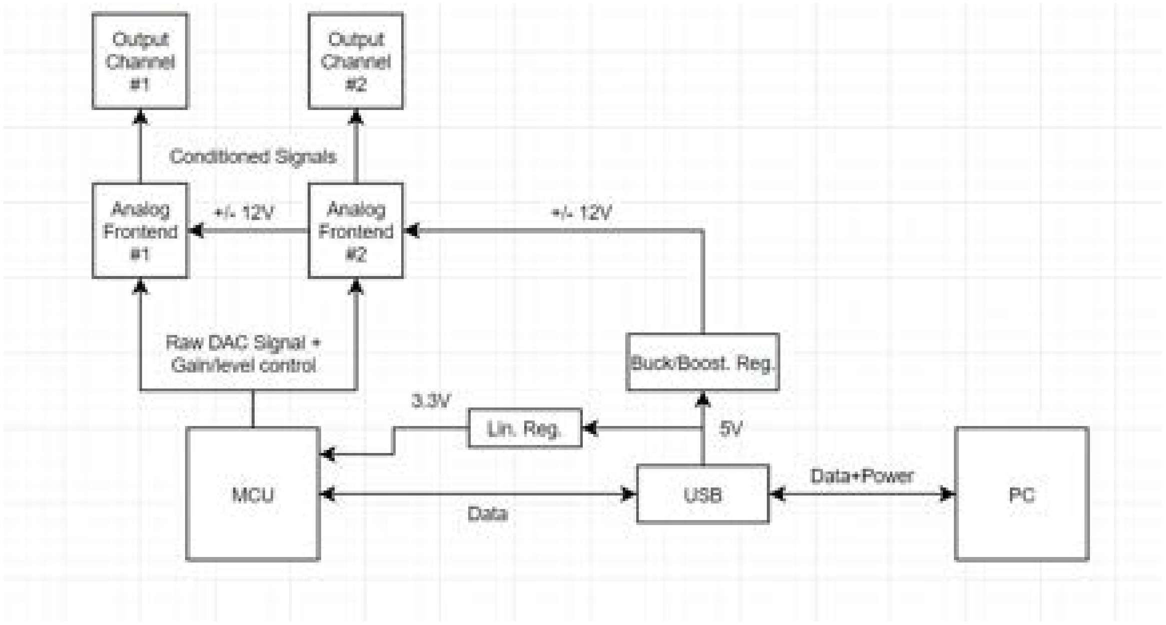
5.1 Functional Decomposition



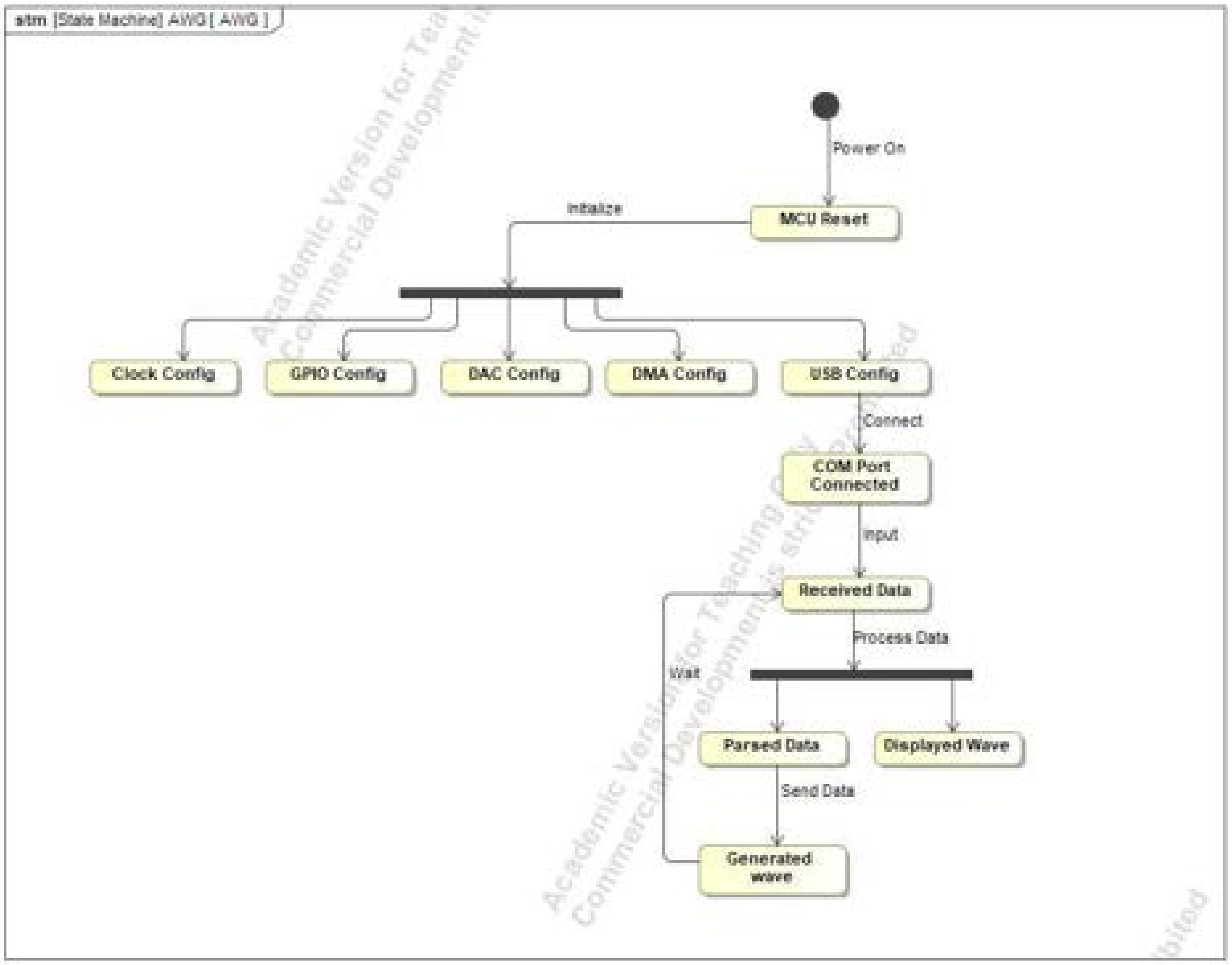
5.2 Physical Architecture



5.3 System Architecture

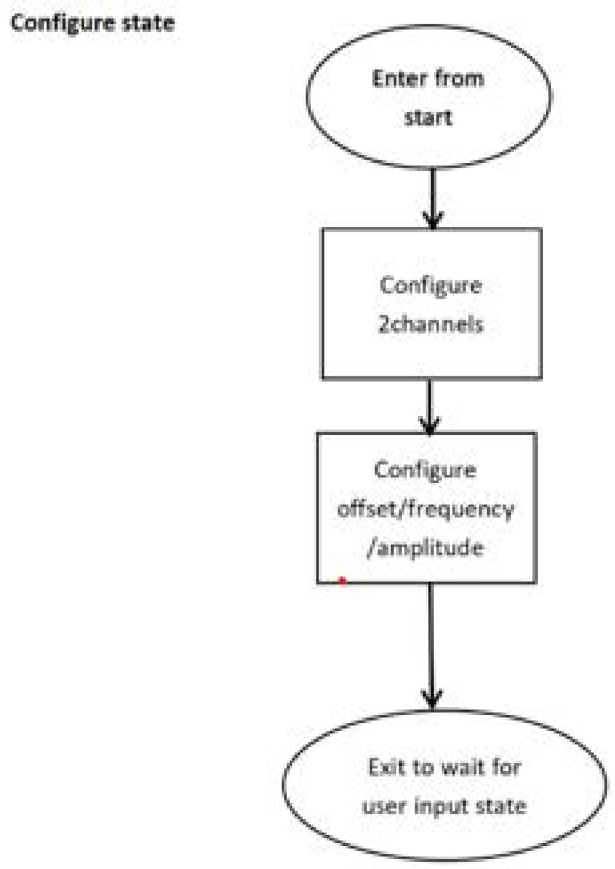


5.4 State Machine Diagram for Arbitrary Waveform

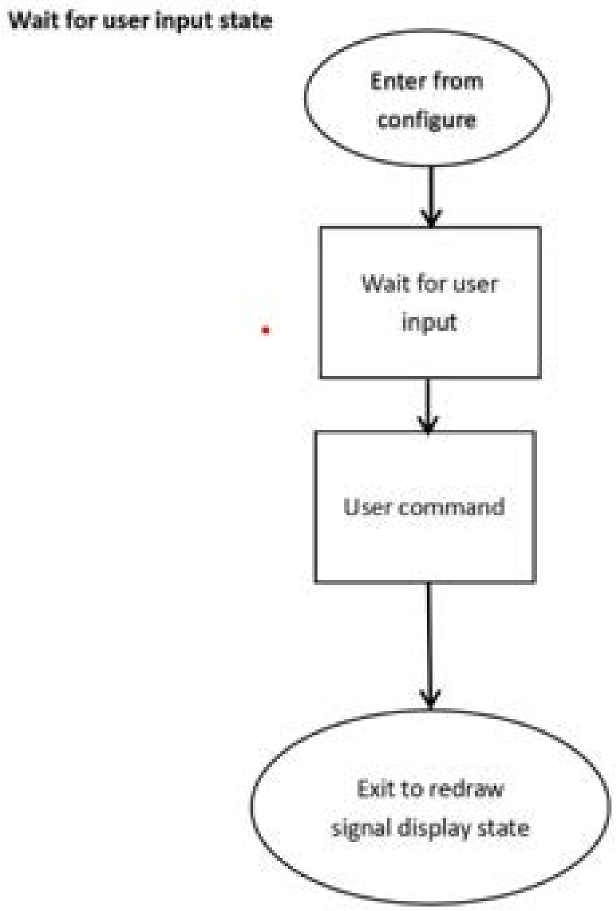


5.5 Flowcharts

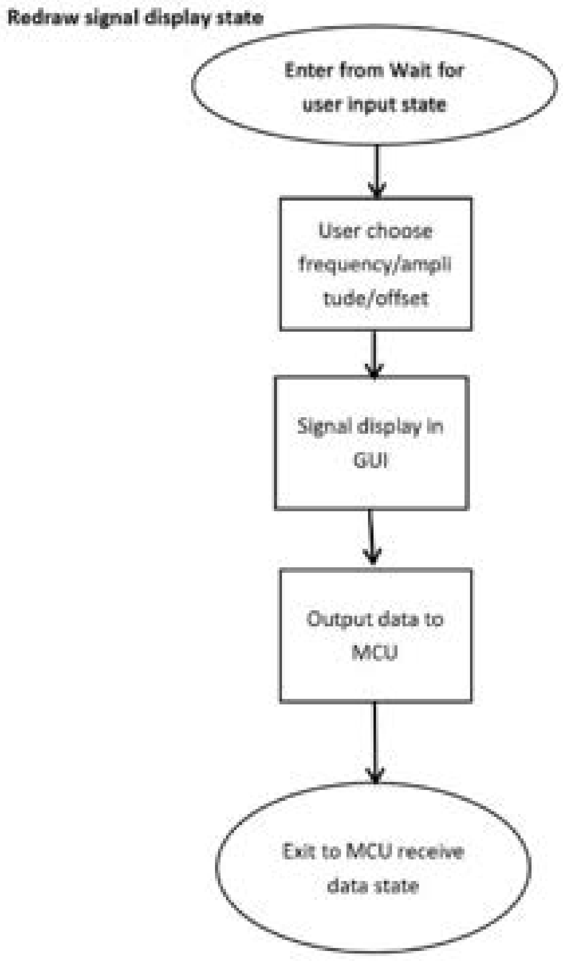
5.5.1 Configure Scenario


Configure state
Enter from start
Configure 2channels
Configure offset/frequency /amplitude
Exit to wait for user input state

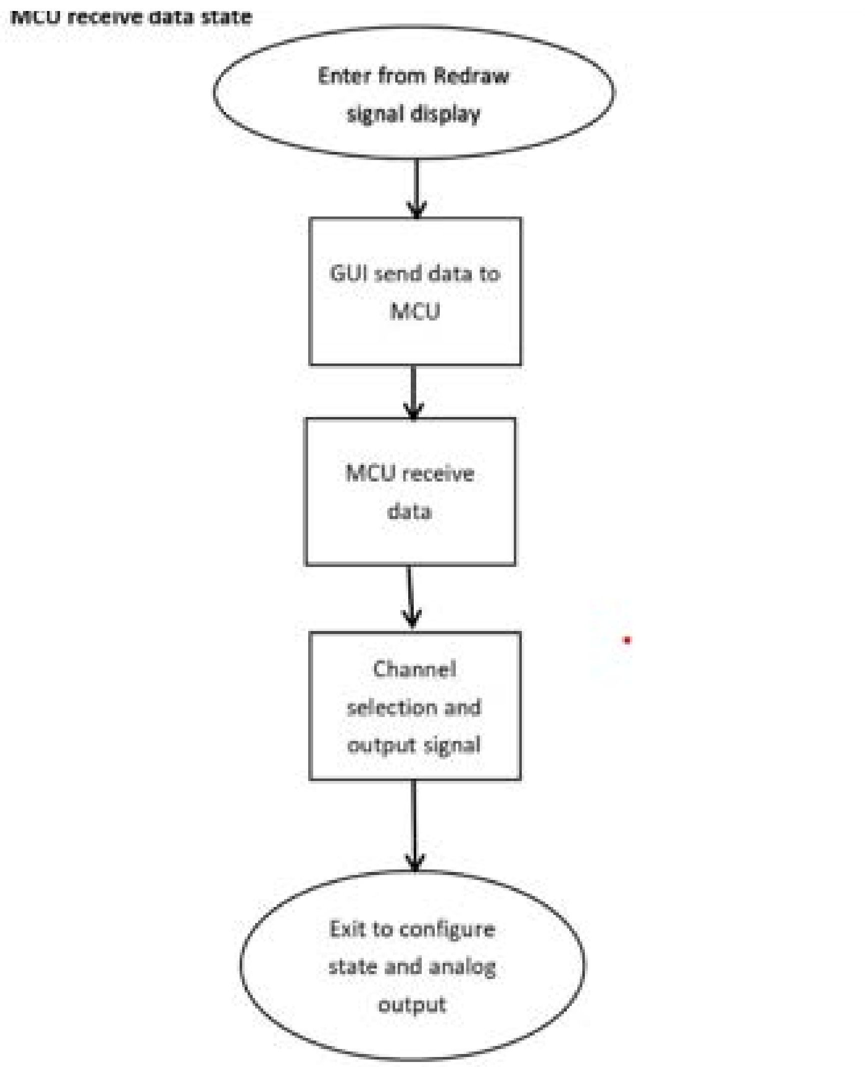
5.5.2 Wait for User Input Scenario


Wait for user input state
Enter from configure
Wait for user input
User command
Exit to redraw signal display state

5.5.3 Redraw Signal Display Scenario


Redraw signal display state
Enter from Wait for user input state
User choose frequency/amplitude/offset
Signal display in GUI
Output data to MCU
Exit to MCU receive data state

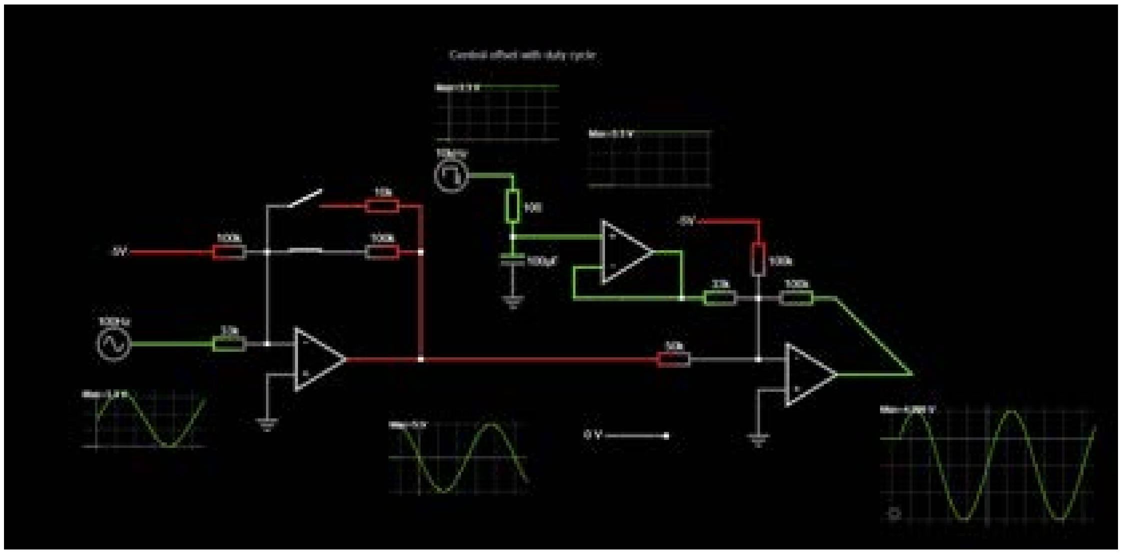
5.5.4 MCU Receive Data Scenario


MCU receive data state
Enter from Redraw signal display
GUI send data to MCU
MCU receive data
Channel selection and output signal
Exit to configure state and analog output

6. Detail Design

6.1 Analog Front End Architecture

6.1.1 Falstad Circuit Simulation



6.1.2 Op Amp Selection

Analog Hardware Selection - OpAmp Bandwidth

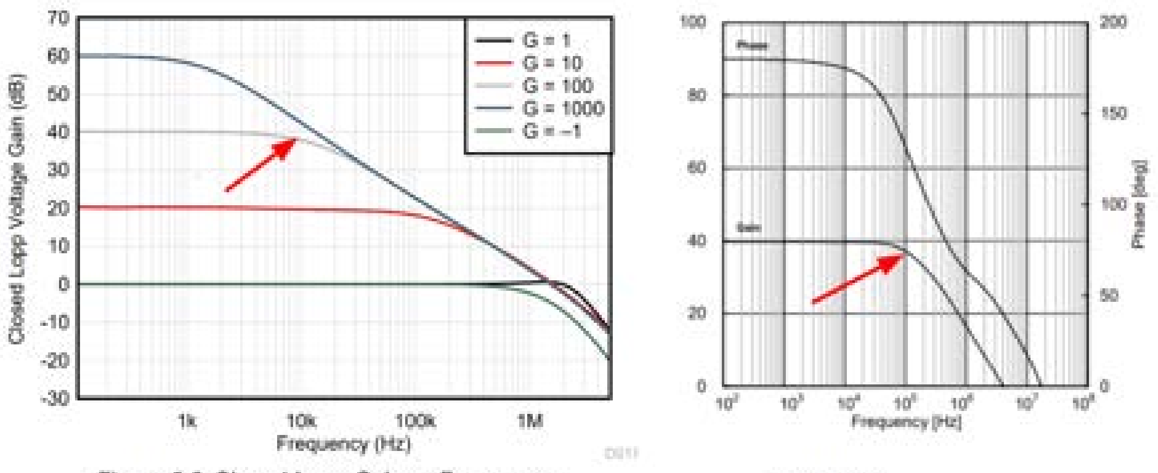
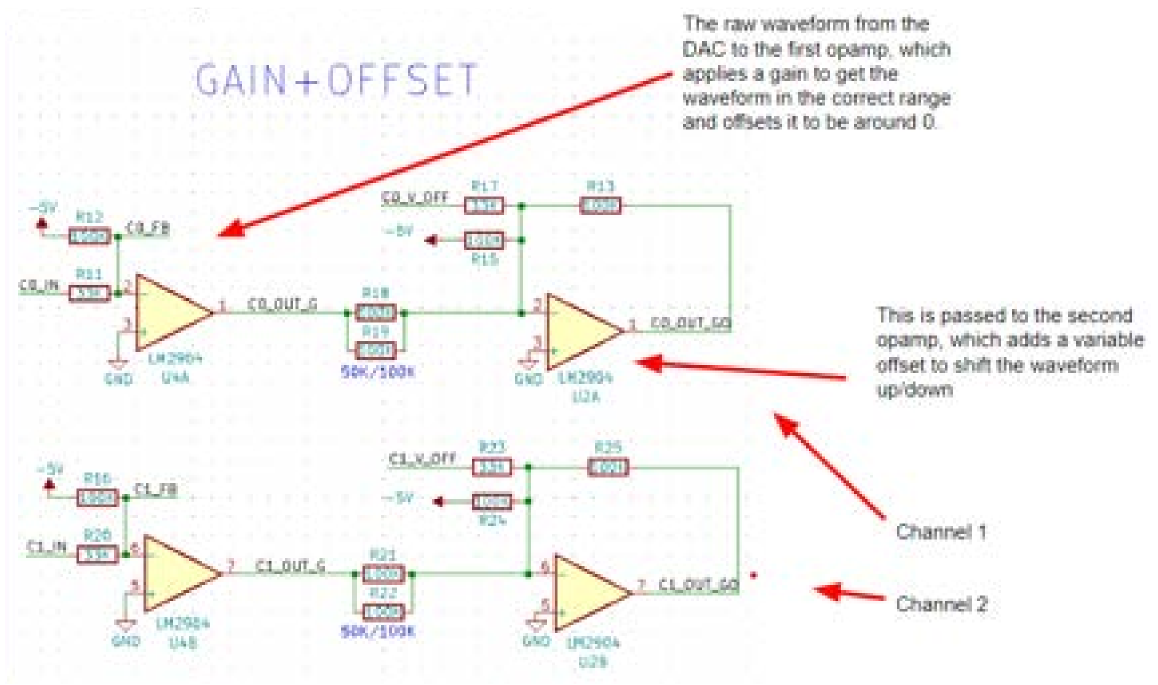


Figure 6-6. Closed-Loop Gain vs Frequency

LM2904BIDDFR
GBP=1Mhz
Cheaper option

LM4565
GBP=10Mhz
(10x bandwidth)
more expensive

6.1.3 Front End Design - Part 1- Gain and Offset

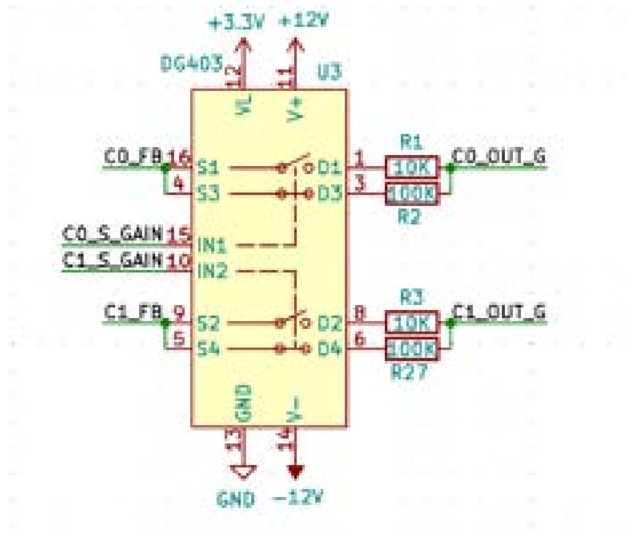


The raw waveform from the DAC to the first opamp, which applies a gain to get the waveform in the correct range and offsets it to be around 0.

This is passed to the second opamp, which adds a variable offset to shift the waveform up/down

Channel 1

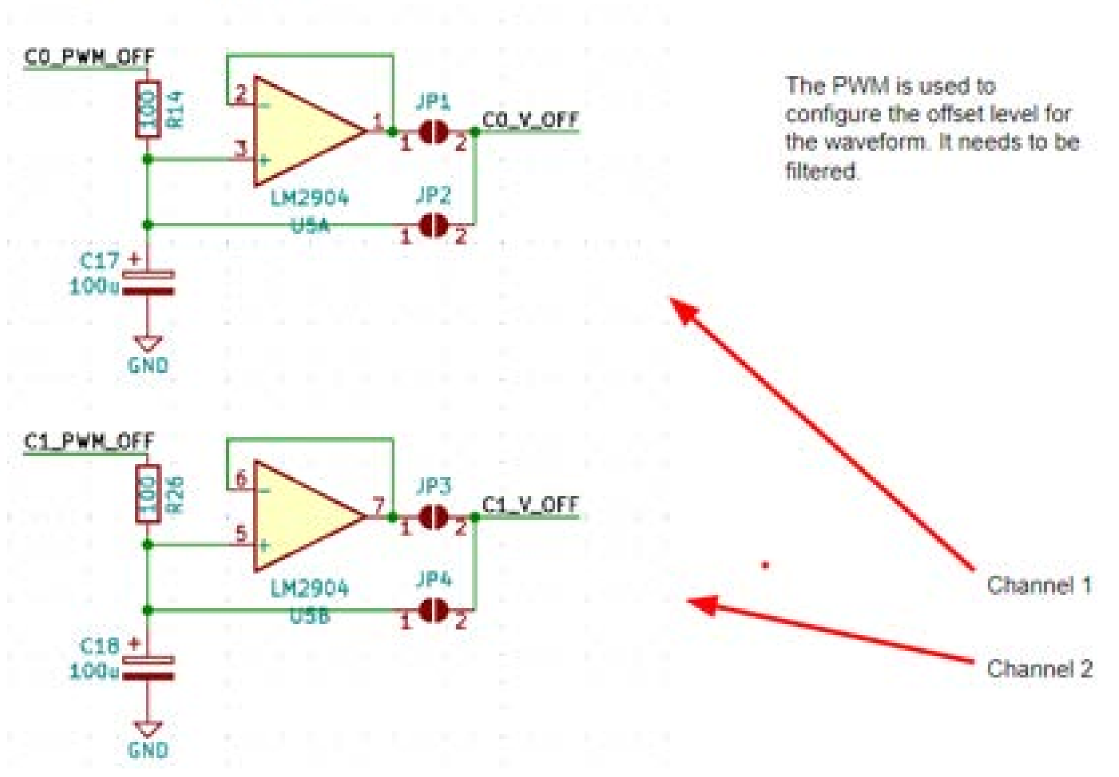
Channel 2



An analog mux is used to configure the resistor in the feedback path of the gain opamp, allowing for two gain settings.

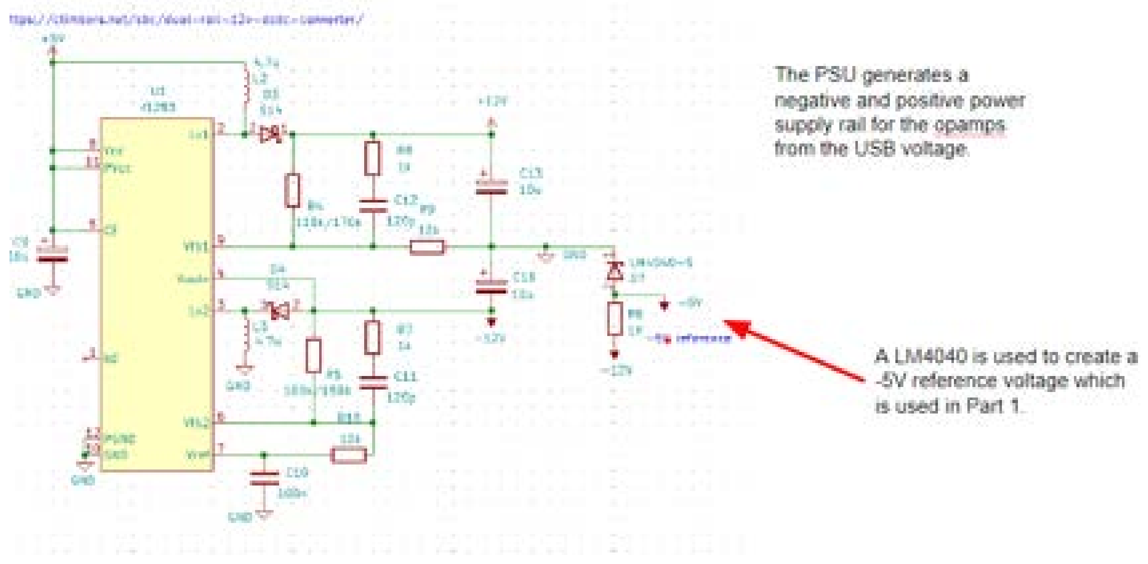
A single chip is used for both channels

6.1.4 Front End Design - Part 2 - Offset



The PWM is used to configure the offset level for the waveform. It needs to be filtered.

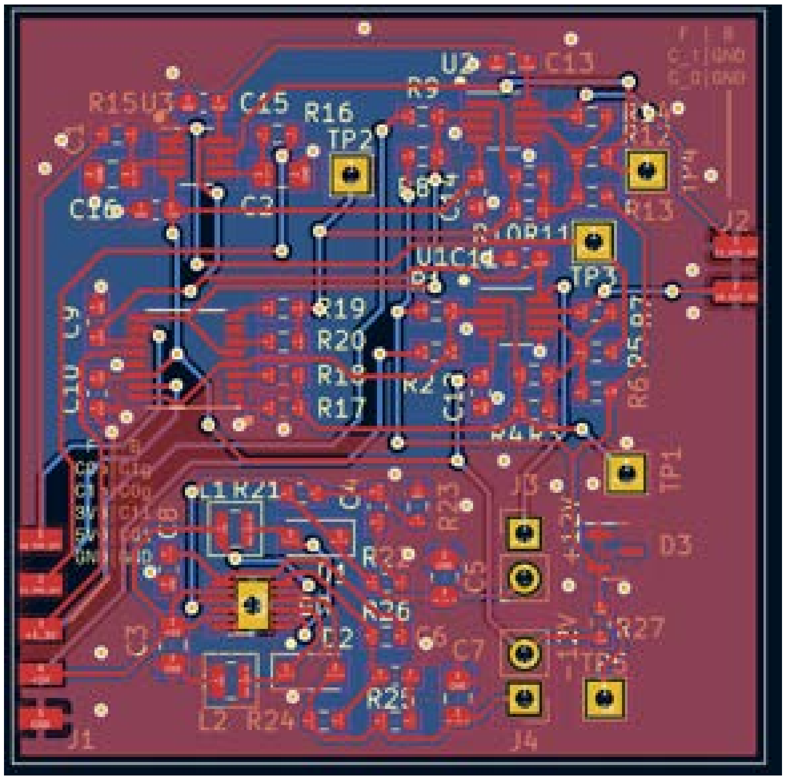
6.1.5 Front End Design - Part 3 - Power Supply Unit



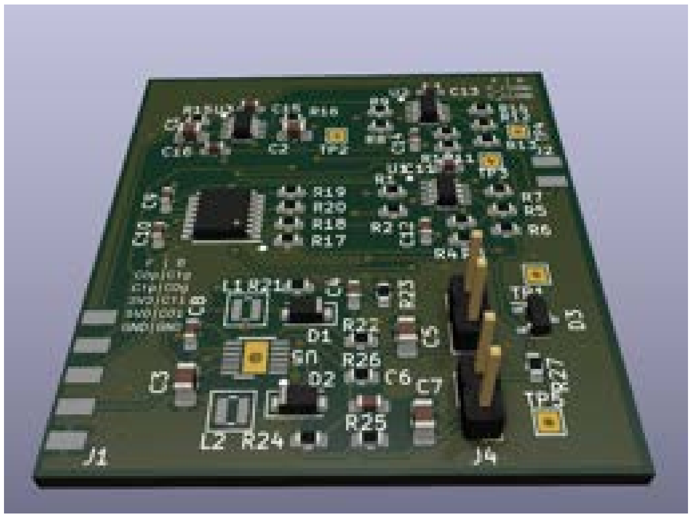
The PSU generates a negative and positive power supply rail for the opamps from the USB voltage.

A LM4040 is used to create a -5V reference voltage which is used in Part 1.

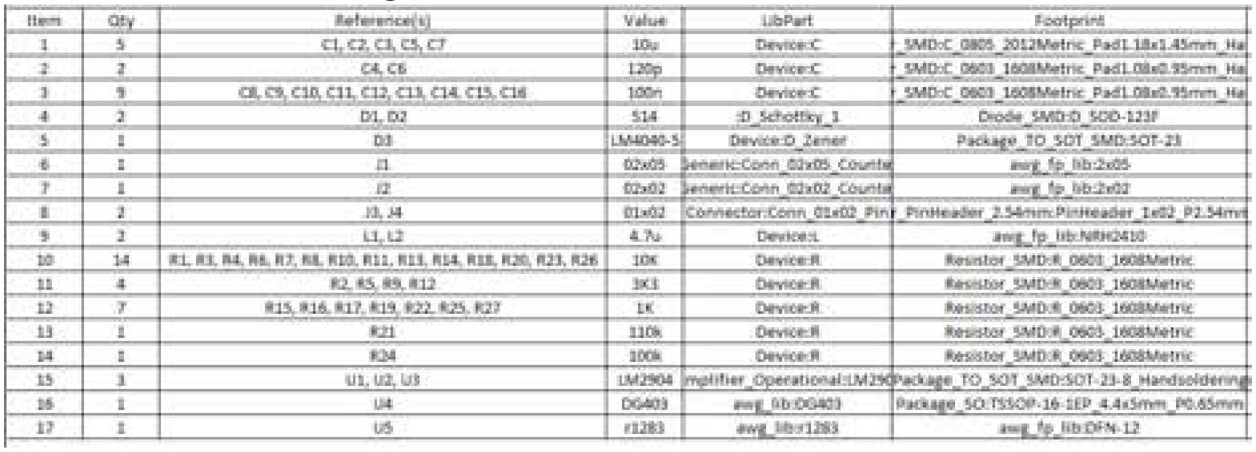
6.1.6 PCB Design Schematic



6.1.7 PCB Design 3D Rendering

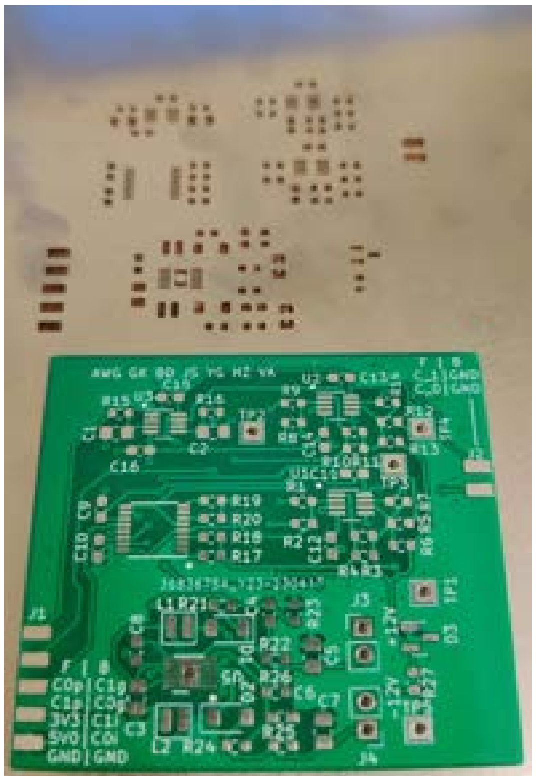


6.1.8 PCB Design - Hardware Breakdown



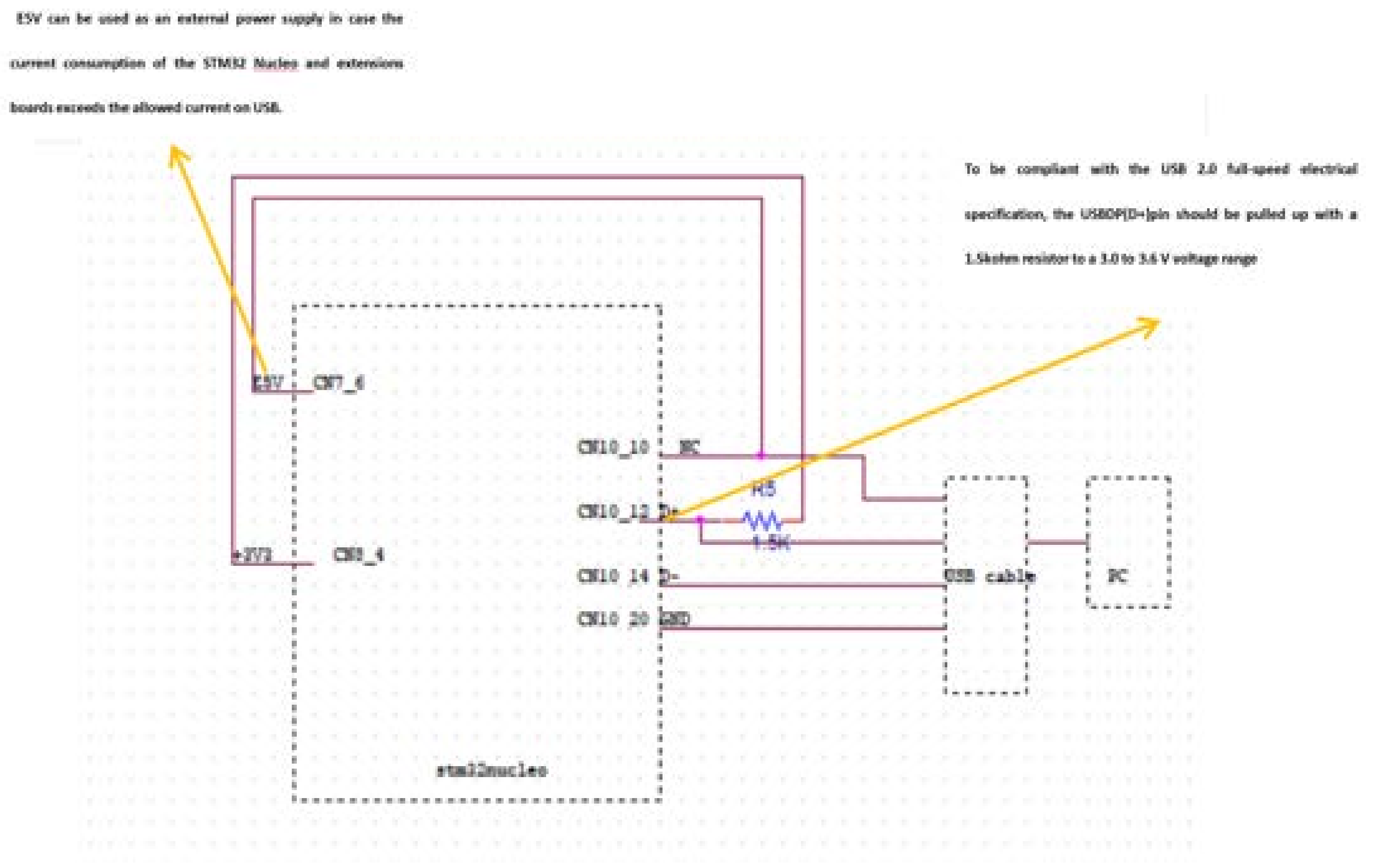
Item	Qty	Reference(s)	Value	LibPart	Footprint
1	5	C1, C2, C3, C5, C7	10u	Device:C	_SMD:C_0805_2012Metric_Pad1.18x1.45mm_Ha
2	2	C4, C6	120p	Device:C	_SMD:C_0603_1608Metric_Pad1.08x0.95mm_Ha
3	9	C8, C9, C10, C11, C12, C13, C14, C15, C16	100n	Device:C	_SMD:C_0603_1608Metric_Pad1.08x0.95mm_Ha
4	2	D1, D2	514	:D_Schottky_1	Diode_SMD:D_SOD-123F
5	1	D3	LM4040-5	Device:D_Zener	Package_TO_SOT_SMD:SOT-23
6	1	J1	02x05	Generic:Conn_02x05_Counte	awg_fp_lib:2x05
7	1	J2	02x02	Generic:Conn_02x02_Counte	awg_fp_lib:2x02
8	2	J3, J4	01x02	Connector:Conn_01x02_Pin	_PinHeader_2.54mm:PinHeader_1x02_P2.54mm
9	2	L1, L2	4.7u	Device:L	awg_fp_lib:NRH2410
10	14	R1, R3, R4, R6, R7, R8, R10, R11, R13, R14, R18, R20, R23, R26	10K	Device:R	Resistor_SMD:R_0603_1608Metric
11	4	R2, R5, R9, R12	3K3	Device:R	Resistor_SMD:R_0603_1608Metric
12	7	R15, R16, R17, R19, R22, R25, R27	1K	Device:R	Resistor_SMD:R_0603_1608Metric
13	1	R21	110k	Device:R	Resistor_SMD:R_0603_1608Metric
14	1	R24	100k	Device:R	Resistor_SMD:R_0603_1608Metric
15	3	U1, U2, U3	LM2904	mplifier_Operational:LM290	Package_TO_SOT_SMD:SOT-23-8_Handsoldering
16	1	U4	DG403	awg_lib:DG403	Package_SO:TSSOP-16-1EP_4.4x5mm_P0.65mm
17	1	U5	r1283	awg_lib:r1283	awg_fp_lib:DFN-12

6.1.9 PCB Board and Discrete Hardware Components



6.2 Microcontroller Design

6.2.1 NUCLEO-F303RE USB Hardware Design


E5V can be used as an external power supply in case the current consumption of the STM32 Nucleo and extensions boards exceeds the allowed current on USB.

To be compliant with the USB 2.0 full-speed electrical specification, the USBDP(D+)pin should be pulled up with a 1.5kohm resistor to a 3.0 to 3.6 V voltage range

7. Preliminary Experimental Plan

7.1 Preliminary Experiment and Testing Plan (ECE 492)

Testing for the prototype and final designs will be split based on the three core components of the function generator: analog front end, MCU programming, and GUI. Each of which will need to be tested individually as well as collectively once the components are integrated.

7.1.1 Analog Front End

The analog front end should be thoroughly tested for correct operation of the gain and offset functionality.

Test #1: Gain

1. Input a sine wave to the analog front end with an amplitude of 1V.
2. Go through all possible gain settings.
3. Measure the output amplitude with each setting.

   This test will ensure the gain functionality is working correctly, and allow us to measure the error.

Test #2: Offset

4. Input a DC voltage to the analog front end.
5. Sweep the offset from the max to min value.
6. Measure the output DC voltage.
7. Repeat at different Gain levels.

   This test will ensure the offset is correctly added to the input signal, and the output range goes from the max to min value while the offset goes from the max to min value. This test will also allow us to measure the error.

Test #3: Bandwidth

This test will ensure the front end has enough bandwidth to pass signals at our max output frequency without losing significant amplitude.

1. Perform a frequency sweep on the input of the analog front end.
2. Measure the output level
3. Repeat at different Gain levels or with different offsets.

7.1.2 MCU

The Nucleo-F303RE board is a popular development board used for prototyping and testing embedded systems. When designing a waveform generator, it is important to thoroughly test the hardware and software before moving to production. Testing with the Nucleo-F091RC board is an essential step in this process as it allows the team to verify that the waveform generator is functioning as intended and to identify and address any potential issues. This IDE is new to all of the members, so preliminary testing is required.

Test #1:
The first test will be used as preliminary testing for the STM32 Nucleo board. Verify the jumper positions, install the USB driver, and connect the board to a PC to ensure that the LED lights are functioning correctly when the button is pressed. Additionally, the team will use the available demonstration and software examples to test the software functionality and compatibility. Any issues with jumper positions, LED lights, or software functionality could be identified and corrected before moving to production.

Test #2:

1. Hardware setup: Connect an LED and a current-limiting resistor to the GPIO output pin of the DAC. Connect the ground of the LED to the ground pin of the DAC.
2. Code setup: Write a simple code that sets the GPIO output pin of the DAC to a known logic level (e.g., high or low), and continuously updates the output value.
3. Verification: Run the code on the microcontroller and observe the LED. The LED should turn on or off, depending on the logic level set in the code.
4. Repeat: Repeat the test with different output values to verify that the GPIO output of the DAC can be set accurately and consistently.

This test can help ensure that the GPIO output of the DAC is functioning correctly and can be set as expected, which is important for a range of applications, such as controlling external devices or triggering other circuit components. Any issues with the GPIO output of the DAC could be identified and corrected before moving to production.

Test #3:

1. Hardware setup: Connect the Nucleo board to a PC via USB using a Type-A to Mini-B cable. Connect via ST-LINK to the Nucleo board using the SWD (Serial Wire Debug) interface.

2. Software setup: In STM32CubeIDE, create a project that sets up a simple data transfer routine, such as sending a string of characters from the microcontroller to the PC over the ST-Link interface. In Python, use the Pyserial library to set up a serial communication object and read the data being sent by the microcontroller over the ST-Link interface.
3. Verification: Build and upload the project to the Nucleo board using STM32CubeIDE. Verify that the data is being transmitted and received correctly, and that there are no errors or data loss during the transfer.
4. Repeat: Repeat the test with different data transfer rates and different data formats to verify that the ST-Link communication is reliable and consistent.

This test can help ensure that the ST-Link communication between the PC and Nucleo board is functioning correctly and can transfer data reliably, which is important for many applications. Any issues with data transfer or communication can be identified and corrected before moving to production.

7.1.3 GUI

The GUI is how the user interacts with the system specifically when it comes to selecting types of waveforms and setting values for criteria such as frequency, offset, and amplitude. Thus, it is important to confirm that the GUI is both displaying desired changes as well as transmitting the proper information to the MCU.

Test #1: The first test is meant to verify that the GUI is properly plotting functions based on user input to ensure that the user has accurate visuals at all times.

1. Design several examples of settings for each of the standard waveform shapes.
2. Input values into both the GUI and the waveform generator on the AD2
3. Verify that the waveforms match relative to the respective axis.

Test #2:

1. Design several waveform examples for testing or use the waveforms from test 1.
2. For each waveform record and display the data that is to be transmitted to the Nucleo board.

3. Verify that the data received matches the sent data for each of the example waveforms.

Test #3:

1. Input data parameters that contain non-numerical values.
2. Verify that the input data does not allow a waveform to be generated.
3. Allow input parameters with a prefix signifying magnitude to be accepted, converted and evaluated if the values are allowed to be maintained by the hardware.

7.2 Testing Plan for Prototype With Integration(ECE 492/493)

The purpose of this testing is to verify that individual components are functionally operational when integrated together.

7.2.1 Analog Front End

Test:

1. Connect the Nucleo-F303RE board to the PC using a USB cable.
2. Using either Waveforms Live or PyQtGraph, generate a digital signal with a specified amplitude and frequency.
3. Adjust the gain and offset settings on the analog front end of the board using the GUI.
4. Measure the output signal using an oscilloscope.
5. Record the output signal amplitude and frequency for each gain and offset setting.
6. Analyze the results to determine the gain and offset error and ensure that the gain and offset functionality is working correctly.
7. Repeat the test with different digital signal amplitudes and frequencies to ensure the accuracy and reliability of the analog front end.

Overall, this test would allow you to use a GUI to adjust the gain and offset settings on the Nucleo-F303RE board and test the functionality of the analog front end with a digital signal generated by the PC. This would ensure that the board is functioning correctly and provide accurate and reliable measurements for your project.

7.2.2 MCU/GUI

Test:

1. Use Pyserial to send the digital signal from the PC's GUI to the Nucleo-F303RE board.
2. Adjust the gain and offset settings on the analog front end of the board using the MCU firmware.
3. Measure the output signal using an oscilloscope or other measurement equipment.
4. Record the output signal amplitude and frequency for each gain and offset setting.
5. Analyze the results to determine the gain and offset error and ensure that the gain and offset functionality is working correctly.
6. Repeat the test with different digital signal amplitudes and frequencies to ensure the accuracy and reliability of the analog front end.

The purpose of this test is to verify Pyserial’s ability to accurately send/receive data between the user controlled GUI interface and MCU’s IDE. Subsequent testing will include removing STM32CubeIDE’s involvement with the User to create a simpler User experience.

Using DMA to transfer data between memory and I/O devices, the CPU can perform other tasks while the data transfer is taking place. With the DAC, DMA can be used to offload the task of transferring data from the MCU to the DAC, reducing the MCU's resource load and improving system performance.The main testing criteria for the MCU is to ensure system resources are kept low while also producing accurate results.

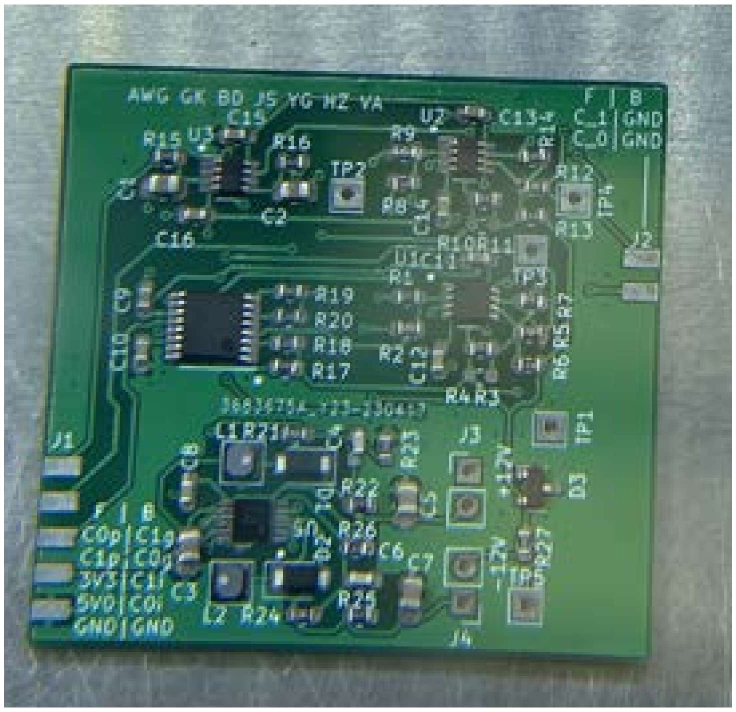
7.2.3 PCB

Designing a custom PCB can be challenging and requires careful consideration of component placement, trace routing, trace widths, and trace lengths to prevent common issues during testing. The trace width affects the resistance and current carrying capacity of the circuit, while the trace length affects the propagation delay of the signal and can introduce issues such as signal skew or reflection. During testing, it's important to inspect the board for physical defects, verify the power supply, and check the resistance between different points on the PCB. Taking the time to optimize the circuit and ensure proper component placement, trace routing, and appropriate trace widths and lengths can help prevent issues during testing and ensure the successful operation of the board.

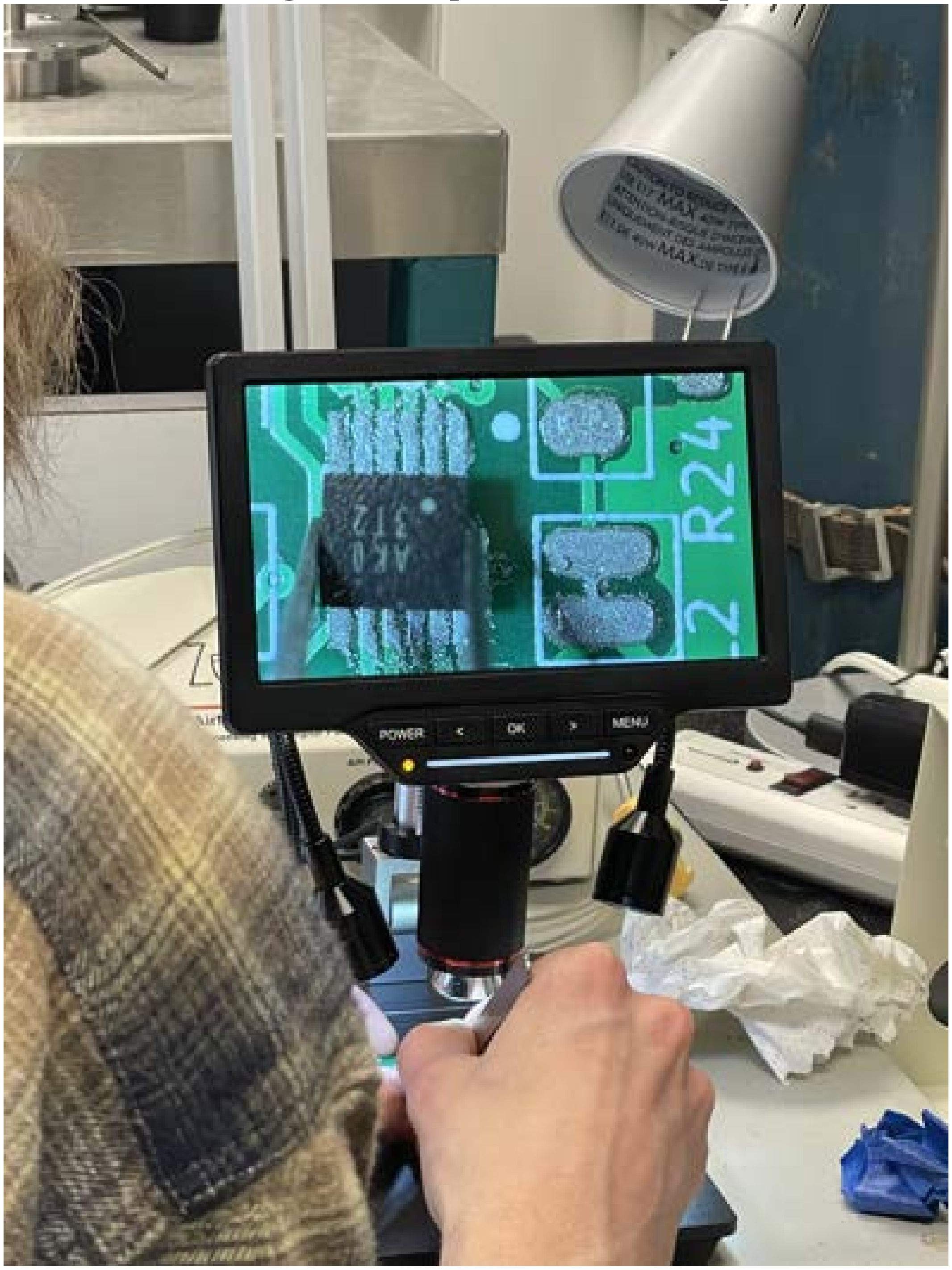
8. Prototyping Progress Report

8.1 Analog Hardware Prototyping

8.1.1 Soldering PCB Components



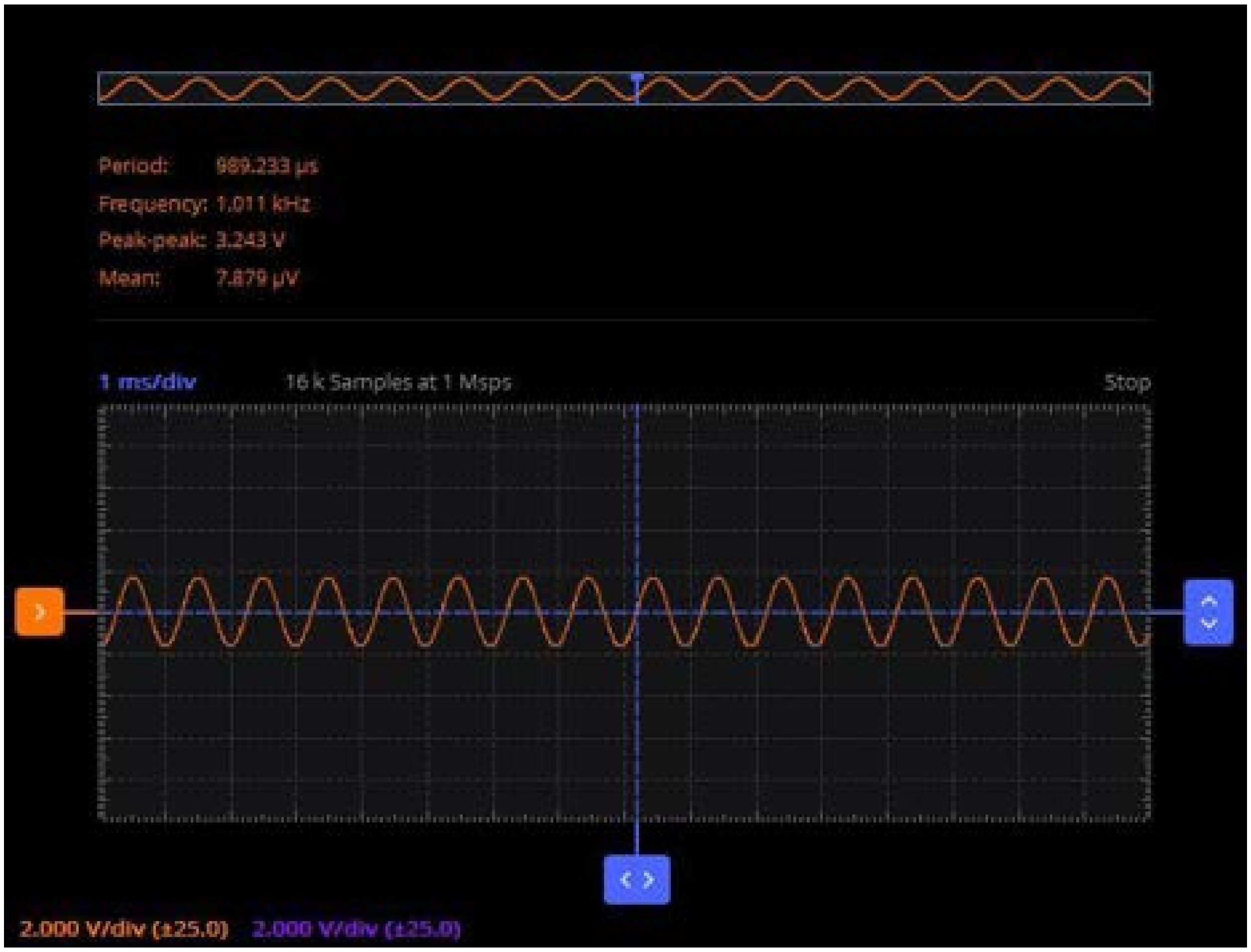
8.1.2 Soldering PCB Components - Microscope Placement



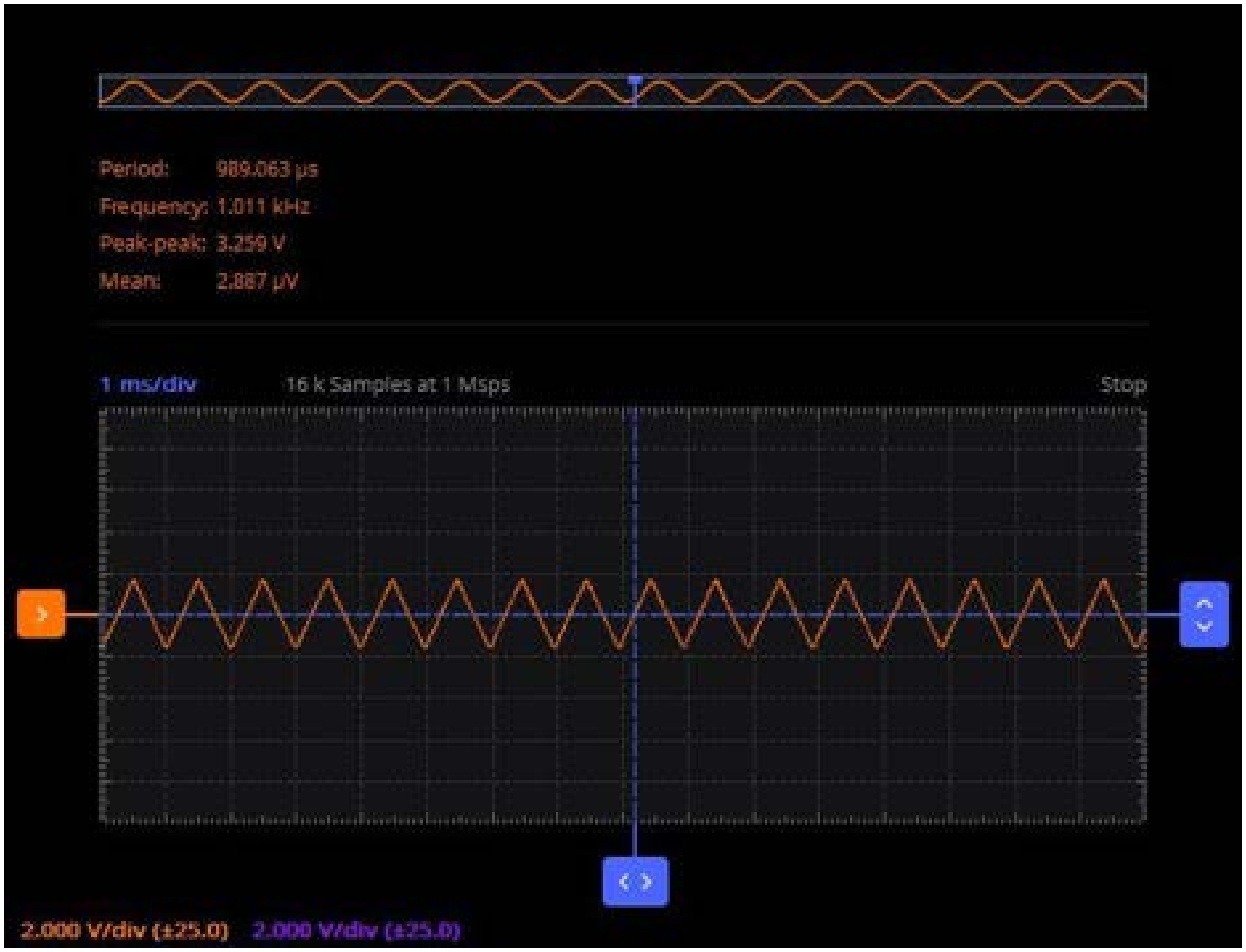
8.2 Microcontroller Prototyping

8.2.1 MCU Waveform Generation using Timer/DMA/DAC

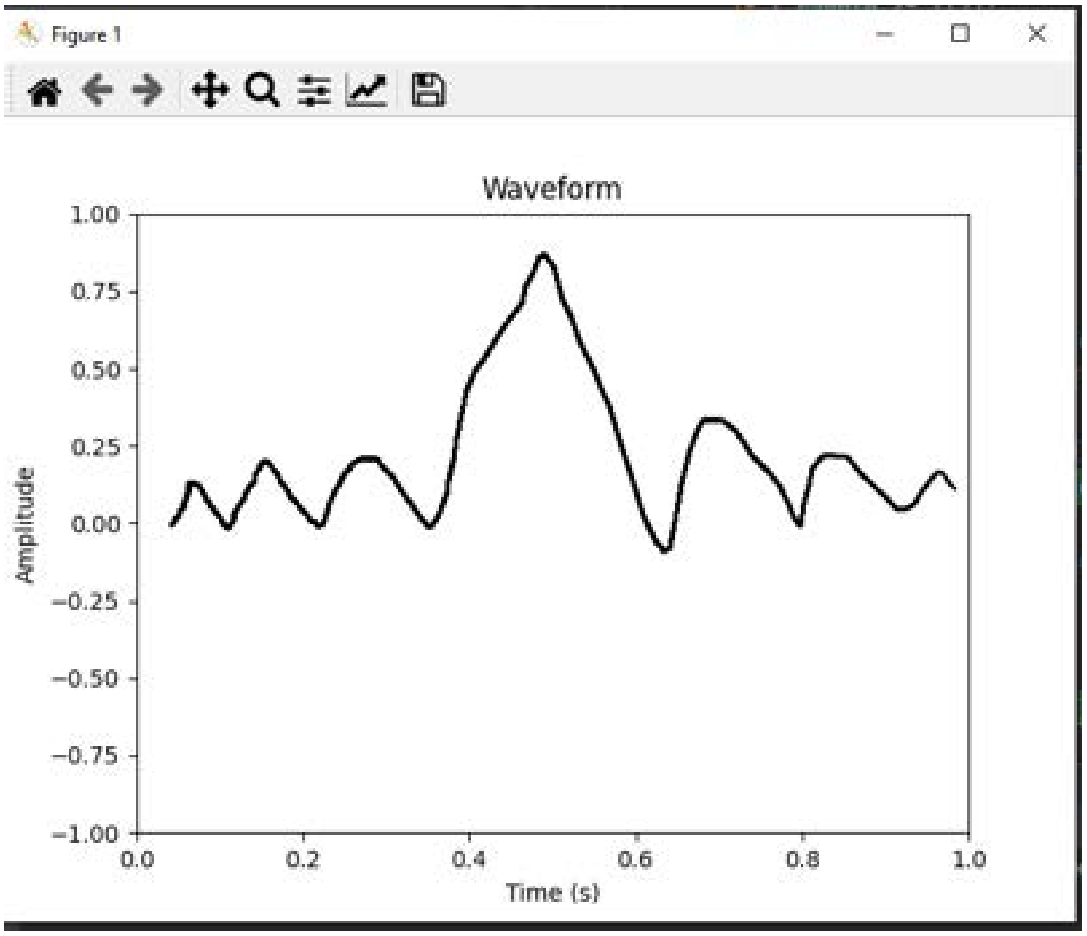
Sine Wave Oscilloscope Measurement:


Period: 989.233 µs
Frequency: 1.011 kHz
Peak-peak: 3.243 V
Mean: 7.879 µV

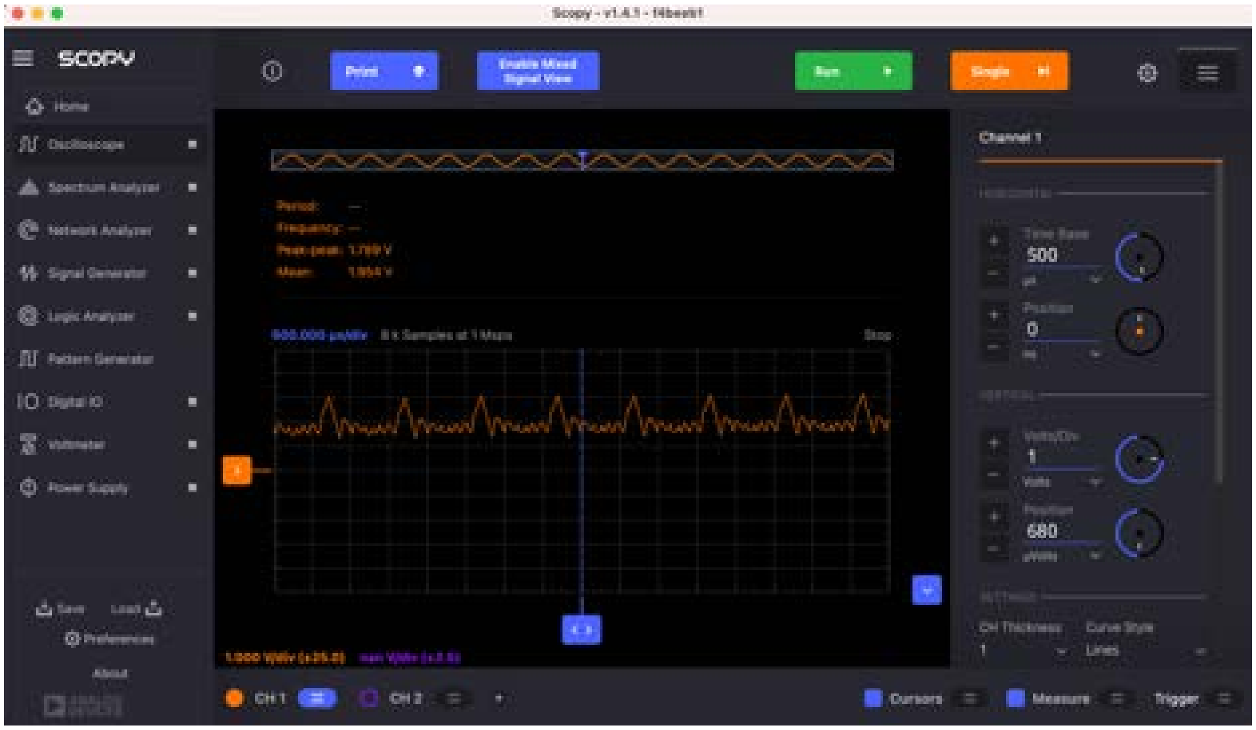
Triangle Wave Oscilloscope Measurement:


Period: 989.063 µs
Frequency: 1.011 kHz
Peak-peak: 3.259 V
Mean: 2.887 µV

User Defined Arbitrary Waveform:



User Defined Arbitrary Wave Oscilloscope Measurement:



Period: —
Frequency: —
Peak-peak: 1.769 V
Mean: 1.954 V

500.000 µs/div 8 k Samples at 1 Msps Stop

1.000 V/div (±25.0) nan V/div (±7.5)

Channel 1

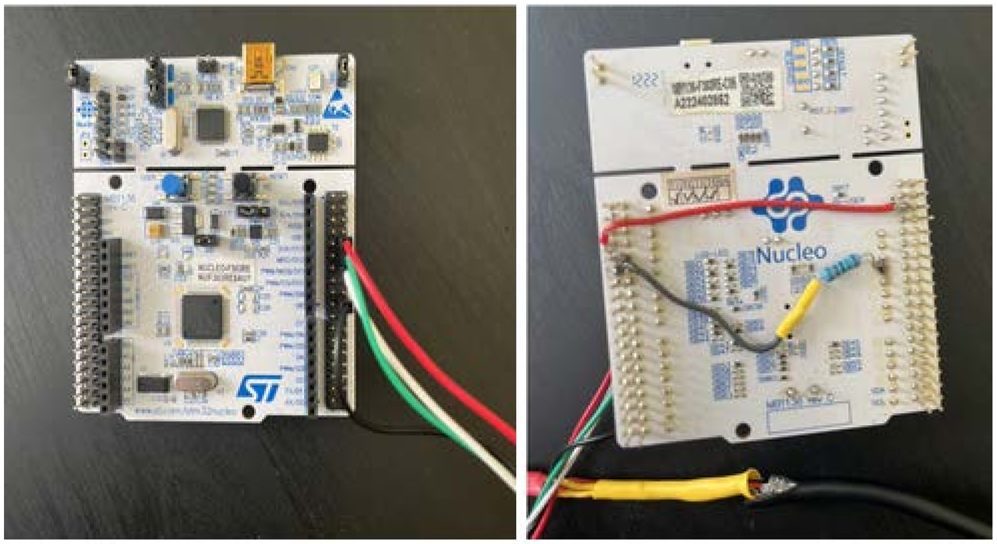
HORIZONTAL
Time Base 500 µs
Position 0 ns

VERTICAL
Volts/Div 1 Volts
Position 680 µVolts

SETTINGS
CH Thickness 1
Curve Style Lines

8.2.2 MCU Hardware Setup

MCU USB Hardware Configuration:

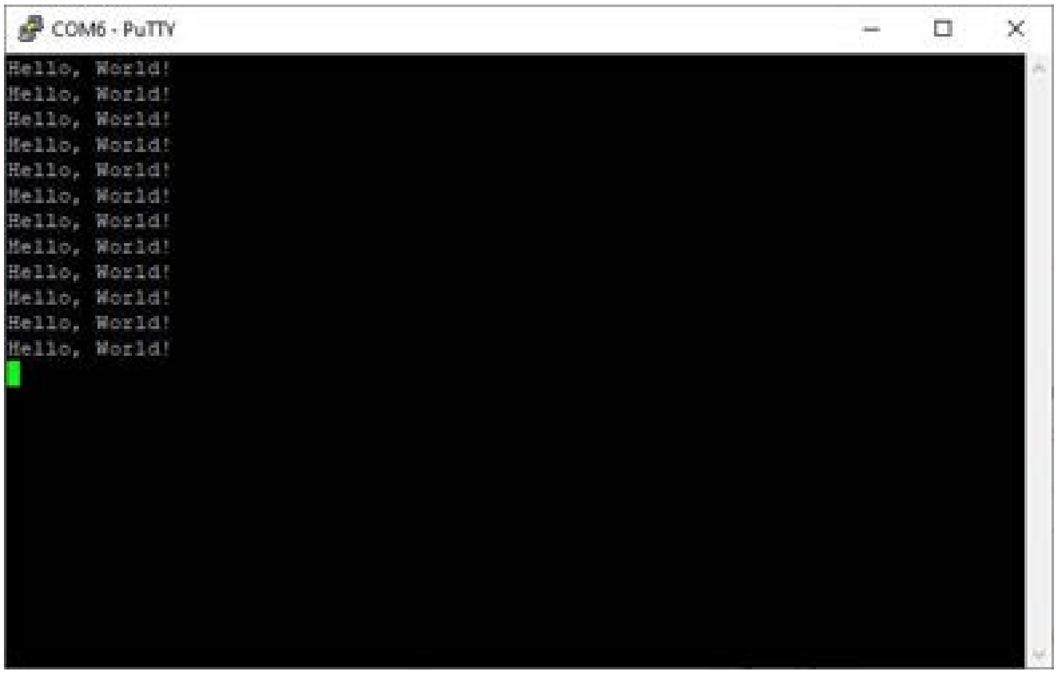


- **HSE oscillator on-board from X3 crystal (not provided):** for typical frequencies and its capacitors and resistors, refer to the STM32 microcontroller datasheet. Refer to the AN2867 Application note for oscillator design guide for STM32 microcontrollers.The X3 crystal has the following characteristics: 8 MHz, 16 pF, 20 ppm, and DIP footprint. It is recommended to use 9SL8000016AFXHF0 manufactured by Hong Kong X'tals Limited.

  The following configuration is needed:
  - SB54 and SB55 OFF
  - R35 and R37 soldered
  - C33 and C34 soldered with 20 pF capacitors
  - SB16 and SB50 OFF

8.2.3 MCU USB Communication

Data Transmission from MCU to PC over serial:

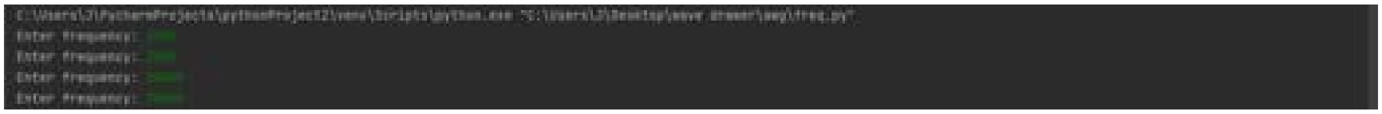


COM6 - PuTTY

```
Hello, World!
Hello, World!
Hello, World!
Hello, World!
Hello, World!
Hello, World!
Hello, World!
Hello, World!
Hello, World!
Hello, World!
Hello, World!
Hello, World!
```

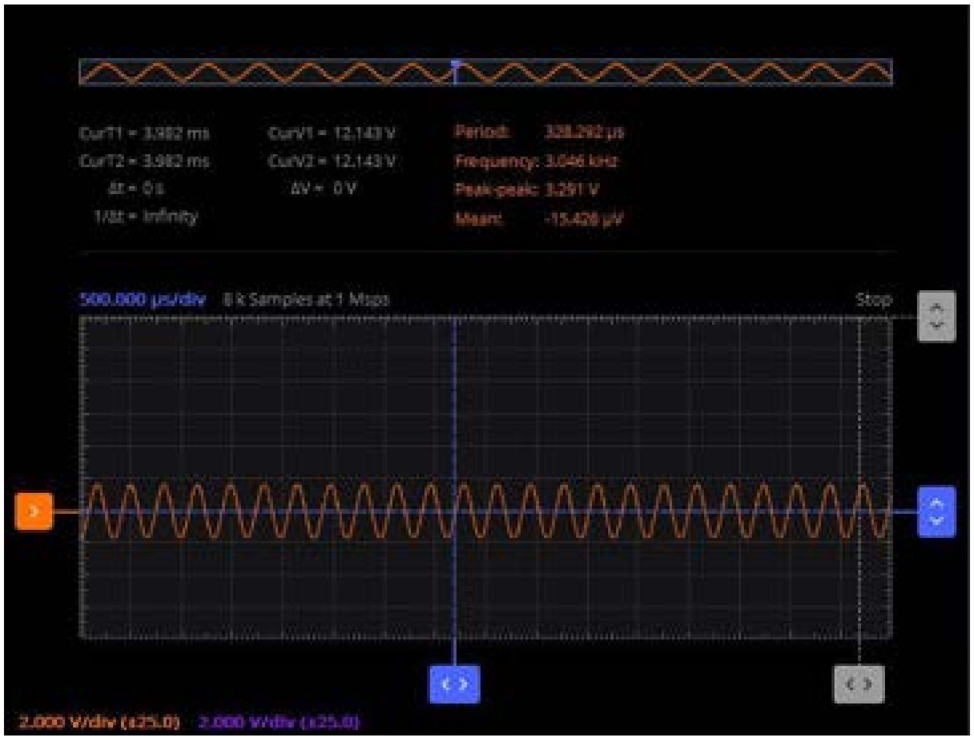
8.2.4 Data Transmission from PC to MCU over serial

Frequency Values Sent to MCU:



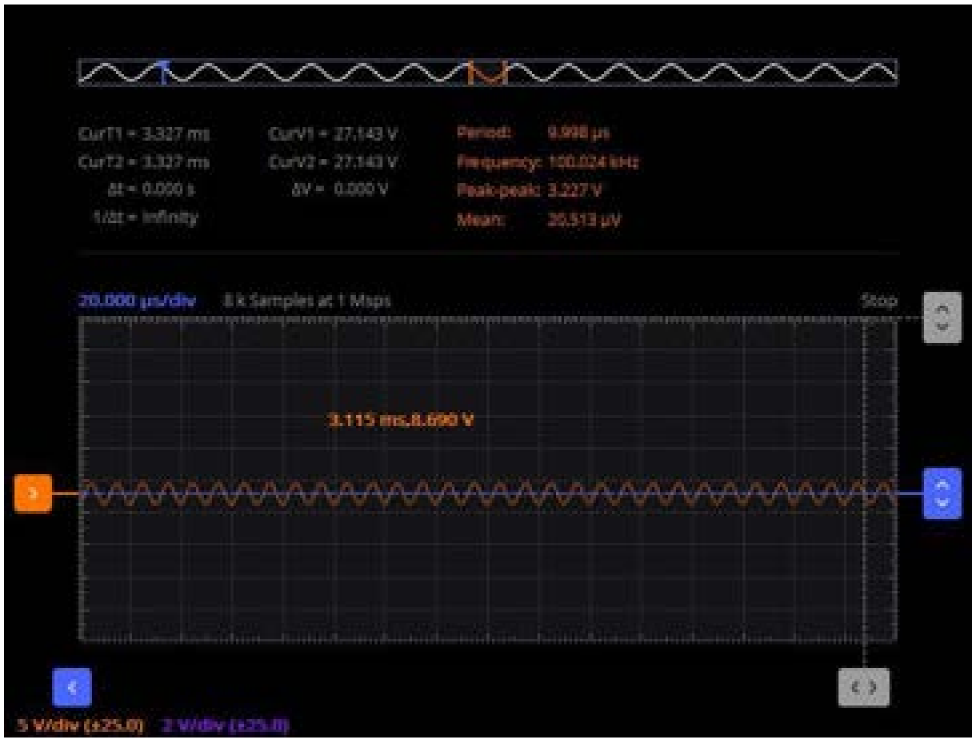
```
C:\Users\J\PycharmProjects\pythonProject2\venv\Scripts\python.exe "C:\Users\J\Desktop\wave drawer\awg\freq.py"
Enter frequency: [illegible]
Enter frequency: [illegible]
Enter frequency: [illegible]
Enter frequency: [illegible]
```

Oscilloscope Measurement: 3 kHz Test:



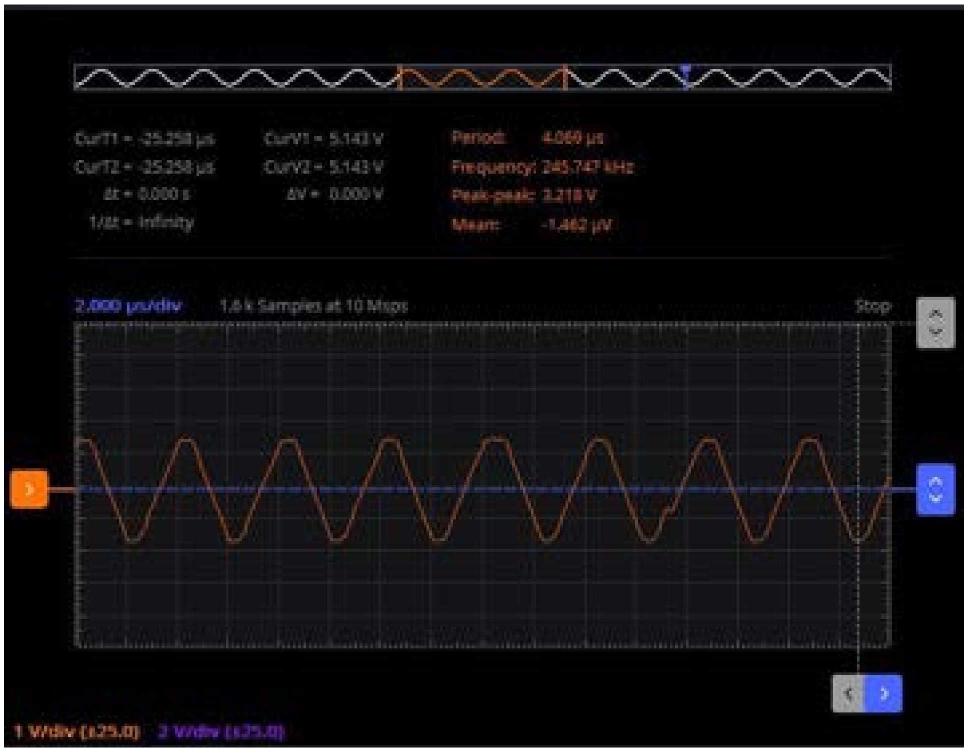
```
CurT1 = 3.982 ms      CurV1 = 12.143 V      Period:     328.292 µs
CurT2 = 3.982 ms      CurV2 = 12.143 V      Frequency:  3.046 kHz
   Δt = 0 s              ΔV =  0 V          Peak-peak:  3.291 V
 1/Δt = Infinity                            Mean:       -15.428 µV

500.000 µs/div   8 k Samples at 1 Msps                               Stop

2.000 V/div (±25.0)   2.000 V/div (±25.0)
```

Oscilloscope Measurement: 100 kHz Test:



```
CurT1 = 3.327 ms      CurV1 = 27.143 V      Period:     9.998 µs
CurT2 = 3.327 ms      CurV2 = 27.143 V      Frequency:  100.024 kHz
   Δt = 0.000 s          ΔV =  0.000 V      Peak-peak:  3.227 V
 1/Δt = Infinity                            Mean:       20.513 µV

20.000 µs/div   8 k Samples at 1 Msps                                Stop

3.115 ms,8.690 V

5 V/div (±25.0)   2 V/div (±25.0)
```

Oscilloscope Measurement: 250 kHz Test:

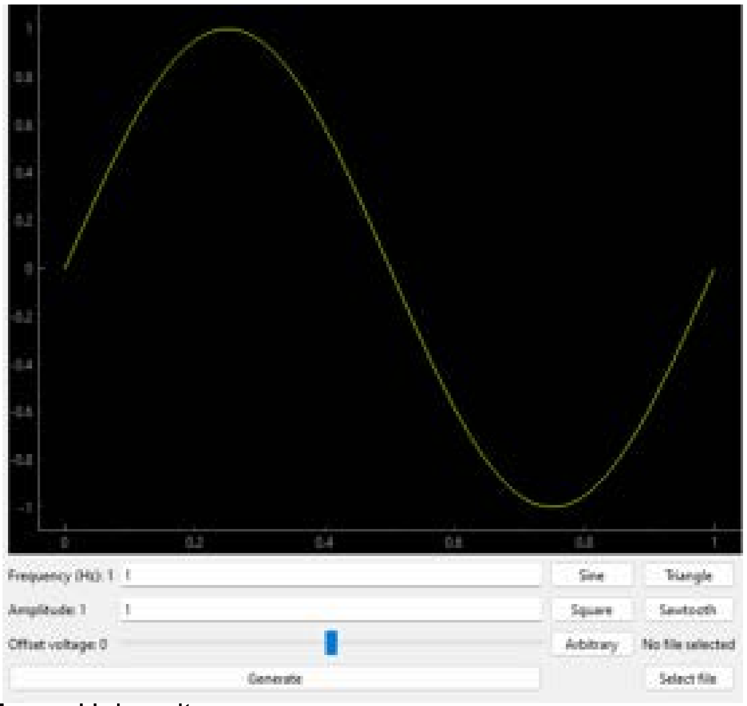


```
CurT1 = -25.258 µs    CurV1 = 5.143 V    Period:     4.069 µs
CurT2 = -25.258 µs    CurV2 = 5.143 V    Frequency:  245.747 kHz
   Δt = 0.000 s          ΔV = 0.000 V    Peak-peak:  3.218 V
 1/Δt = Infinity                         Mean:       -1.462 µV

2.000 µs/div    1.6 k Samples at 10 Msps    Stop

1 V/div (±25.0)    2 V/div (±25.0)
```

8.3 Graphical User Interface Prototyping

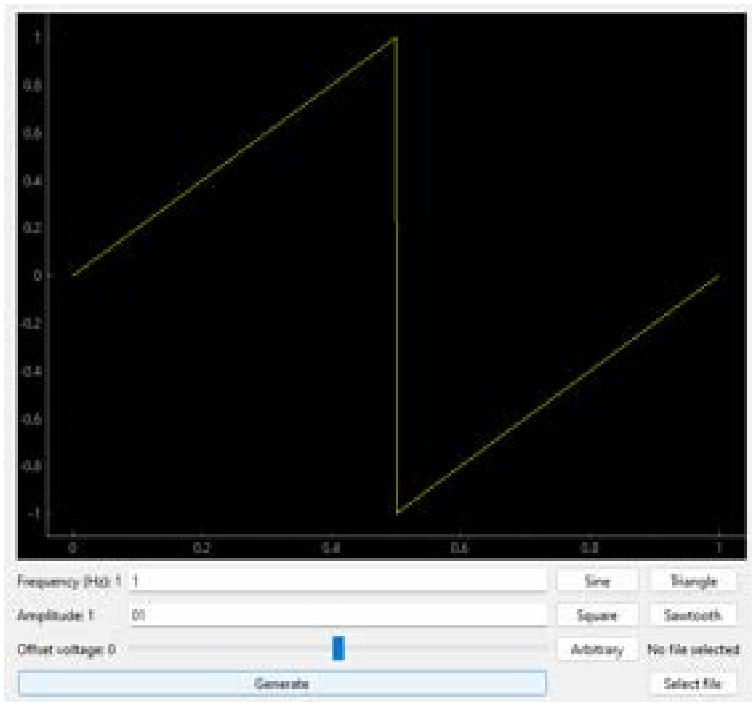
8.3.1 Default Waveform shapes:

Sine wave:



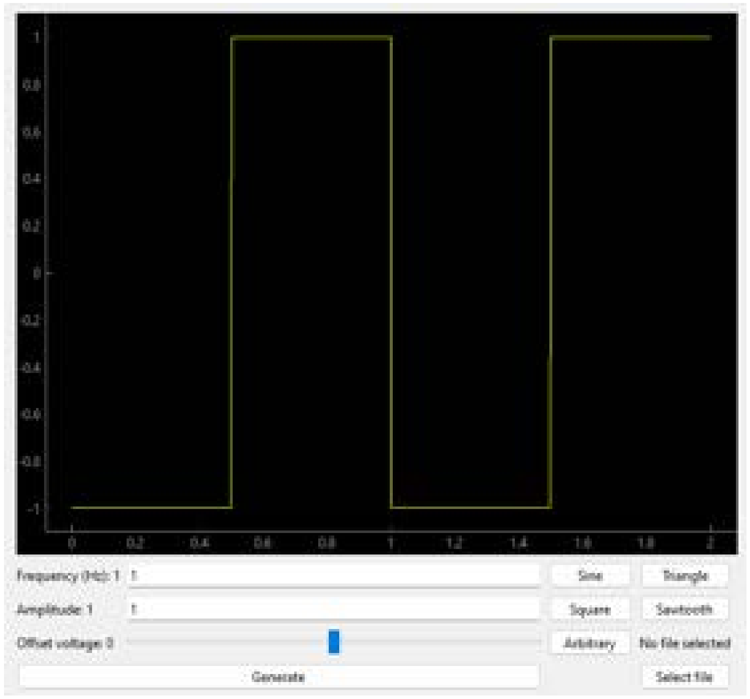
Frequency (Hz): 1 | 1
Amplitude: 1 | 1
Offset voltage: 0
No file selected

Sawtooth wave:



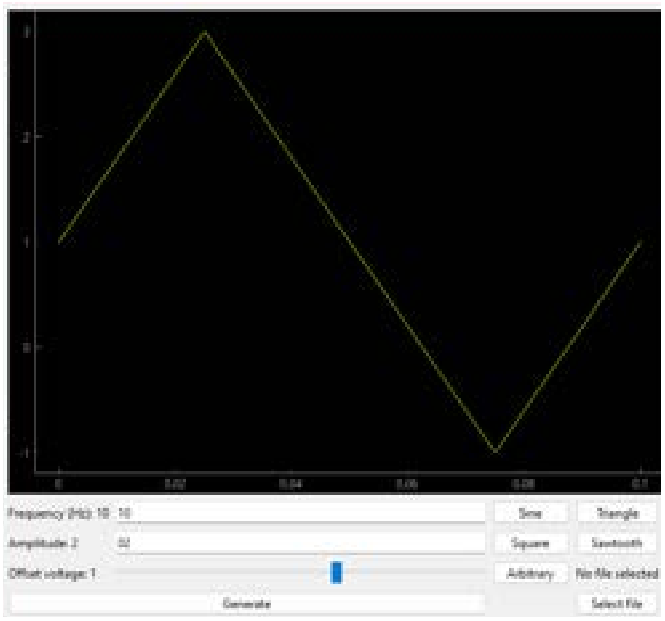
Frequency (Hz): 1 1
Amplitude: 1 01
Offset voltage: 0

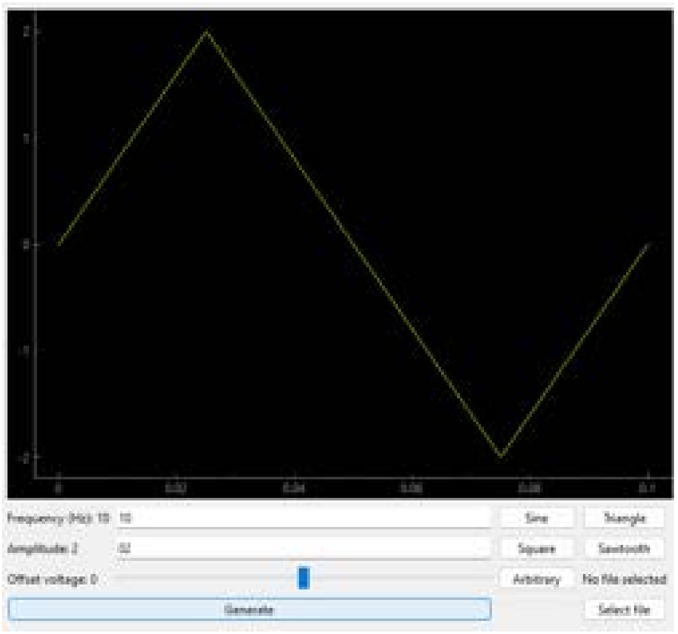
Square wave:



Frequency (Hz): 1 1
Amplitude: 1 1
Offset voltage: 0

8.3.2 Signal modifications:

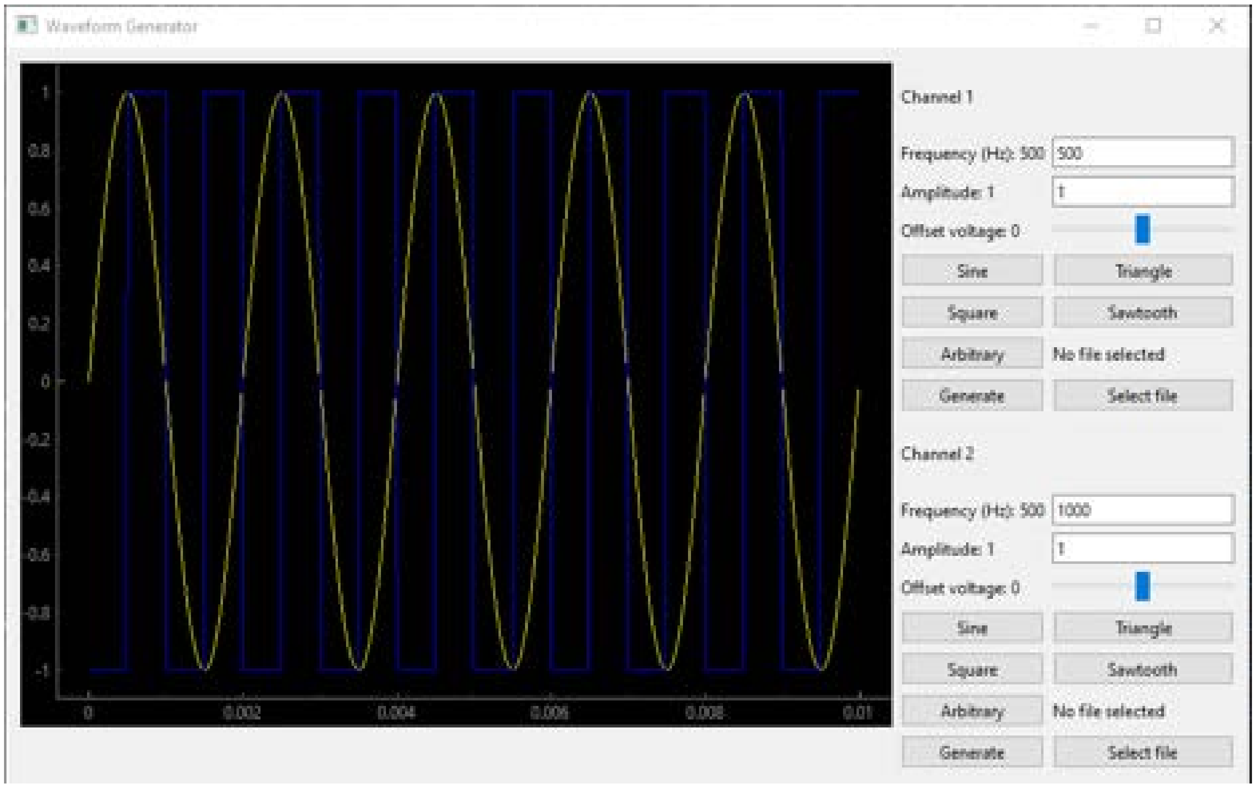

Frequency (Hz): 10 10
Amplitude: 2 02
Offset voltage: 1


Frequency (Hz): 10 10
Amplitude: 2 02
Offset voltage: 0

Triangle Wave (Amplitude: 2, Offset: 0)

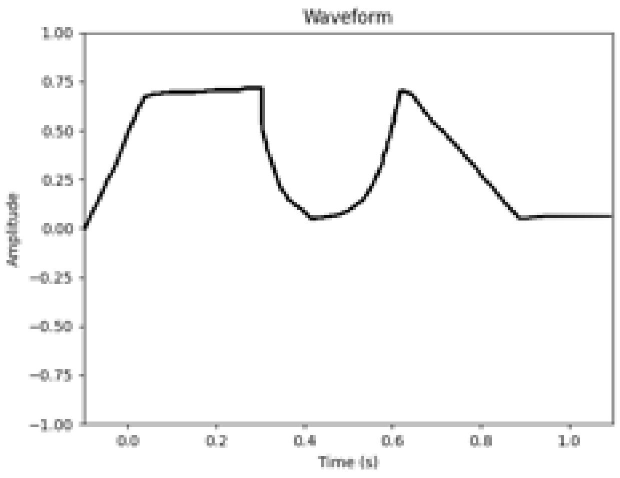
Triangle Wave (Amplitude 2, Offset: 1)

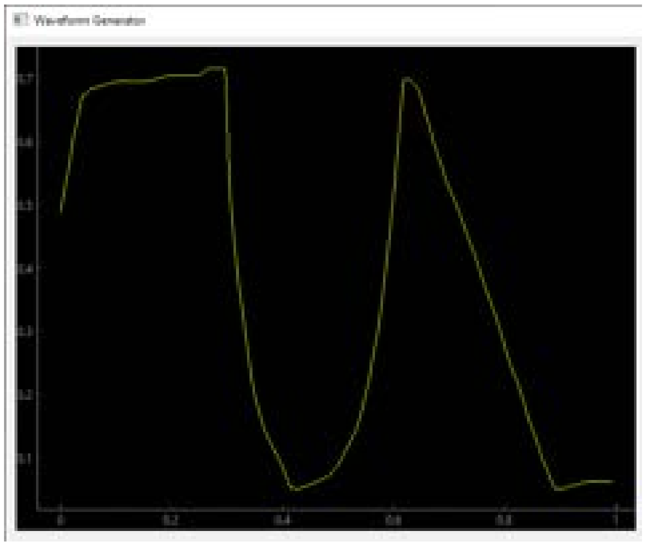
8.3.3 Current GUI Layout:


Channel 1
Frequency (Hz): 500 500
Amplitude: 1 1
Offset voltage: 0
No file selected

Channel 2
Frequency (Hz): 500 1000
Amplitude: 1 1
Offset voltage: 0
No file selected

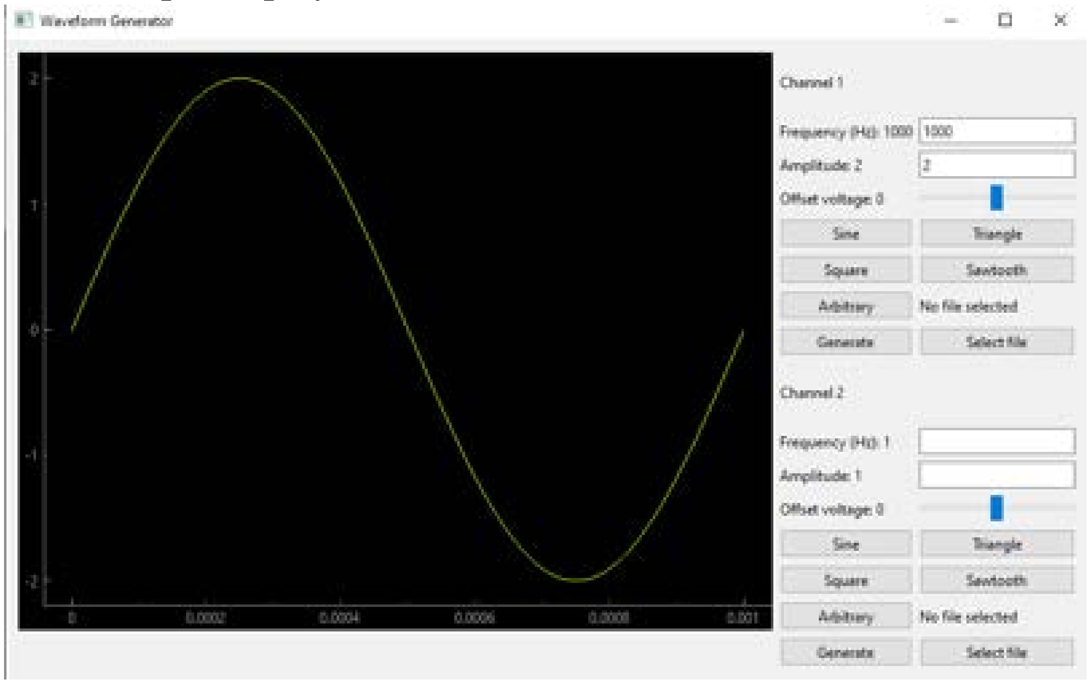
8.3.4 Arbitrary Waveform Display:





User created arbitrary waveform: GUI generated Waveform

8.3.5: Graph Display Validation



Channel 1

Frequency (Hz): 1000 1000

Amplitude: 2 2

Offset voltage: 0

No file selected

Channel 2

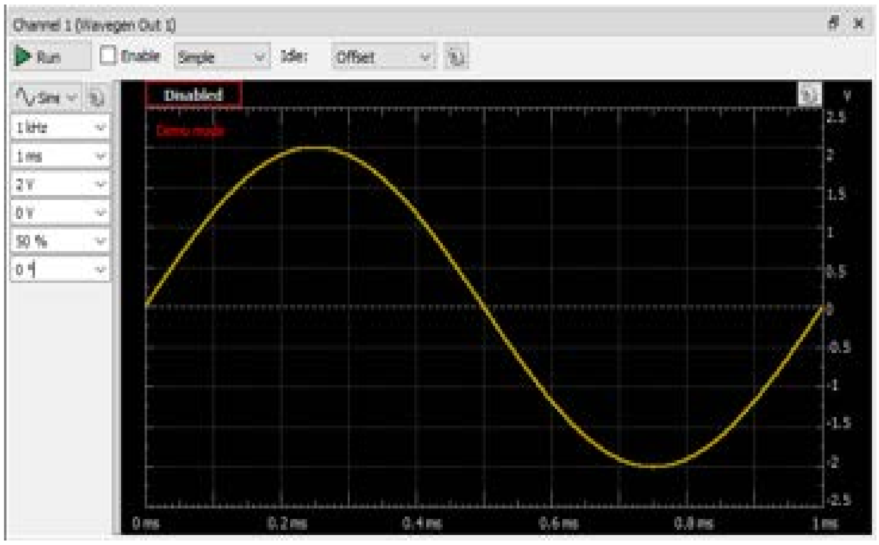
Frequency (Hz): 1

Amplitude: 1

Offset voltage: 0

No file selected

Prototype GUI: 1KHz, 2V Amplitude, 0V Offset



AD2 Display: 1KHz, 2V Amplitude, 0V Offset

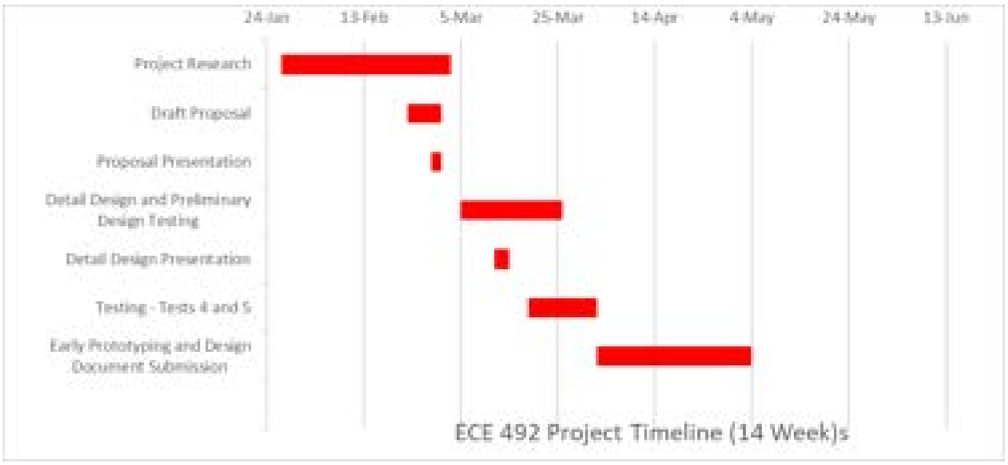
9. Schedule and Milestones

9.1 Allocation of Responsibilities

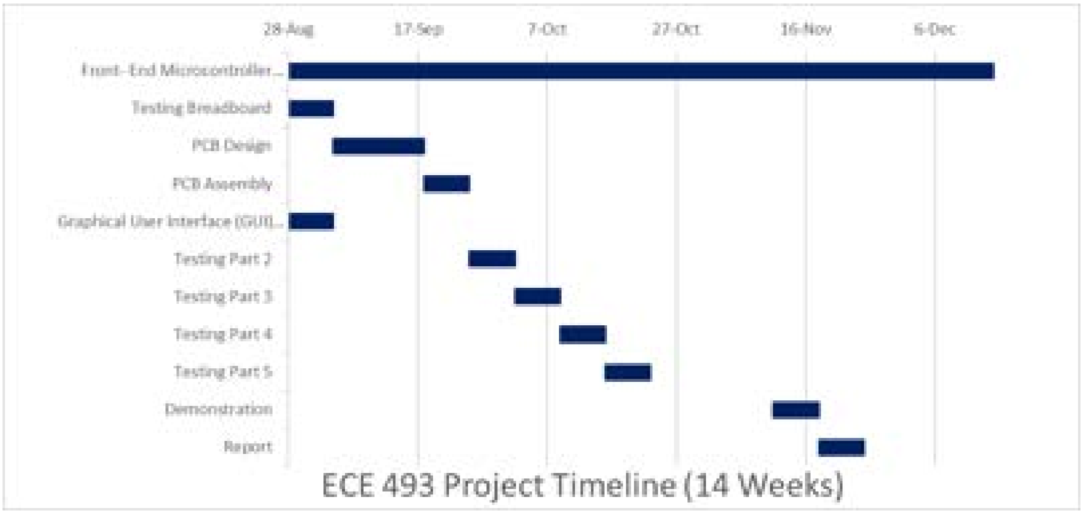
The individual tasks are divided based upon background, skills, and general interest in the project area.

- Yifei Gao / James Schaeffer : Microcontroller Unit (MCU) Programming
- Vikram Arunachalam / Hussain Zainal : Graphical User Interface (GUI) Programming
- German Kuznetsov / William Denham : Hardware Design

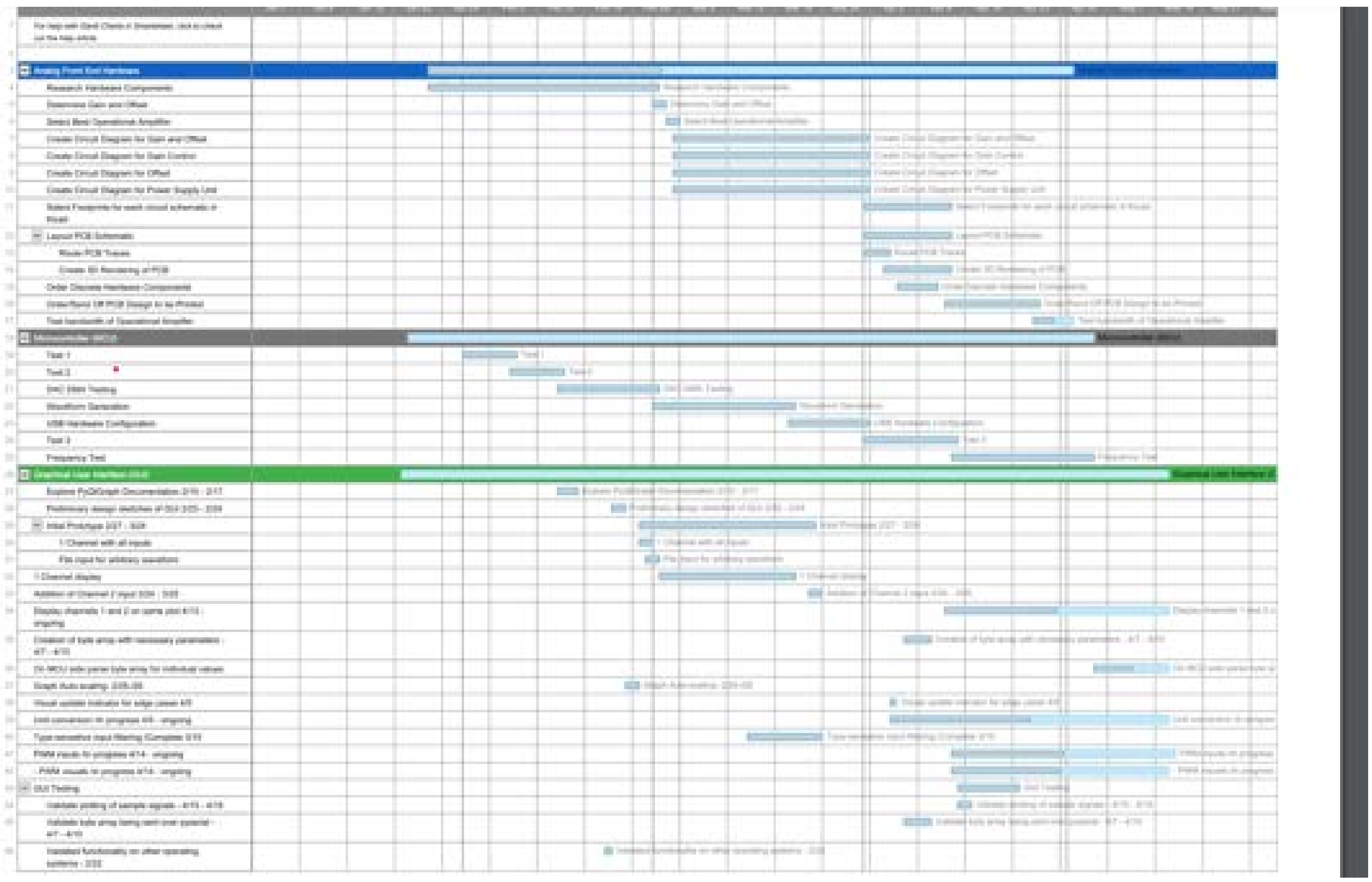
9.2 Course Project Plan Timeline ECE 492 (14 Weeks)



9.3 Course Project Plan Timeline ECE 493 (14 Weeks)



9.4 Detail Project Plan Timeline by Sub-System ECE 492



10. Problems Encountered

10.1 Hardware design

- Footprints not being available on Kicad for various components
- Symbols not being available on Kicad for some components - having to custom draw/create
- Compactness of PCB Design could cause problems with the assembly and make soldering a bit more difficult
- Lead Times on Ordering Printed Circuit Boards
- Size of discrete components

10.2 MCU

- Output Buffer limiting signal to 250 kHz
- Data not being received over serial
- USB configuration requires hardware changes
- Debugging is difficult

10.3 GUI

- Packets of data were hanging when sent to the MCU after the first packet successfully transmitted.
- Higher frequencies resulted in a poorly sampled graph to be displayed on the GUI, this graph did not resemble the intended graph.
- Data that was either too low or too high were being able to be entered despite safeguards implemented.
- Storing the key parameters faulted at occasions after the initial waveform parameters were saved.
- When connecting the MCU to establish communication, different comm ports were selected on different devices.

11. References

[1] "ADALM2000," *Evaluation Board | Analog Devices*, 09-Jan-2023. [Online]. Available: https://www.analog.com/en/design-center/evaluation-hardware-and-software/evaluation-boards-kits/adalm2000.html#eb-documentation. [Accessed: 26-Feb-2023].

[2] "Campus covid-19 data archive," *George Mason University*. [Online]. Available: https://www.gmu.edu/campus-covid-19-data-archive. [Accessed: 26-Feb-2023].

[3] "Dataman 531 arbitrary waveform generator," *Device Programmers & ISP Programming by Dataman Programmers*. [Online]. Available: https://www.dataman.com/dataman-531-arbitrary-waveform-generator.html. [Accessed: 26-Feb-2023].

[4] *Hantek2000 series - hantek electronic & your testing solution provider*. [Online]. Available: http://www.hantek.com/products/detail/13174. [Accessed: 26-Feb-2023].

[5] "Picoscope 2000 specifications," *PicoScope 2000 Specifications | Pico Technology*. [Online]. Available: https://www.picotech.com/oscilloscope/2000/picoscope-2000-specifications. [Accessed: 26-Feb-2023].

[6] S. K, "Analog discovery 2," *Analog Discovery 2 - Digilent Reference*. [Online]. Available: https://digilent.com/reference/test-and-measurement/analog-discovery-2/start. [Accessed: 26-Feb-2023].

[7] "Wiki," *Reference Manual (Understanding the Internals) [Analog Devices Wiki]*. [Online]. Available: https://wiki.analog.com/university/tools/m2k/users/reference_manual. [Accessed: 26-Feb-2023].

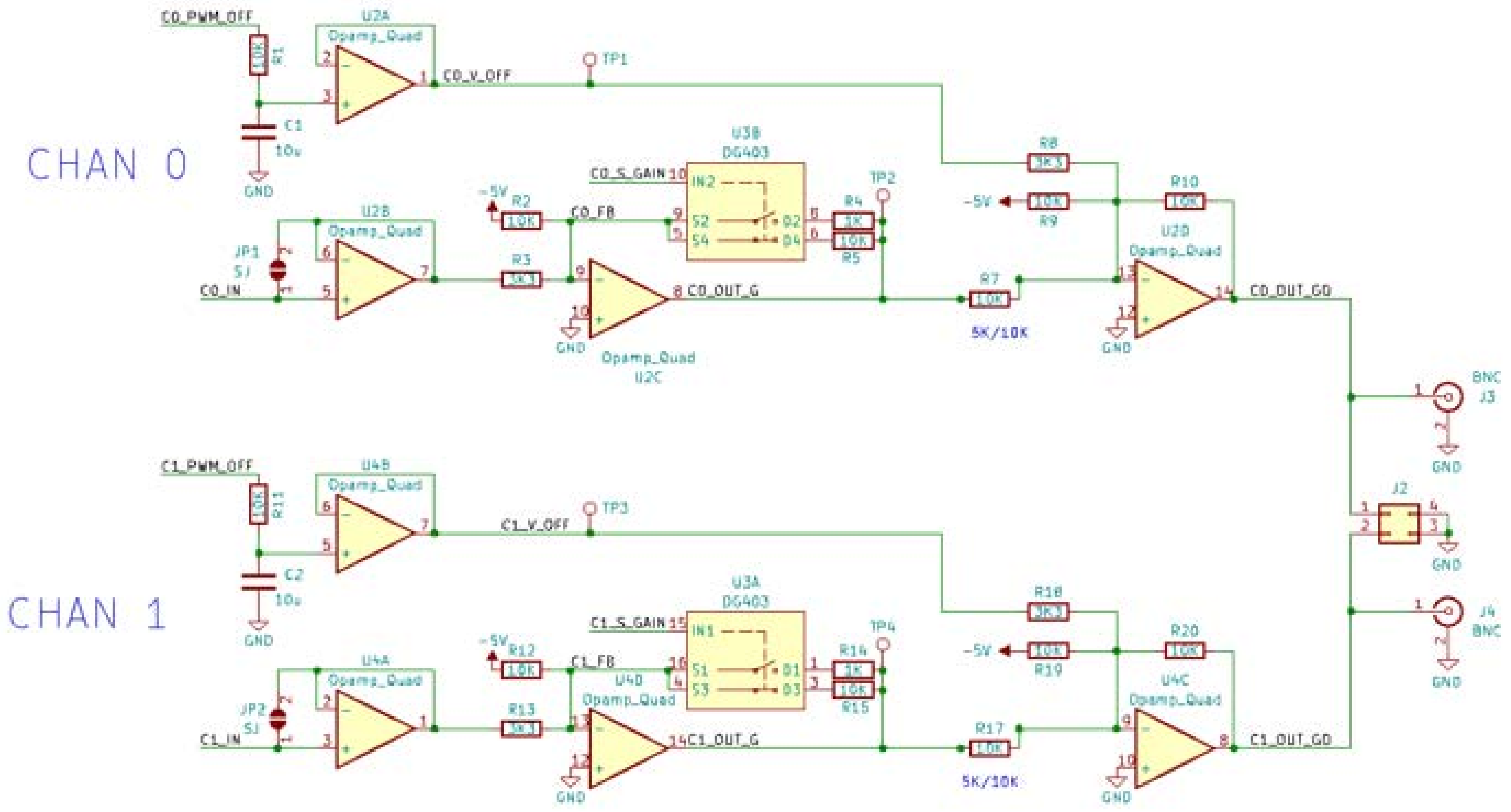
16. Appendix C: Schematic

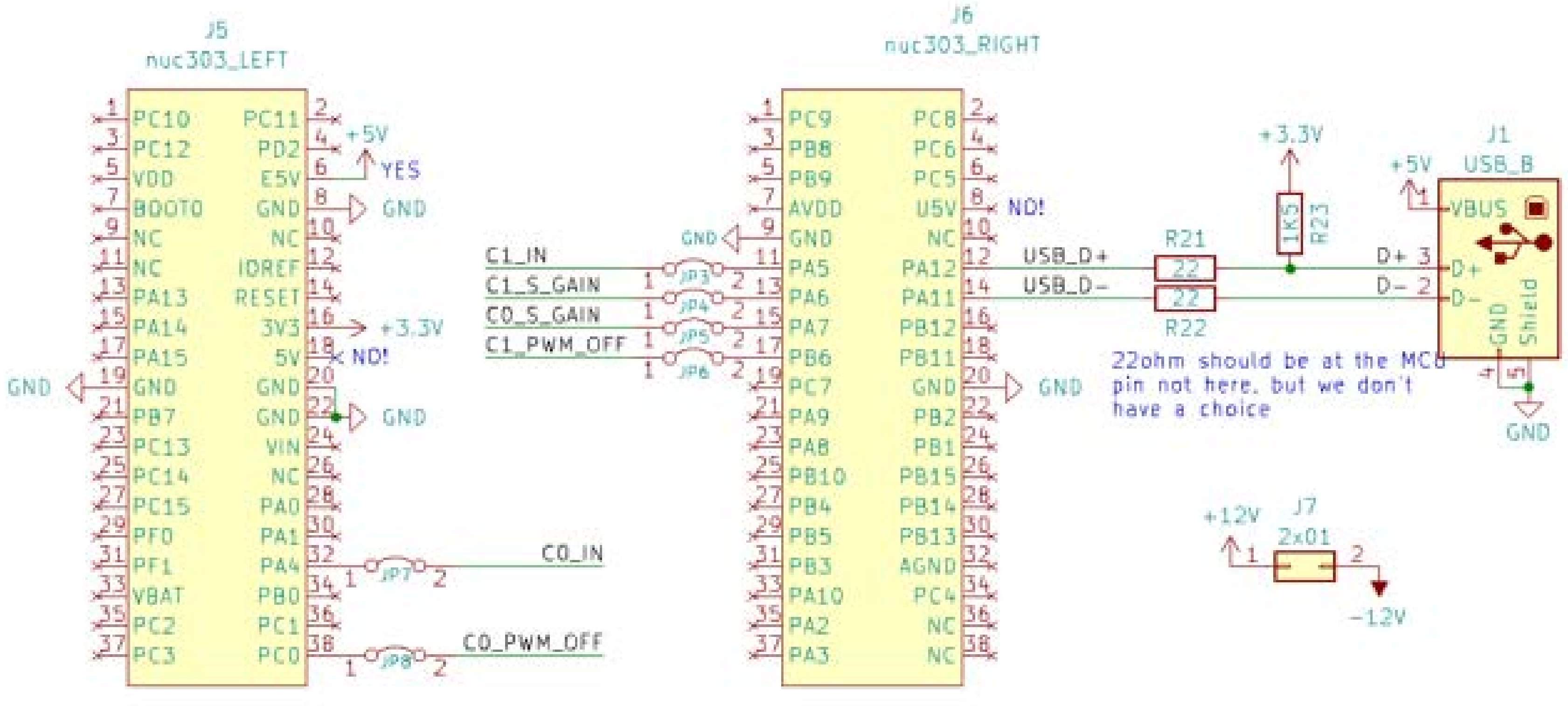
Page 1: Analog Front end

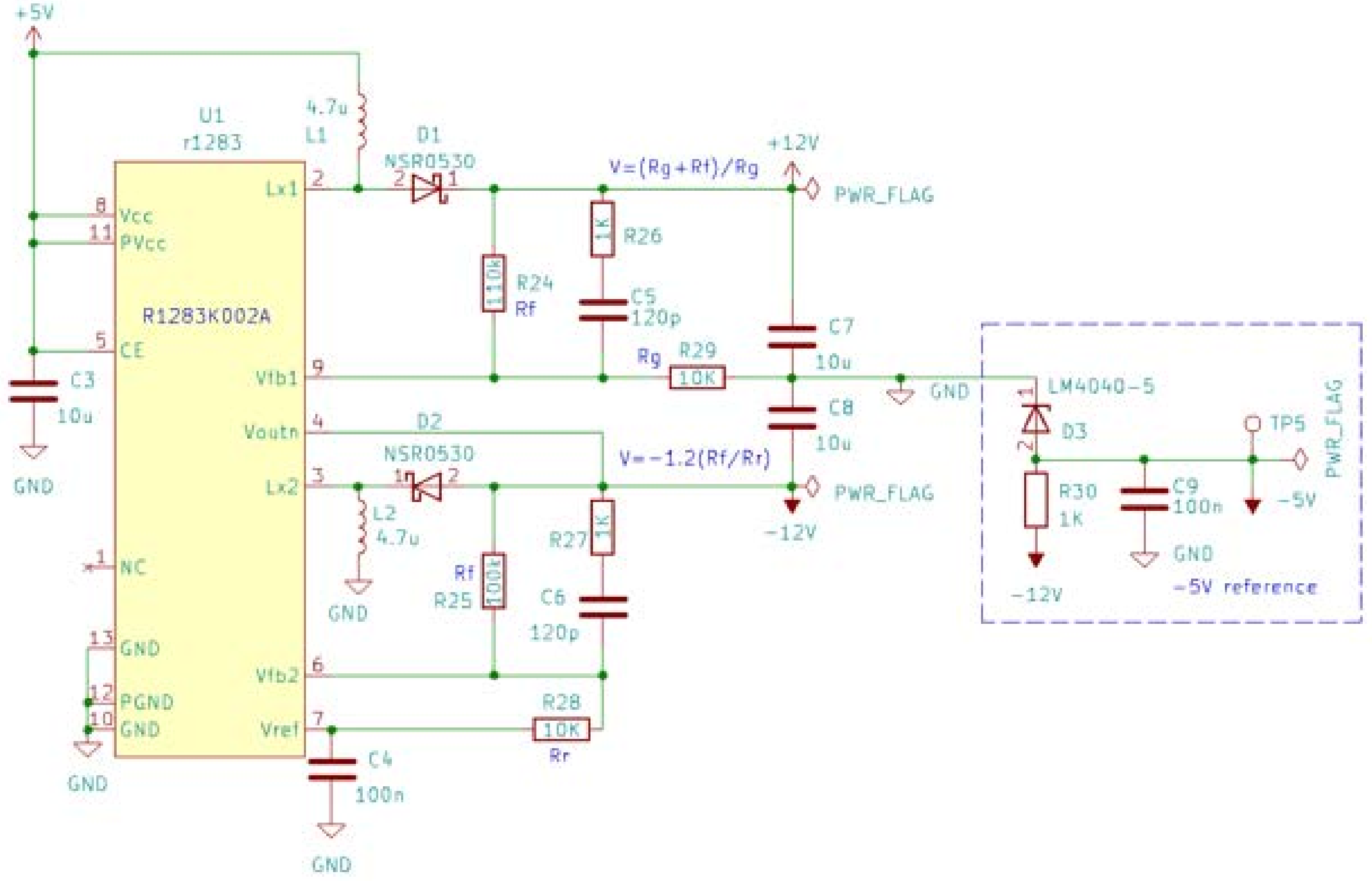
Page 2:Connectors

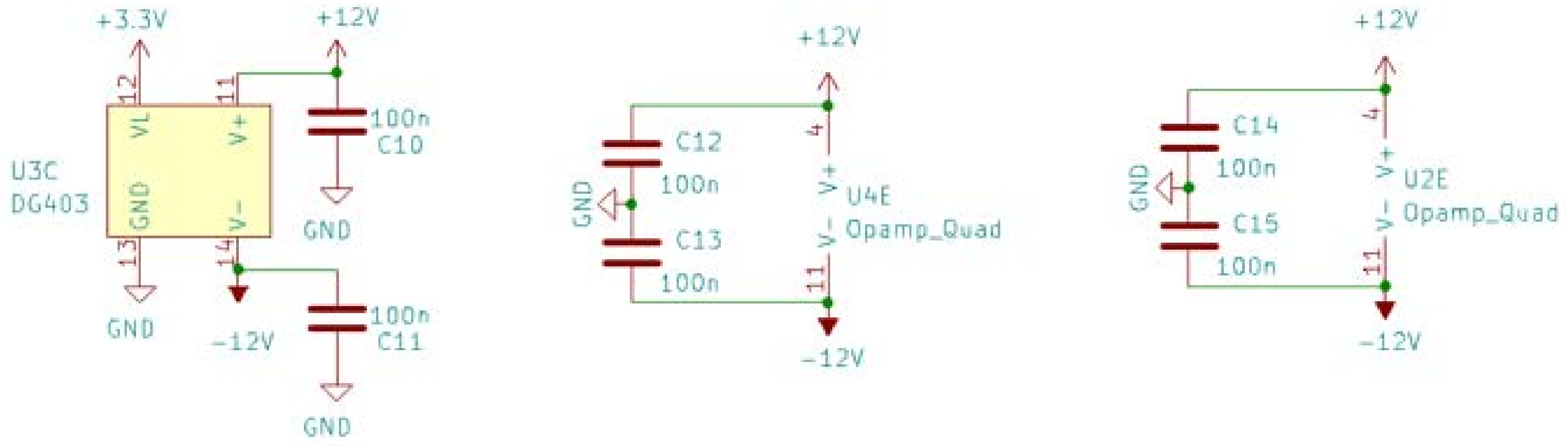
Page 3: PSU

Page 4: Misc


CHAN 0
CHAN 1
C0_PWM_OFF
U2A
Opamp_Quad
C0_V_OFF
TP1
C1
10u
GND
R1
10K
U3B
DG403
C0_S_GAIN
IN2
S2
S4
D2
D4
R4
1K
R5
10K
TP2
R8
3K3
R9
10K
-5V
R10
10K
U2D
Opamp_Quad
U2B
Opamp_Quad
JP1
SJ
C0_IN
R2
10K
C0_FB
R3
3K3
C0_OUT_G
U2C
R7
10K
5K/10K
C0_OUT_GO
BNC
J3
J2
J4
C1_PWM_OFF
U4B
R11
C1_V_OFF
TP3
C2
U3A
C1_S_GAIN
IN1
S1
S3
D1
D3
R14
R15
TP4
R18
R19
R20
U4C
U4A
JP2
C1_IN
R12
C1_FB
U4D
R13
C1_OUT_G
R17
C1_OUT_GO


J5
nuc303_LEFT
PC10
PC12
VDD
BOOT0
NC
NC
PA13
PA14
PA15
GND
PB7
PC13
PC14
PC15
PF0
PF1
VBAT
PC2
PC3
PC11
PD2
E5V
GND
NC
IOREF
RESET
3V3
5V
GND
GND
VIN
NC
PA0
PA1
PA4
PB0
PC1
PC0
+5V
YES
GND
+3.3V
ND!
C0_IN
C0_PWM_OFF
JP7
JP8
J6
nuc303_RIGHT
PC9
PB8
PB9
AVDD
GND
PA5
PA6
PA7
PB6
PC7
PA9
PA8
PB10
PB4
PB5
PB3
PA10
PA2
PA3
PC8
PC6
PC5
U5V
NC
PA12
PA11
PB12
PB11
GND
PB2
PB1
PB15
PB14
PB13
AGND
PC4
NC
NC
C1_IN
C1_S_GAIN
C0_S_GAIN
C1_PWM_OFF
JP3
JP4
JP5
JP6
USB_D+
USB_D-
R21
R22
22
22
+3.3V
1K5
R23
J1
USB_B
VBUS
D+
D-
GND
Shield
22ohm should be at the MCU pin not here, but we don't have a choice
J7
2x01
+12V
-12V


+5V
U1
r1283
R1283K002A
4.7u
L1
D1
NSR0530
V=(Rg+Rf)/Rg
+12V
PWR_FLAG
R26
1K
R24
110k
Rf
C5
120p
C7
10u
Rg
R29
10K
GND
C8
10u
LM4040-5
D3
TP5
PWR_FLAG
R30
1K
C9
100n
-5V
-12V
-5V reference
Vcc
PVcc
CE
C3
10u
Vfb1
Voutn
D2
NSR0530
V=-1.2(Rf/Rr)
Lx1
Lx2
L2
4.7u
R27
1K
R25
100k
C6
120p
NC
GND
PGND
Vfb2
Vref
R28
10K
Rr
C4
100n


+3.3V
+12V
12
11
VL
V+
U3C
DG403
GND
V-
13
14
100n
C10
GND
GND
-12V
100n
C11
GND
+12V
C12
100n
GND
C13
100n
4
V+
U4E
V-
Opamp_Quad
11
-12V
+12V
C14
100n
GND
C15
100n
4
V+
U2E
V-
Opamp_Quad
11
-12V

17. Appendix D: Code Listing MCU

17.1 Interface.c

```c
#include "interface.h"
#include "stm32f3xx.h"
#include "usb_device.h"

uint8_t awg_lut[AWG_NUM_CHAN][AWG_SAMPLES*2];

uint16_t BULK_BUFF_RECV = 0;
uint8_t *BULK_BUFF;

extern DAC_HandleTypeDef hdac1;

extern TIM_HandleTypeDef htim2;
extern TIM_HandleTypeDef htim6;
extern TIM_HandleTypeDef htim7;
extern DMA_HandleTypeDef hdma_dac1_ch1;
extern DMA_HandleTypeDef hdma_dac1_ch2;


const uint8_t ACK_STRING[ACK_STRING_LEN] = {'S', 'T', 'M', 'A', 'W', 'G',
'2', '3'};
const uint8_t HS_STRING[HS_STRING_LEN] = {'I', 'N', 'I', 'T'};

void SendAck(){
      TRANS_Packet pack;
      pack.packet_type = 0;
      memcpy(pack.ack_string, ACK_STRING, ACK_STRING_LEN);
      if (CDC_Transmit_FS(&pack, sizeof(TRANS_Packet))) {
      //printLine("BUSY");
      }
}

uint16_t numSamples[AWG_NUM_CHAN];
uint16_t phaseARR[AWG_NUM_CHAN];
uint16_t ARR_hold[AWG_NUM_CHAN];

void GotCDC_64B_Packet(char *ptr) {
      if (!BULK_BUFF_RECV) {
      RECV_Packet *packet = (RECV_Packet *) ptr;
      if (packet->packet_type == 0) {
            // Handle Handshake packet as before
            uint8_t *magic = &(packet->Content.HandShake.handshake_string);

            int match = 1;
            for (int i = 0; i < HS_STRING_LEN; i++) {
                  if (magic[i] != HS_STRING[i]) match = 0;
            }
            if (match) {
                  SendAck();
            }
      } else if (packet->packet_type == 1) {
            uint8_t chan = packet->Content.AWG_SET.channel;
            uint16_t PSC = packet->Content.AWG_SET.PSC;
            uint16_t ARR = packet->Content.AWG_SET.ARR;
            uint16_t CCR_offset = packet->Content.AWG_SET.CCR_offset;
            numSamples[chan] = packet->Content.AWG_SET.numSamples;
            phaseARR[chan] = packet->Content.AWG_SET.phaseARR;
            uint8_t gain = packet->Content.AWG_SET.gain;




            BULK_BUFF_RECV = numSamples[chan] < 32 ? 128 : numSamples[chan]
*2;

            BULK_BUFF = (uint8_t *) awg_lut[chan];

            if(chan == 0){
                   TIM2->CCR1 = CCR_offset;
                   TIM6->ARR = ARR;
                   TIM6->PSC = PSC;
                   ARR_hold[0] = ARR;
                   HAL_GPIO_WritePin(GAIN_C0_GPIO_Port, GAIN_C0_Pin, gain);
            }else{
                   TIM2->CCR2 = CCR_offset;
                   TIM7->ARR = ARR;
                   TIM7->PSC = PSC;

                   ARR_hold[1] = ARR;
                   HAL_GPIO_WritePin(GAIN_C1_GPIO_Port, GAIN_C1_Pin, gain);
            }
            //restart both channels to get correct phase
             //stop both timers (without using HAL_TIM_Base_Stop to prevent
side effects)
              __HAL_TIM_DISABLE(&htim6);
              __HAL_TIM_DISABLE(&htim7);

             //restart both DMAs
            HAL_DAC_Stop_DMA(&hdac1, DAC_CHANNEL_1);
             HAL_DAC_Stop_DMA(&hdac1, DAC_CHANNEL_2);
             HAL_DAC_Start_DMA(&hdac1, DAC_CHANNEL_1,
(uint32_t*)awg_lut[0], numSamples[0], DAC_ALIGN_12B_R);
             HAL_DAC_Start_DMA(&hdac1, DAC_CHANNEL_2,
(uint32_t*)awg_lut[1], numSamples[1], DAC_ALIGN_12B_R);

             //reset counters, otherwise prescale counter value can mess up
wphase
             TIM6 -> EGR = TIM_EGR_UG;
             TIM7 -> EGR = TIM_EGR_UG;

             //set clock phase
            TIM6->CNT = phaseARR[0] - ARR_hold[1];
            TIM7->CNT = phaseARR[1] - ARR_hold[0];
             //TIM6->CNT = 0;
             //TIM7->CNT = 6;

            //restart both timers (again without HAL_TIM_Base_Start).
            //The generated asm code should enable both within two
instruction
            //this code is a bit loony and isn't perfectly synchronized
anyway
            volatile uint32_t *CCR6_add = &(htim6.Instance->CR1);
                   uint32_t CCR6_new = *CCR6_add | TIM_CR1_CEN;
                   volatile uint32_t *CCR7_add = &(htim7.Instance->CR1);
                   uint32_t CCR7_new = *CCR7_add | TIM_CR1_CEN;
                   *CCR6_add = CCR6_new;
                   *CCR7_add = CCR7_new;
                   //restart of both channels complete
      }
      } else {
      memcpy(BULK_BUFF, ptr, 64);
      BULK_BUFF += 64;
      BULK_BUFF_RECV -= 64;
      if (!BULK_BUFF_RECV) {
            SendAck();
      }
      }
}
```

17.2 Interface.h

```
#ifndef SRC_INTERFACE_H_
#define SRC_INTERFACE_H_

#include <stdint.h>

#define AWG_SAMPLES (1024 * 4)
#define AWG_NUM_CHAN 2
#define MAGIC_NUM 0x42
#define HS_STRING_LEN 4
#define ACK_STRING_LEN 8

#define PACK __attribute__((packed))

extern uint8_t awg_lut[AWG_NUM_CHAN][AWG_SAMPLES*2];

typedef struct {
      uint8_t packet_type;
      union {
      struct { // packet_type = 0
            uint8_t handshake_string[HS_STRING_LEN];
      } PACK HandShake;
      struct { // packet_type = 1
            uint8_t channel;
            uint8_t gain;
            //uint8_t temp;
            uint16_t PSC;
            uint16_t ARR;
            uint16_t CCR_offset;
            uint16_t numSamples;
            uint16_t phaseARR;
      } PACK AWG_SET;
      } PACK Content;
} PACK RECV_Packet;

typedef struct PACK {
      uint8_t packet_type;
      uint8_t ack_string[ACK_STRING_LEN];
} TRANS_Packet;

void GotCDC_64B_Packet(char *ptr);

#endif /* SRC_INTERFACE_H_ */
```

17.3 main.c

```
/* USER CODE BEGIN Header */
/**

****************************************************************************
***
 * @file           : main.c
 * @brief          : Main program body

****************************************************************************
***
 * @attention
 *
 * Copyright (c) 2023 STMicroelectronics.
 * All rights reserved.
 *
 * This software is licensed under terms that can be found in the LICENSE
file
 * in the root directory of this software component.
 * If no LICENSE file comes with this software, it is provided AS-IS.
 *

****************************************************************************
***
```

```
 */
/* USER CODE END Header */
/* Includes
------------------------------------------------------------------*/
#include "main.h"
#include "usb_device.h"

/* Private includes
----------------------------------------------------------*/
/* USER CODE BEGIN Includes */
#include "interface.h"
/* USER CODE END Includes */

/* Private typedef
-----------------------------------------------------------*/
/* USER CODE BEGIN PTD */

/* USER CODE END PTD */

/* Private define
------------------------------------------------------------*/
/* USER CODE BEGIN PD */

/* USER CODE END PD */

/* Private macro
-------------------------------------------------------------*/
/* USER CODE BEGIN PM */

/* USER CODE END PM */

/* Private variables
---------------------------------------------------------*/
DAC_HandleTypeDef hdac1;
DMA_HandleTypeDef hdma_dac1_ch1;
DMA_HandleTypeDef hdma_dac1_ch2;

TIM_HandleTypeDef htim2;
TIM_HandleTypeDef htim6;
TIM_HandleTypeDef htim7;

/* USER CODE BEGIN PV */
uint16_t dc_volt1[256] = {2048, 2048};
```

```
/* USER CODE END PV */

/* Private function prototypes
-----------------------------------------------*/
void SystemClock_Config(void);
static void MX_GPIO_Init(void);
static void MX_DMA_Init(void);
static void MX_DAC1_Init(void);
static void MX_TIM2_Init(void);
static void MX_TIM6_Init(void);
static void MX_TIM7_Init(void);
/* USER CODE BEGIN PFP */

/* USER CODE END PFP */

/* Private user code
---------------------------------------------------------*/
/* USER CODE BEGIN 0 */
/* USER CODE END 0 */

/**
  * @brief  The application entry point.
  * @retval int
  */
int main(void)
{
  /* USER CODE BEGIN 1 */

  /* USER CODE END 1 */

  /* MCU
Configuration--------------------------------------------------------*/

  /* Reset of all peripherals, Initializes the Flash interface and the
Systick. */
  HAL_Init();

  /* USER CODE BEGIN Init */

  /* USER CODE END Init */

  /* Configure the system clock */
  SystemClock_Config();

  /* USER CODE BEGIN SysInit */

  /* USER CODE END SysInit */

  /* Initialize all configured peripherals */
  MX_GPIO_Init();
  MX_DMA_Init();
  MX_USB_DEVICE_Init();
  MX_DAC1_Init();
  MX_TIM2_Init();
  MX_TIM6_Init();
  MX_TIM7_Init();
  /* USER CODE BEGIN 2 */

  HAL_DAC_Start_DMA(&hdac1, DAC_CHANNEL_1, (uint32_t*) dc_volt1, (uint32_t)
2, DAC_ALIGN_12B_R);
  HAL_TIM_Base_Start(&htim6);
  HAL_DAC_Start_DMA(&hdac1, DAC_CHANNEL_2, (uint32_t*) dc_volt1, (uint32_t)
2, DAC_ALIGN_12B_R);
  HAL_TIM_Base_Start(&htim7);


  volatile int a = offsetof(RECV_Packet, packet_type);
  volatile int b = offsetof(RECV_Packet, Content.AWG_SET.channel);
  //volatile int c = offsetof(RECV_Packet, Content.AWG_SET.temp);
  volatile int d = offsetof(RECV_Packet, Content.AWG_SET.gain);
  volatile int e = offsetof(RECV_Packet, Content.AWG_SET.PSC);
  volatile int f = offsetof(RECV_Packet, Content.AWG_SET.ARR);
  volatile int g = offsetof(RECV_Packet, Content.AWG_SET.CCR_offset);
  volatile int h = offsetof(RECV_Packet, Content.AWG_SET.numSamples);
  volatile int o = offsetof(RECV_Packet, Content.AWG_SET.phaseARR);
//  int k = offsetof(RECV_Packet, Content.AWG_SET.channel);


  static_assert(sizeof(RECV_Packet) <= 64);

    HAL_TIM_PWM_Start(&htim2, TIM_CHANNEL_1);
    HAL_TIM_PWM_Start(&htim2, TIM_CHANNEL_2);
    TIM2->CCR1 = 2048;
    TIM2->CCR2 = 2048;
  /* USER CODE END 2 */

  /* Infinite loop */
  /* USER CODE BEGIN WHILE */
    while (1) {
      /* USER CODE END WHILE */

      /* USER CODE BEGIN 3 */

    }
  /* USER CODE END 3 */
}

/**
  * @brief System Clock Configuration
  * @retval None
  */
void SystemClock_Config(void)
{
  RCC_OscInitTypeDef RCC_OscInitStruct = {0};
  RCC_ClkInitTypeDef RCC_ClkInitStruct = {0};
  RCC_PeriphCLKInitTypeDef PeriphClkInit = {0};

  /** Initializes the RCC Oscillators according to the specified parameters
  * in the RCC_OscInitTypeDef structure.
  */
  RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_HSE;
  RCC_OscInitStruct.HSEState = RCC_HSE_ON;
  RCC_OscInitStruct.HSIState = RCC_HSI_ON;
  RCC_OscInitStruct.PLL.PLLState = RCC_PLL_ON;
  RCC_OscInitStruct.PLL.PLLSource = RCC_PLLSOURCE_HSE;
  RCC_OscInitStruct.PLL.PLLMUL = RCC_PLL_MUL9;
  RCC_OscInitStruct.PLL.PREDIV = RCC_PREDIV_DIV1;
  if (HAL_RCC_OscConfig(&RCC_OscInitStruct) != HAL_OK)
  {
      Error_Handler();
  }

  /** Initializes the CPU, AHB and APB buses clocks
  */
  RCC_ClkInitStruct.ClockType = RCC_CLOCKTYPE_HCLK|RCC_CLOCKTYPE_SYSCLK
                              |RCC_CLOCKTYPE_PCLK1|RCC_CLOCKTYPE_PCLK2;
  RCC_ClkInitStruct.SYSCLKSource = RCC_SYSCLKSOURCE_PLLCLK;
  RCC_ClkInitStruct.AHBCLKDivider = RCC_SYSCLK_DIV1;
  RCC_ClkInitStruct.APB1CLKDivider = RCC_HCLK_DIV2;
  RCC_ClkInitStruct.APB2CLKDivider = RCC_HCLK_DIV1;

  if (HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_2) != HAL_OK)
  {
      Error_Handler();
  }
  PeriphClkInit.PeriphClockSelection =
RCC_PERIPHCLK_USB|RCC_PERIPHCLK_TIM2;
  PeriphClkInit.USBClockSelection = RCC_USBCLKSOURCE_PLL_DIV1_5;
  PeriphClkInit.Tim2ClockSelection = RCC_TIM2CLK_HCLK;
  if (HAL_RCCEx_PeriphCLKConfig(&PeriphClkInit) != HAL_OK)
  {
      Error_Handler();
  }
}

/**
  * @brief DAC1 Initialization Function
  * @param None
  * @retval None
  */
static void MX_DAC1_Init(void)
{

  /* USER CODE BEGIN DAC1_Init 0 */

  /* USER CODE END DAC1_Init 0 */

  DAC_ChannelConfTypeDef sConfig = {0};

  /* USER CODE BEGIN DAC1_Init 1 */

  /* USER CODE END DAC1_Init 1 */

  /** DAC Initialization
  */
  hdac1.Instance = DAC1;
  if (HAL_DAC_Init(&hdac1) != HAL_OK)
  {
      Error_Handler();
  }
  /** DAC channel OUT1 config
  */
  sConfig.DAC_Trigger = DAC_TRIGGER_T6_TRGO;
  sConfig.DAC_OutputBuffer = DAC_OUTPUTBUFFER_DISABLE;
  if (HAL_DAC_ConfigChannel(&hdac1, &sConfig, DAC_CHANNEL_1) != HAL_OK)
  {
      Error_Handler();
  }

  /** DAC channel OUT2 config
  */
  sConfig.DAC_Trigger = DAC_TRIGGER_T7_TRGO;
  if (HAL_DAC_ConfigChannel(&hdac1, &sConfig, DAC_CHANNEL_2) != HAL_OK)
  {
      Error_Handler();
  }
  /* USER CODE BEGIN DAC1_Init 2 */

  /* USER CODE END DAC1_Init 2 */

}

/**
  * @brief TIM2 Initialization Function
  * @param None
  * @retval None
  */
static void MX_TIM2_Init(void)
{

  /* USER CODE BEGIN TIM2_Init 0 */

  /* USER CODE END TIM2_Init 0 */

  TIM_ClockConfigTypeDef sClockSourceConfig = {0};
  TIM_MasterConfigTypeDef sMasterConfig = {0};
  TIM_OC_InitTypeDef sConfigOC = {0};

  /* USER CODE BEGIN TIM2_Init 1 */

  /* USER CODE END TIM2_Init 1 */
  htim2.Instance = TIM2;
  htim2.Init.Prescaler = 0;
  htim2.Init.CounterMode = TIM_COUNTERMODE_UP;
  htim2.Init.Period = 4096-1;
  htim2.Init.ClockDivision = TIM_CLOCKDIVISION_DIV1;
  htim2.Init.AutoReloadPreload = TIM_AUTORELOAD_PRELOAD_DISABLE;
  if (HAL_TIM_Base_Init(&htim2) != HAL_OK)
  {
      Error_Handler();
  }
  sClockSourceConfig.ClockSource = TIM_CLOCKSOURCE_INTERNAL;
  if (HAL_TIM_ConfigClockSource(&htim2, &sClockSourceConfig) != HAL_OK)
  {
      Error_Handler();
  }
  if (HAL_TIM_PWM_Init(&htim2) != HAL_OK)
  {
      Error_Handler();
  }
  sMasterConfig.MasterOutputTrigger = TIM_TRGO_RESET;
  sMasterConfig.MasterSlaveMode = TIM_MASTERSLAVEMODE_DISABLE;
  if (HAL_TIMEx_MasterConfigSynchronization(&htim2, &sMasterConfig) !=
HAL_OK)
  {
      Error_Handler();
  }
  sConfigOC.OCMode = TIM_OCMODE_PWM1;
  sConfigOC.Pulse = 0;
  sConfigOC.OCPolarity = TIM_OCPOLARITY_HIGH;
  sConfigOC.OCFastMode = TIM_OCFAST_DISABLE;
  if (HAL_TIM_PWM_ConfigChannel(&htim2, &sConfigOC, TIM_CHANNEL_1) !=
HAL_OK)
  {
      Error_Handler();
  }
  if (HAL_TIM_PWM_ConfigChannel(&htim2, &sConfigOC, TIM_CHANNEL_2) !=
HAL_OK)
  {
      Error_Handler();
  }
  /* USER CODE BEGIN TIM2_Init 2 */

  /* USER CODE END TIM2_Init 2 */
  HAL_TIM_MspPostInit(&htim2);
```

```
}

/**
  * @brief TIM6 Initialization Function
  * @param None
  * @retval None
  */
static void MX_TIM6_Init(void)
{

  /* USER CODE BEGIN TIM6_Init 0 */

  /* USER CODE END TIM6_Init 0 */

  TIM_MasterConfigTypeDef sMasterConfig = {0};

  /* USER CODE BEGIN TIM6_Init 1 */

  /* USER CODE END TIM6_Init 1 */
  htim6.Instance = TIM6;
  htim6.Init.Prescaler = 0;
  htim6.Init.CounterMode = TIM_COUNTERMODE_UP;
  htim6.Init.Period = 65535;
  htim6.Init.AutoReloadPreload = TIM_AUTORELOAD_PRELOAD_DISABLE;
  if (HAL_TIM_Base_Init(&htim6) != HAL_OK)
  {
      Error_Handler();
  }
  sMasterConfig.MasterOutputTrigger = TIM_TRGO_UPDATE;
  sMasterConfig.MasterSlaveMode = TIM_MASTERSLAVEMODE_DISABLE;
  if (HAL_TIMEx_MasterConfigSynchronization(&htim6, &sMasterConfig) !=
HAL_OK)
  {
      Error_Handler();
  }
  /* USER CODE BEGIN TIM6_Init 2 */

  /* USER CODE END TIM6_Init 2 */

}

/**
```

```
  * @brief TIM7 Initialization Function
  * @param None
  * @retval None
  */
static void MX_TIM7_Init(void)
{

  /* USER CODE BEGIN TIM7_Init 0 */

  /* USER CODE END TIM7_Init 0 */

  TIM_MasterConfigTypeDef sMasterConfig = {0};

  /* USER CODE BEGIN TIM7_Init 1 */

  /* USER CODE END TIM7_Init 1 */
  htim7.Instance = TIM7;
  htim7.Init.Prescaler = 0;
  htim7.Init.CounterMode = TIM_COUNTERMODE_UP;
  htim7.Init.Period = 65535;
  htim7.Init.AutoReloadPreload = TIM_AUTORELOAD_PRELOAD_DISABLE;
  if (HAL_TIM_Base_Init(&htim7) != HAL_OK)
  {
      Error_Handler();
  }
  sMasterConfig.MasterOutputTrigger = TIM_TRGO_UPDATE;
  sMasterConfig.MasterSlaveMode = TIM_MASTERSLAVEMODE_DISABLE;
  if (HAL_TIMEx_MasterConfigSynchronization(&htim7, &sMasterConfig) !=
HAL_OK)
  {
      Error_Handler();
  }
  /* USER CODE BEGIN TIM7_Init 2 */

  /* USER CODE END TIM7_Init 2 */

}

/**
  * Enable DMA controller clock
  */
static void MX_DMA_Init(void)
{
  /* DMA controller clock enable */
  __HAL_RCC_DMA1_CLK_ENABLE();
  __HAL_RCC_DMA2_CLK_ENABLE();

  /* DMA interrupt init */
  /* DMA1_Channel3_IRQn interrupt configuration */
  HAL_NVIC_SetPriority(DMA1_Channel3_IRQn, 0, 0);
  HAL_NVIC_EnableIRQ(DMA1_Channel3_IRQn);
  /* DMA2_Channel4_IRQn interrupt configuration */
  HAL_NVIC_SetPriority(DMA2_Channel4_IRQn, 0, 0);
  HAL_NVIC_EnableIRQ(DMA2_Channel4_IRQn);

}

/**
  * @brief GPIO Initialization Function
  * @param None
  * @retval None
  */
static void MX_GPIO_Init(void)
{
  GPIO_InitTypeDef GPIO_InitStruct = {0};
/* USER CODE BEGIN MX_GPIO_Init_1 */
/* USER CODE END MX_GPIO_Init_1 */

  /* GPIO Ports Clock Enable */
  __HAL_RCC_GPIOC_CLK_ENABLE();
  __HAL_RCC_GPIOF_CLK_ENABLE();
  __HAL_RCC_GPIOA_CLK_ENABLE();
  __HAL_RCC_GPIOB_CLK_ENABLE();

  /*Configure GPIO pin Output Level */
  HAL_GPIO_WritePin(GPIOA, GAIN_C1_Pin|GAIN_C0_Pin, GPIO_PIN_RESET);

  /*Configure GPIO pin : B1_Pin */
  GPIO_InitStruct.Pin = B1_Pin;
  GPIO_InitStruct.Mode = GPIO_MODE_IT_FALLING;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  HAL_GPIO_Init(B1_GPIO_Port, &GPIO_InitStruct);

  /*Configure GPIO pins : GAIN_C1_Pin GAIN_C0_Pin */
  GPIO_InitStruct.Pin = GAIN_C1_Pin|GAIN_C0_Pin;
  GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
  HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);

/* USER CODE BEGIN MX_GPIO_Init_2 */
/* USER CODE END MX_GPIO_Init_2 */
}

/* USER CODE BEGIN 4 */

/* USER CODE END 4 */

/**
  * @brief  This function is executed in case of error occurrence.
  * @retval None
  */
void Error_Handler(void)
{
  /* USER CODE BEGIN Error_Handler_Debug */
    /* User can add his own implementation to report the HAL error return
state */
    __disable_irq();
    while (1) {
    }
  /* USER CODE END Error_Handler_Debug */
}

#ifdef  USE_FULL_ASSERT
/**
  * @brief  Reports the name of the source file and the source line number
  *         where the assert_param error has occurred.
  * @param  file: pointer to the source file name
  * @param  line: assert_param error line source number
  * @retval None
  */
void assert_failed(uint8_t *file, uint32_t line)
{
  /* USER CODE BEGIN 6 */
  /* User can add his own implementation to report the file name and line
number,
      ex: printf("Wrong parameters value: file %s on line %d\r\n", file,
line) */
```

```
  /* USER CODE END 6 */
}
#endif /* USE_FULL_ASSERT */
```

18. Appendix E: Code Listing GUI

18.1 Main.py

```
import importlib
import subprocess
import sys
import os
import wave_drawer
import subprocess
import math
import numpy as np
import pyqtgraph as pg
from connection import Connection
from pyqtgraph.Qt import QtCore, QtWidgets
from PyQt6.QtCore import pyqtSignal
from PyQt6.QtWidgets import (
   QApplication, QWidget, QLineEdit, QToolBar, QPushButton, QVBoxLayout, QFrame, QLabel,
QMessageBox,QComboBox
)
from PyQt6.QtCore import Qt
from PyQt6 import QtGui
from wavegen import generateSamples
import threading
import platform
from channel import Channel

#why windows why?
if platform.system() == "Windows":
   import ctypes
   myappid = u'mycompany.myproduct.subproduct.version' # arbitrary string
   ctypes.windll.shell32.SetCurrentProcessExplicitAppUserModelID(myappid)

def resource_path(relative_path):
   """Get absolute path to a resource, works for dev and for PyInstaller

   Parameters:
   relative_path (str): relative path to requested resource

   Returns:
   absolute path to resource that can be used when running standalone or with pyinstaller
```

```
    """
    base_path = getattr(sys, '_MEIPASS', os.path.dirname(os.path.abspath(__file__)))
    return os.path.join(base_path, relative_path)

class WaveformGenerator(QtWidgets.QWidget):
    """Handles the main window for the waveform generator application """

    statusCallbackSignal = pyqtSignal(str, str)
    """Used to pass signals about connection updates between different threads """

    def statusCallback(self, status, message):
        """Callback function for the serial connection. Updates the status label and
enables/disables the connect button.

        Parameters:
        status (str): The new status of the connection
        message (str): if not null, a popup with this message will be shown
        """
        self.status_label.setText("Status: " + status)
        self.connectButton.setEnabled(status == "disconnected")
        if message:
            pg.QtWidgets.QMessageBox.critical(self, 'Error', message)
        if status == "connected":
            #reset channels to off when connected
            for c in self.channels:
                c.setRunningStatus(False)

    def connectButtonClicked(self):
        """Function called when the connect button is clicked. Attempts to connect to the
device."""
        self.connectButton.setEnabled(False)
        self.conn.tryConnect()

    def setSyncStatus(self, status):
        """ Updates the sync status UI

        Parameters:
        status (bool): True/False depending on if sync is enabled
        """
        if status:
            self.synced_status.setText("Synced")
            self.synced_status.setStyleSheet("color : green")
        else:
            self.synced_status.setText("Not Synced")
            self.synced_status.setStyleSheet("color : red")

    def updateWave(self, changed = -1):
        """ Called when a channel changed a wave or when the sync setting is changed. Takes
the waves settings and sends them to connection

        Parameters:
        changed (int): Indicates the channel (0 or 1) that was changed. -1 indicates both
channels need to be updates (due to change of sync settings)
        """
        set = [self.channels[0].waveSettings, self.channels[1].waveSettings]
        syncNotPossible = (set[0].type == "dc" or set[1].type == "dc")
        #the user doesn't want sync or sync is not possible
        if not(self.syncButton.isChecked()) or syncNotPossible:
            #updates waves
            if changed == 0 or changed == -1:
                self.conn.sendWave(0, freq = set[0].freq, wave_type = set[0].type, amplitude =
set[0].amp, offset = set[0].offset, arbitrary_waveform = set[0].arb, duty = set[0].duty,
phase = set[0].phase, )
            if changed == 1 or changed == -1:
                self.conn.sendWave(1, freq = set[1].freq, wave_type = set[1].type, amplitude =
set[1].amp, offset = set[1].offset, arbitrary_waveform = set[1].arb, duty = set[1].duty,
phase = set[1].phase)

            if syncNotPossible:
                self.setSyncStatus(False)
            else: #check if there is sync
                ns0, arr0, psc0 = self.conn.calc_val(set[0].freq)
                ns1, arr1, psc1 = self.conn.calc_val(set[1].freq)
                synced = (ns1 * (arr1 + 1) * (psc1 + 1)) / (ns0 * (arr0 + 1) * (psc0 + 1)) ==
(set[0].freq / set[1].freq)
                self.setSyncStatus(synced)
        #user wants sync
        else:
            #find multiple of frequencies
            min = None
            for a in range(1, 101):
                for b in range(1, 101):
                    if a/b == set[0].freq / set[1].freq:
                        if min == None or min[0] * min[1] > a*b:
                            min = (a, b)
            if min == None: #sync is still not possible
                self.conn.sendWave(0, freq = set[0].freq, wave_type = set[0].type, amplitude =
 set[0].amp, offset = set[0].offset, arbitrary_waveform = set[0].arb, duty = set[0].duty,
 phase = set[0].phase)
                self.conn.sendWave(1, freq = set[1].freq, wave_type = set[1].type, amplitude =
 set[1].amp, offset = set[1].offset, arbitrary_waveform = set[1].arb, duty = set[1].duty,
 phase = set[1].phase)
                self.setSyncStatus(False)
            else:  #send synchronous waves
                a, b = min
                f_comm = set[0].freq / a #same as set[1].freq / b
                self.conn.sendWave(0, f_comm, wave_type = set[0].type, amplitude = set[0].amp,
 offset = set[0].offset, arbitrary_waveform = set[0].arb, duty = set[0].duty, phase =
 set[0].phase, numPeriods = a)
                self.conn.sendWave(1, f_comm, wave_type = set[1].type, amplitude = set[1].amp,
 offset = set[1].offset, arbitrary_waveform = set[1].arb, duty = set[1].duty, phase =
 set[1].phase, numPeriods = b)
                self.setSyncStatus(True)

    def __init__(self):
        """ Initializes the window."""
        super().__init__()

        self.setWindowTitle('Waveform Generator')

        #uses grid layout, should be replaced with a hierarchy of vertical/horizontal layouts
        self.grid_layout = QtWidgets.QGridLayout()

        #load the icons for file types and more
        self.setWindowIcon(QtGui.QIcon(resource_path("icon/icon.ico")))
        icons = {}
        for x in ["run", "stop", "sine", "tri", "saw", "square", "arb", "dc"]:
            icons[x] = QtGui.QIcon(resource_path("icon/" + x + ".png"))

        self.grid_layout.addWidget(QtWidgets.QLabel("Waveform generator"), 0, 0, 1, 1)

        self.synced_status = QtWidgets.QLabel("")
        self.grid_layout.addWidget(self.synced_status, 0, 5, 1, 1)
        self.setSyncStatus(False)

        self.syncButton = QtWidgets.QCheckBox  ("Force Sync")
        self.grid_layout.addWidget(self.syncButton, 0, 6, 1, 1)
        self.syncButton.clicked.connect(lambda: self.updateWave(-1))

        self.open_drawer = QtWidgets.QPushButton('Open Wave Drawer')
        self.grid_layout.addWidget(self.open_drawer, 0, 1)
        self.open_drawer.clicked.connect(self.fun_open_drawer)
        #init connection object
        self.statusCallbackSignal.connect(self.statusCallback)
        self.conn = Connection(self.statusCallbackSignal)
        #init channel objects
        self.channels = []
        for i in range(2):
            self.channels += [Channel(i, self.grid_layout, icons, self.updateWave)]
        #this must be done AFTER both channels are created otherwise updateWave() will be
 called while the second channel is still not initialized
        for c in self.channels:
            c.enableUpdates()

        self.status_label = QtWidgets.QLabel("Status: disconnected")
        self.grid_layout.addWidget(self.status_label, 19, 0, 1, 1)

        self.connectButton = QtWidgets.QPushButton("Connect")
        self.connectButton.setEnabled(False)
        self.grid_layout.addWidget(self.connectButton, 19, 1, 1, 1)
        self.connectButton.clicked.connect(self.connectButtonClicked)

        # Theme H Layout
        themeLayout = QtWidgets.QHBoxLayout()

        # Dropdown label
        themeLabel = QtWidgets.QLabel("Theme:")
        themeLayout.addWidget(themeLabel)

        # Theme Dropdown
        self.themeDropdown = QtWidgets.QComboBox()
        self.themeList = {'Default' : self.defaultTheme, 'Blue Mode' : self.lightMode, 'Dark
 Mode' : self.darkMode}
        self.themeDropdown.addItems(self.themeList.keys())
        self.themeDropdown.currentTextChanged.connect(lambda:
 self.themeList[self.themeDropdown.currentText()]())

        # Custom width dropdown menu
```

```
        self.themeDropdown.view().setFixedWidth(125)
        themeLayout.addWidget(self.themeDropdown)
        self.grid_layout.addLayout(themeLayout, 19, 2, 1, 1)

        #set grid scaling
        for i in range(1, 7):
            self.grid_layout.setColumnStretch(i, 1)
        for i in range(0, 19):
            self.grid_layout.setRowStretch(i, 1)

        self.setLayout(self.grid_layout)

        self.defaultTheme()

    def fun_open_drawer(self):
        """ Function called when the open arbitrary waveform drawer button is clicked. Opens
the arbitrary waveform drawer window."""
        drawer_window.show()

    def closeEvent(self, event):
        """ Function called when the window is closed. Closes the serial connection."""
        self.conn.close()

    def defaultTheme(self):
        """ Sets the theme to default."""
        if platform.system() != "Darwin":
            app.setStyleSheet("")
        else:
            app.setStyle("fusion")
            app.setStyleSheet("""
                QWidget {
                    font-family: 'Verdana', sans-serif;
                    font-size: 12px;
                    color: #000000;
                    background-color: #ffffff;
                }

                QLineEdit, QComboBox, QTextEdit {
                    border: 1px solid #000;
                    padding: 2px;
                    background-color: #FFFFFF;
                    color: #000000;
                }

                QPushButton {
                    font-family: 'Verdana', sans-serif;
                    color: #000000;
                    background-color: #E0E0E0;
                    border: 1px solid #000;
                    padding: 5px 10px;
                }""")

    def lightMode(self):
        """ Sets the theme to light mode."""
        app.setStyleSheet("""
            QWidget {
                font-family: 'Verdana', sans-serif;
                font-size: 12px;
                color: #000000;
                background-color: #0F3E62;
            }

            QLineEdit, QComboBox, QTextEdit {
                border: 1px solid #105C8D;
                padding: 2px;
                background-color: #FFFFFF;
                color: #000000;
                border-radius: 2px;
            }

            QPushButton {
                font-family: 'Verdana', sans-serif;
                color: #FFFFFF;
                background-color: #1874CD;
                border-radius: 5px;
                padding: 5px 10px;
                border: 1px solid #105C8D;
            }

            QPushButton:hover {
                background-color: #1C86EE;
            }

            QPushButton:disabled {
                background-color: #1874CD;
                color: #FFFFFF;
            }

            QLabel {
                color: #FFFFFF;
            }

            QLabel#freqLabel, #ampLabel, #offsetLabel, #dcLabel, #phaseLabel{
                background-color: #1874CD;
                border: 2px solid #1874CD;
                border-radius: 5px;
                padding: 2px;
                color: white;
            }
            """)
        pass

    def darkMode(self):
        """ Sets the theme to dark mode."""
        app.setStyleSheet("""
            QWidget {
                font-family: 'Verdana', sans-serif;
                font-size: 12px;
                color: #E0E0E0;
                background-color: #2C2C2C;
            }

            QLineEdit, QComboBox, QTextEdit {
                border: 1px solid #3C3C3C;
                padding: 2px;
                background-color: #2E2E2E;
                color: #E0E0E0;
            }

            QPushButton {
                color: #FFFFFF;
                background-color: #32B58F;
                border-radius: 4px;
                padding: 5px 10px;
                border: none;
            }
```

```
QPushButton:hover {
    background-color: #2D9C8F;
}

QPushButton:disabled {
    background-color: #5E5E5E;
    color: #3C3C3C;
}

QLabel {
    color: #E0E0E0;
}

QCheckBox, QRadioButton {
    color: #E0E0E0;
}

QGroupBox {
    border: 1px solid #3C3C3C;
    margin-top: 20px;
}

QGroupBox::title {
    color: #E0E0E0;
    subcontrol-origin: margin;
    left: 10px;
    padding: 0 3px 0 3px;
}

QSlider::groove:horizontal {
    border: 1px solid #3C3C3C;
    height: 8px;
    background: #2C2C2C;
    margin: 2px 0;
}

QSlider::handle:horizontal {
    background: #32B58F;
    border: 1px solid #2C2C2C;
    width: 18px;
    margin: -2px 0;
```

```
            }

            QSlider::add-page:horizontal {
                background: #555;
            }

            QSlider::sub-page:horizontal {
                background: #32B58F;
            }
            """)
        pass

if __name__ == '__main__':
    """ Main function that runs the program."""
    app = QtWidgets.QApplication(sys.argv)
    waveform_generator = WaveformGenerator()

    waveform_generator.show()
    waveform_generator.connectButtonClicked()

    drawer_window = wave_drawer.AppWindow(waveform_generator.channels)
    app.exec()
    sys.exit()
```

18.2 Wave_Drawer.py

```
import pyqtgraph as pg
import os
from PyQt6 import QtWidgets
from PyQt6.QtGui import QMouseEvent
from PyQt6.QtWidgets import QComboBox, QPushButton, QLineEdit, QMessageBox
from wavegen import generateSamples, resample, sample
import numpy as np
import math
from random import random
from PyQt6 import QtGui, QtCore

SAMPLE_POINTS = 1024*4
"""Max number of samples"""
ICON_SIZE = 64
"""Size in pixels of the icon to represent a arbitrary wave"""
```

```
class AW:
   """Stores the arbitrary waveform's name, list of samples, and icon"""

   def __init__(self, name, samples):
       """Creates an arbitrary wave with the given name and samples.

       Parameters:
           name (str): name of the wave
           samples (list of float): the initial samples for the wave
       """
       self.name = name
       self.samples = samples
       self.icon = None
       #generate samples if not provided
       if self.samples == None:
           self.samples = [0] * SAMPLE_POINTS
       self.genIcon()

   def lineDraw(self, buffer, x1, y1, x2, y2):
       """
       Draws a line on the given buffer from (x1, y1) to (x2, y2) using the DDA(?) algorithm
       """
       dx = x2 - x1
       dy = y2 - y1
       if abs(dx) > abs(dy):
           steps = abs(dx)
       else:
           steps = abs(dy)
       xincrement = dx/steps
       yincrement = dy/steps
       i = 0
       while i < steps:
           i +=1
           x1 = x1 + xincrement
           y1 = y1 + yincrement
           brush = 1
           _x = int(max(x1 - brush, 0))
           while _x < min(x1 + 1 + brush, ICON_SIZE - 1):
               _y = int(max(y1 - brush, 0))
               while _y < min(y1 + 1 + brush, ICON_SIZE - 1):
                   buffer[(_y * ICON_SIZE + _x) * 3 + 0] = 0
                   buffer[(_y * ICON_SIZE + _x) * 3 + 1] = 255
                   buffer[(_y * ICON_SIZE + _x) * 3 + 2] = 255
                   _y += 1
               _x += 1

   def genIcon(self):
       """
       Generates/updates the icon for the wave.
       """
       map =  [255]*(ICON_SIZE*ICON_SIZE * 3)
       last = None
       for x in range(ICON_SIZE):
           y = -sample(self.samples, x / ICON_SIZE) * ICON_SIZE / 2+ ICON_SIZE / 2
           y = max(min(y, ICON_SIZE - 1), 0)
           if last:
               self.lineDraw(map, last[0], last[1], x, y)
           last = (x, y)
       self.icon = QtGui.QIcon(QtGui.QPixmap(QtGui.QImage(bytes(map), ICON_SIZE, ICON_SIZE,
QtGui.QImage.Format.Format_RGB888)))
       return

class MyPlotWidget(pg.PlotWidget):
   """Handles the custum drawable plot"""

   def __init__(self, **kwargs):
       """Init"""

       super().__init__(**kwargs)
       self.setMouseEnabled(x=False, y=False)
       self.setXRange(0, 1)
       self.setYRange(-1, 1)
       self.am_drawing = False

       self.line = self.plot([], [], pen='c')
       self.valuesX = np.linspace(0, 1, SAMPLE_POINTS, endpoint=False)
       self.valuesY = [0] * SAMPLE_POINTS
       self.updateGraph()

   def updateGraph(self):
       """Draw the new samples on the graph"""

       self.line.setData(self.valuesX, self.valuesY)
```

```
    def movedPen(self, pos):
        """Called when the pen has moved to a new position. sets the samples between the old a
new position in a line. Updates the graph.

        Parameters:
            pos: object containing the new position coordinate
        """

        newX = round(pos.x() * SAMPLE_POINTS)
        newY = pos.y()

        #determine the starting location and direction/slope to fill samples in
        posX = self.lastX
        posY = self.lastY
        dirX = 1 if newX > posX else -1
        if newX != self.lastX:
            dirY = (newY - self.lastY) / (newX - self.lastX) * dirX
        else:
            dirY = 0

        #sets samples while adjusting and y coordinate according to slope
        while posX != newX + dirX:
            if posX >= 0 and posX < SAMPLE_POINTS:
                self.valuesY[posX] = max(min(posY, 1), -1)
            posX += dirX
            posY += dirY

        self.lastX = newX
        self.lastY = newY
        self.updateGraph()

    def mousePressEvent(self, event: QMouseEvent):
        """
        Handles the mouse press event for the wave drawer.
        """
        if event.button().name == 'LeftButton':
            pos = self.plotItem.vb.mapSceneToView(event.position())

            self.lastX = round(pos.x() * SAMPLE_POINTS)
            self.lastY = pos.y()
            self.am_drawing = True
    def mouseReleaseEvent(self, event: QMouseEvent):
        """
        Handles the mouse release event for the wave drawer.
        """
        if event.button().name == 'LeftButton' and self.am_drawing:
            pos = self.plotItem.vb.mapSceneToView(event.position())
            self.movedPen(pos)
            self.am_drawing = False

    def mouseMoveEvent(self, event: QMouseEvent):
        """
        Handles the mouse move event for the wave drawer.
        """
        if self.am_drawing:
            pos = self.plotItem.vb.mapSceneToView(event.position())
            self.movedPen(pos)

    # -"how much security vunrability do you want?"
    # -"yes"
    def generate_code(self, code):
        """Generates a wave based on given python code.

        Parameters:
            code (str): a peice of python code to determine a new wave
        """

        try:
            tempY = self.valuesY
            #setup execution environment to limit the functionality the user provided code has
access too
            env = {}
            env["locals"]   = None
            env["globals"]  = None
            env["__name__"] = None
            env["__file__"] = None
            env["__builtins__"] = None
            env["math"] = math
            for funct in dir(math):
                if funct[0] != '_':
                    env[funct] = getattr(math, funct)
            #go through all points
            for i in range(SAMPLE_POINTS):
                env["x"] = i / SAMPLE_POINTS
                env["rand"] = random()*2-1
                y = eval(code, env)
                tempY[i] = max(min(y, 1), -1)
        except Exception as e:
            print(e)
            return
        self.valuesY = tempY
        self.updateGraph()

    def generate_preset(self, type):
        """Generates a wave based on a preset type (sine, square, etc). Updates the graph.
        Parameters:
            type (str): the wave type
        """
        res = generateSamples(wavetype=type, numSamples=SAMPLE_POINTS, amplitude=1)
        self.valuesY = res[1]
        self.updateGraph()

    def setSamples(self, samples):
        """
        Sets the list of samples to the given list. Updates the graph.

        Parameters:
            samples (list): The list of samples
        """
        self.valuesY = samples
        self.updateGraph()

class AppWindow(QtWidgets.QWidget):
    """The window for the arbitrary waveform drawer"""

    def __init__(self, chans):
        """
        Initializes the wave drawer window with the given channels.

        Parameters:
            chans (list): The list of channels to update when a wave is modified
        """
        super(AppWindow, self).__init__()

        self.chans = chans
        self.setWindowTitle('Waveform drawer')
        self.currently_loaded_wave = None

        self.listAW = []

        self.pl = MyPlotWidget()


        self.stored_waves = QComboBox(self)
        self.stored_waves.currentIndexChanged.connect(self.dropDownIndexChanged)
        self.stored_waves.view().setIconSize(QtCore.QSize(ICON_SIZE,ICON_SIZE))

        self.dropdown = QComboBox(self)
        self.dropdown.addItem('DC')
        self.dropdown.addItem('Sine')
        self.dropdown.addItem('Triangle')
        self.dropdown.addItem('Sawtooth')
        self.dropdown.addItem('Square')

        self.gen_preset = QPushButton("Generate Preset")
        self.gen_preset.resize(200, 50)
        self.gen_preset.clicked.connect(lambda:
self.pl.generate_preset(self.dropdown.currentText().lower()))

        self.code_input = QLineEdit  ("rand * 0.1 + sin(x*2*pi)*0.5")

        self.gen_code = QPushButton("Generate Code")
        self.gen_code.resize(200, 50)
        self.gen_code.clicked.connect(lambda: self.pl.generate_code(self.code_input.text()))
        self.saveWaveButton = QPushButton("Save Wave")
        self.saveWaveButton.resize(200, 50)
        self.saveWaveButton.clicked.connect(self.saveFunction)

        self.delWave_button = QPushButton("Delete")
        self.delWave_button.resize(200, 50)
        self.delWave_button.clicked.connect(self.delButtonClicked)

        self.newWaveButtpm = QPushButton("Add new")
        self.newWaveButtpm.resize(200, 50)
        self.newWaveButtpm.clicked.connect(lambda: (self.addWave(), self.updateDropDown(),
self.stored_waves.setCurrentIndex(len(self.listAW) - 1)))
```

```
        self.nameEdit = QLineEdit()
        self.nameEdit.editingFinished.connect(self.nameEditDone)

        vert = QtWidgets.QVBoxLayout()
        vert.addWidget(QtWidgets.QLabel("Wave Preset"))
        vert.addWidget(self.dropdown)
        vert.addWidget(self.gen_preset)
        vert.addWidget(self.code_input)
        vert.addWidget(self.gen_code)
        vert.addSpacing(300)
        vert.addWidget(QtWidgets.QLabel("Select"))
        vert.addWidget(self.stored_waves)
        vert.addWidget(self.newWaveButtpm)

        vert.addWidget(QtWidgets.QLabel("Edit name:"))
        vert.addWidget(self.nameEdit)

        vert.addWidget(self.saveWaveButton)
        vert.addWidget(self.delWave_button)

        layout = QtWidgets.QHBoxLayout()
        layout.addWidget(self.pl)
        layout.addLayout(vert)

        self.setLayout(layout)
        self.loadFile()

    def dropDownIndexChanged(self, ind):
        """Called when the selected arb. waveform is changed. Updates the graph and name
textbox based on the new selected wave.

        ind (int): the new index of the selected waveform
        """
        if ind == -1:
            return
        self.nameEdit.setText(self.listAW[ind].name)
        self.pl.setSamples(self.listAW[ind].samples.copy())

    def nameEditDone(self):
        """Called when the name of a wave is changed, updates the dropdown via
updateDropDown()"""
        self.listAW[self.stored_waves.currentIndex()].name = self.nameEdit.text()
        self.updateDropDown()

    def saveFunction(self):
        """
        Saves the current wave to the list and generates an icon for it. Updates the dropdown
via updateDropDown().
        """
        self.listAW[self.stored_waves.currentIndex()].samples = self.pl.valuesY.copy()
        self.listAW[self.stored_waves.currentIndex()].genIcon()
        #for c in self.chans:
        #     c.updateAWList(self.listAW, )
        #self.saveFile()
        self.updateDropDown(cause = "mod", modified = self.stored_waves.currentIndex())

    def updateDropDown(self, cause = "", modified = -1):
        """
        Updates the drop down list of waves. Calls the channel's updateAWList() to propagate
the changes there.

        Parameters:
            cause (str): The cause of the update, can be "mod" for modified, "del" for deleted
or "" for other
            modified (int): The index of the modified wave, -1 if no wave was modified
        """
        ind = self.stored_waves.currentIndex()

        #update the waves
        self.stored_waves.clear()
        i = 0
        for aw in self.listAW:
            self.stored_waves.addItem(aw.name)
            if aw.icon:
                self.stored_waves.setItemIcon(i, aw.icon)
                i += 1
        l = len(self.listAW)
        self.delWave_button.setEnabled(l > 1)

        #reset selected index
        if l > 0 and ind != -1:
            self.stored_waves.setCurrentIndex(min(ind, l - 1))

        #propagate changes to the channel
```

```python
        for c in self.chans:
            c.updateAWList(self.listAW, cause, modified)
        #save changes
        self.saveFile()

    def saveFile(self):
        """Saves the arbitrary wave list to disk"""
        with open('saved.txt', 'w') as f:  # Open in 'w' mode to overwrite the file
            for aw in self.listAW:
                f.write(aw.name + ":")
                for i in range(len(aw.samples)):
                    f.write(str(aw.samples[i]))
                    if i != len(aw.samples) - 1:
                        f.write(",")
                f.write("\n")

    def nameUsed(self, name):
        """
        Checks if the name is already used.

        Parameters:
            name (str): The name to check

        Returns:
            bool: True if the name is already used, False otherwise
        """
        for aw in self.listAW:
            if aw.name == name:
                return True
        return False

    def addWave(self, name = None, arr = None):
        """
        Adds a new wave to the list.

        Parameters:
            name (str): The name of the wave. If None, a default name is used
            arr (list): The list of samples. If None, a default list is used

        Returns:
            None
        """
        if name == None:
            num = 0
            while True:
                s = "Custum " + str(num)
                num += 1
                if not self.nameUsed(s):
                    name = s
                    break
        self.listAW.append(AW(name, arr))

    def loadFile(self):
        """
        Loads the saved waves from the file, if there are no saved waves, a default wave is
added.

        Parameters:
            None

        Returns:
            None
        """
        if os.path.exists("saved.txt"):
            with open("saved.txt", "r") as f:
                for line in f:
                    line = line.strip()
                    name = line[: line.find(':')]
                    strArray = line[line.find(':') + 1:]
                    strArray = strArray.split(",")
                    arr = [float(val) for val in strArray]
                    self.addWave(name, arr)
        if len(self.listAW) == 0:
            self.addWave()
        self.updateDropDown()

    def delButtonClicked(self):
        """
        Deletes the currently selected wave.

        Parameters:
            None

        Returns:
            None
        """
        ind = self.stored_waves.currentIndex()
        del self.listAW[ind]
        self.updateDropDown(cause = "del",  modified = ind)
```

18.3 Wavegen.py

```
import numpy as np
from math import floor

def lerp(a, b, f):
   """
   Performs linear interpolation between two values a and b.

   Parameters:
   a (float): The start value.
   b (float): The end value.
   f (float): The interpolation factor between 0 and 1, where 0 yields `a` and 1 yields `b`.

   Returns:
   float: The interpolated value between `a` and `b`.
   """
   return a * (1 - f) + (b * f)

def resample(samples, newNumSamples):
   """
   Resamples a sequence of data points to a new number of samples. (I don't think this is
used anywhere but might be needed in the future)

   Parameters:
   samples (list of float): The original sequence of sample points.
   newNumSamples (int): The desired number of sample points in the resampled sequence.

   Returns:
   list of float: The resampled sequence of sample points.
   """
   if len(samples) == newNumSamples:
       return samples
   values = []
   for i in range(newNumSamples):
       pos = i / newNumSamples * len(samples)
       ind = floor(pos)
       ind2 = (ind + 1) % len(samples)
       f = pos - ind

       #print(i, pos, ind, ind2, f)
       values += [lerp(samples[ind], samples[ind2], f)]
   return values

def sample(samples, x):
   """Samples a list of points with linear interpolation.

   Given a floating-point index `x`, this function calculates the interpolated
   value using the fractional part of `x` to blend between the nearest sample
   points. `x` = 0 maps to the start of the list and `x` = 1 maps to the end of the list.

   Parameters:
   samples (list of float): The list of sample points to sample.
   x (float): A floating-point coordinate of where to sample.

   Returns:
   float: The interpolated value from the `samples` list at the index `x`.
   """
   x *= len(samples)
   ind = floor(x)
   ind2 = (ind + 1) % len(samples)
   f = x - ind
   return lerp(samples[ind], samples[ind2], f)

def generateSamples(wavetype="sine", numSamples=1024, amplitude=5, arbitrary_waveform=None,
duty=50, phase=0, offset=0,
                   timeRange=1, clamp=None, numT = 1):
   """Generates a waveform of the given type and parameters.

   Parameters:
   wavetype (str): Type of the waveform to generate, defaults to "sine".
   numSamples (int): Number of samples to generate, defaults to 1024.
   amplitude (float): The peak amplitude of the waveform, defaults to 5.
   arbitrary_waveform (list of float): A list of samples for the user-defined arbitrary
waveforms, defaults to None.
   duty (int): Duty cycle for square waves, defaults to 50 percent.
   phase (float): Phase shift for the waveform, defaults to 0.
```

```
   timeRange (float): The time range over which to generate the waveform, defaults to 1.
   clamp (function or None): An optional function to clamp values, defaults to None (no
clamping is done).
   numT (int): Number of periods to generate, defaults to 1.

   Returns:
       Tuple of (time, voltage)
   """
   t = np.linspace(0, numT, numSamples, endpoint=False)
   tt = t
   phase = float(phase)
   t = np.mod(t + phase, 1)

   if wavetype == 'arbitrary':
       y = np.zeros(numSamples)
       for i in range(numSamples):
           y[i] = sample(arbitrary_waveform, t[i])
   else:
       if wavetype == 'sine':
           y = np.sin(2 * np.pi * t)
       elif wavetype == "triangle":
           t = np.mod(t + 0.25, 1)
           y = (np.mod(t * 2, 1) * -(np.floor(t * 2) * 2 - 1) + np.floor(t * 2)) * 2 - 1
       elif wavetype == "square":
           y = np.ones(numSamples)
           y[t >= float(duty) / 100] = -1
       elif wavetype == "sawtooth":
           t = np.mod(t + 0.5, 1)
           y = np.mod(t * 2, 2) - 1
       elif wavetype == "dc":
           y = np.zeros(numSamples)
       else:
           print("bad wavetype")

   tt = tt * timeRange
   y = y * amplitude + offset
   if clamp:
       np.clip(y, clamp[0], clamp[1], y)
   return (tt, y)


def samplesToBytes(samples):
   """converts a list of samples to bytes. The samples will be stored in 16 bit (2 byte)
little endian. The bytearray's length will be padded to the nearest multiple of 64.

   Parameters:
       samples (list of float): samples to convert to bytes

   Returns:
       Bytearray(?) representing the samples.

   """
   ns = len(samples)
   if ns % 64 != 0:
       add = 64 - (ns % 64)
       samples = np.pad(samples, (0,add), 'constant', constant_values=(0,))

   return samples.astype(dtype = "<u2", casting='unsafe').tobytes()
```

18.4 Input Field

```
from PyQt6.QtWidgets import QLineEdit

def findBestPrefix(val, prefixes):
    """ Finds the best prefix for a given value.

    Parameters:
    val (float): The value to find the best prefix for.
    prefixes (dict of str:int): dictionary holding prefixes (m for milli, k for kilo, etc)
mapped to their value.

    Returns:
    (float, str): The value with the best prefix, and the prefix itself."""
    for p in prefixes:
        n_val = val / prefixes[p]
        if n_val >= 1 and n_val < 1e3:
            return (n_val, p)
    return (val, "")

def endsWithLower(s, endsWithStr):
    """ Checks if a string ends with another string, ignoring case.

    Parameters:
    s (str): the string to test
```

```
   endsWithStr (str):  the ending

   Returns: boolean
   """
   return s[-len(endsWithStr):].lower() == endsWithStr.lower()

def parseStringToVal(str_in, prefixes, expected_unit):
   """ Parses a string into a value.

   Parameters:
   str_in (str): The string to parse.
   prefixes (dict of str:int): dictionary holding prefixes (m for milli, k for kilo, etc)
mapped to their value.
   expected_unit (str): The expected unit of the value.

   Returns:
   float: The value parsed from the string, or None if the string could not be parsed.
   """
   str_in = str_in.replace(" ", "")

   if endsWithLower(str_in, expected_unit):
       str_in = str_in[:-len(expected_unit)]

   mag = 1
   for p in prefixes:
       if endsWithLower(str_in, p):
           str_in = str_in[:-len(p)]
           mag = prefixes[p]

   try:
       val = float(str_in)
   except Exception as e:
       return None

   return val * mag

def clamp(val, minVal, maxVal):
   """Clamps a value to a given range.

   Parameters:
   val (float): The value to clamp.
   minVal (float): The minimum value of the range.
   maxVal (float): The maximum value of the range.

   Returns:
   float: The clamped value.
   """
   return float(min(maxVal, max(minVal, val)))

class Input(QLineEdit):
   """Handles a text input box with automatic range clamping, unit display, scrolling"""

   def editFin(self):
       """Function called when the user has finished editing the text box. If invalid value
is entered, the text box is reset to the previous value."""
       val = parseStringToVal(self.text(), self.prefixes, self.def_unit)
       if val is None:
           self.undo()
       else:
           self.setVal(val)

   def keyPressEvent(self, event):
       """Event handler for key input. If the up or down arrow keys are pressed, the value is
incremented or decremented by 1, respectively.

       Parameters:
       event (QKeyEvent): The key event that was triggered.
       """
       if event.key() == 16777235:
           self.setVal(self.value + 1)
       elif event.key() == 16777237:
           self.setVal(self.value - 1)
       else:
           QLineEdit.keyPressEvent(self, event)

   def wheelEvent(self, event):
       """Event handler for mouse scrolling. The value is incremented or decremented based on
the scroll direction.

       Parameters:
       event (QKeyEvent): The key event that was triggered.
       """
       self.setVal(self.value + event.angleDelta().y() / 120)
       event.accept()
```

```python
    def setVal(self, val, runCallback = True):
        """ Sets the value of the text box.

        Parameters:
        val (float): The value to set the text box to.
        runCallback (bool): Whether or not to run the callback function after setting the value, defaults to True. This can be used to update a textbox when a different textbox is updated (for example have the frequency update when period is changed and viceversa), without causing a infinite loop of updates. But currently this isn't used.
        """
        val = clamp(val, self.range[0], self.range[1])
        self.value = val
        val, prefix = findBestPrefix(val, self.prefixes)
        self.setText(f"{val} {prefix}{self.def_unit}")
        if runCallback:
            self.callback_update()

    def __init__(self, callback_update, range, init_val, def_unit, prefixes = [], *args, **kwargs):
        """Initializes the object

        Parameters:
        callback_update (funct): function to call when the value is changed
        range (tuple (float, float)): a tuple holding the min and max values the input value can be.
        init_val (float): the initial value of the input box
        def_unit (str): the unit symbol (such as "v" or "hz")
        prefixes (dict of str:float): dictionary holding prefixes (m for milli, k for kilo, etc) mapped to their value.
        *args: extra args
        **kwargs: extra args
        """
        QLineEdit.__init__(self, *args, **kwargs)
        self.editingFinished.connect(self.editFin)
        self.callback_update = callback_update
        self.range = range
        self.prefixes = prefixes
        self.def_unit = def_unit
        self.setVal(init_val)
```

18.5 Connection.py

```
import serial
import serial.tools.list_ports
import threading
import math
from wavegen import *
from struct import pack
import os
from queue import Queue
import time

class Connection:
   """Handles the connection to the AWG device"""

   def sendHandShakePacket(self):
       """ Sends a handshake packet to the device.

       This function should be called when the device is first connected to and during
keep-alive."""
       if self.status == "disconnected":
           return
       bytes = pack("B4B59x", 0, ord('I'), ord('N'), ord('I'), ord('T'))
       assert len(bytes) == 64
       self.sendQ.put(bytes)

   def read_disconnect(self, msg):
       """ Disconnects the device and emits a disconnected signal to the main UI.

       Parameters:
       msg (str): The reason for the disconnect.
       """
       if self.status != "disconnected":
           self.status = "disconnected"
           self.statusCallback.emit("disconnected", msg)
           self.ser.close()

   def read_funct(self):
       """ The function that runs the read thread. This function reads data from the serial
port. Detects acknowledgments and disconnects due to timeout, emitted updates to the main UI
as needed."""
       timeouts = 0
       while self.status != "disconnected":
           try:
               buff = self.ser.read(64)
           except:
               buff = None
               self.read_disconnect("Connection Disconnected")
               break

           if len(buff) == 0: #timeout
               #send keep alive packets?
               if timeouts == 0:
                   timeouts = 1
                   self.sendHandShakePacket()
               else:
                   self.read_disconnect("timeout")
               pass
           else:
               if(buff[0:9] == bytes("\0STMAWG23", "ascii")): #ack packet
                   timeouts = 0
                   if self.status == "connecting":
                       self.statusCallback.emit("connected", None)
                   self.status = "connected"
               else:   #bad reply
                   self.status = "disconnected"
                   self.sendQ.put(None)
                   self.statusCallback.emit("disconnected", "Bad packet")
       #causes the write thread to wake up so that it can exit.
       self.sendQ.put(None)

   def write_funct(self):
       """ The function that runs the write thread. This function writes packets from the
sendQ to the serial port."""
       while self.status != "disconnected":
           packet = self.sendQ.get()
           if packet:
               try:
                   self.ser.write(packet)
               except:
                   pass

   def close(self):
       """ Disconnects the device and indirectly shutdowns the read/write threads."""
       if self.status != "disconnected":
           self.status = "disconnected"
           self.ser.close()

#    def up64(self, bytes):
#        while(len(bytes) % 64 != 0):
#            bytes += [0]
#        return bytes

   def getSkips(self, freq, numSamples, fclk):
       """Calculates the sample period in clock cycles for a given frequency, sample number,
and MCU clock speed"""
       return fclk / (freq * numSamples)

   def calc_val(self, freq):
       """Dynamically picks the best numSamples value, and corresponding values for the
ARR/PSC registers."""
       fclk = 72e6
       skipGoal = 25 #minimum sample period target
       max_samples = 1024*4
       numSamples = max_samples
       #get close to the target sample period without going under
       while (skips := self.getSkips(freq, numSamples, fclk)) < skipGoal:
           numSamples /= 2

       numSamples = int(numSamples)

       #calculate PSC and ARR from the sample period (skips)
       PSC = 1
       while (ARR := skips / PSC) > 2**16:
           PSC += 1
       PSC -= 1
       ARR = round(ARR - 1)

       return numSamples, ARR, PSC



   def sendWave(self, chan, freq = 1e3, wave_type = "sin", amplitude = 5, offset = 0,
arbitrary_waveform = None, duty = 50, phase = 0, numPeriods = 1):
       """ Sends a waveform to the device.

       Parameters:
       chan (int): The channel to send the waveform to.
       freq (float): The frequency of the waveform.
       wave_type (str): The type of waveform to send.
       amplitude (float): The amplitude of the waveform.
       offset (float): The offset of the waveform.
       arbitrary_waveform (list of float): A user-defined function for arbitrary waveforms,
defaults to None.
       duty (int): Duty cycle for square waves, defaults to 50 percent.
       phase (float): Phase shift for the waveform, defaults to 0.
       numPeriods (int): Number of periods to generate, defaults to 1.
       """
       if self.status == "disconnected":
           return

       #move constants to init or something
       fclk = 72e6
       dac_bits = 12
       pwm_bits = 12
       offset_amp = 5
       PWM_ARR = 2**pwm_bits - 1
       gain_amp = [5, 0.5]

       #determines to use high or low gain
       if wave_type == "dc" or offset > 5:
           gain = 0
       else:
           gain = 0 if abs(amplitude) > 0.5 else 1

       #calculate offset CCR value for offset
       offset_pwm = max(min(offset, 5), -5)
       CCR_offset = max(min(math.floor((-offset_pwm + offset_amp) / (offset_amp * 2) *
PWM_ARR), PWM_ARR), 0)
       offset_dac = offset - offset_pwm

       #determines numSamples, and ARR/PSC values
       if wave_type == "dc":
           numSamples, ARR, PSC = (2, int(2**15), 0)
       else:
           numSamples, ARR, PSC = self.calc_val(freq)
       #skips_act = (PSC+1)*(ARR+1)

       #determines phase
       #this code is complicated because there are two waves of setting phase:
```

```
        #   1) shifting the samples, which has lower resolution but a full 360 deg range
        #   2) shifting the clock cycle on which the output starts, which has a resolution of
13.9 ns (at 72mhz)
        #we want to do both
        #PSC not being 0 makes things complicated and the functionality is not well tested for
slow waves.
        phase_clocks = numSamples * (ARR + 1) * phase
        phase_samples = phase_clocks / (ARR + 1) / numSamples
        phase_arr = 0
        #ALLIGNINS CLOCK SIGNAL PHASES TDOWN TO THE CLOCK CYCLE, SEEMS TO BE RANDOM BETWEEN
DEVICES AND NEEDS TO BE ADJUSTED MANUALY
        if chan == 1:
            phase_arr += 6 // (PSC + 1)
        phase_arr += int(phase_clocks / (PSC + 1))
        phase_arr = phase_arr % (ARR + 1)

        #generate the samples
        dac_scale = (2**dac_bits) / 2
        samples = generateSamples(wavetype = wave_type, numSamples = numSamples, amplitude =
amplitude / gain_amp[gain] * dac_scale, arbitrary_waveform = arbitrary_waveform, duty = duty,
phase = phase_samples, offset = dac_scale + offset_dac / gain_amp[gain] * dac_scale, clamp =
[0, 2**dac_bits - 1], numT = numPeriods)
        samples = samples[1]

        #generate the packet
        bytes = pack("<BBBHHHHH51x", 1, chan, gain, PSC, ARR, CCR_offset, numSamples,
phase_arr)
        sample_bytes = samplesToBytes(samples)
        assert len(bytes) % 64 == 0
        bytes += sample_bytes
        self.sendQ.put(bytes)

    def tryConnect(self):
        """ Attempts to connect to the device."""
        if self.status != "disconnected":
            return

        #try to find the port with the correct vid/pid
        portName = None
        ports = list(serial.tools.list_ports.comports())
        for port in ports:
            if(port.vid == 1155 and port.pid == 22336):
                portName = port.name
                break

        if not portName:
            self.statusCallback.emit("disconnected", "device not found")
            return

        #add the "/dev/" for unix based systems
        if os.name == "posix":
            portName = "/dev/" + portName

        try:
            self.ser = serial.Serial(portName, 500000, timeout = 5) #BAUD rate Doesnt actually
matter
        except Exception as e:
            print(e)
            self.statusCallback.emit("disconnected", "unable to open port")
            return

        self.status = "connecting"

        self.sendQ = Queue()
        self.sendHandShakePacket()

        self.write_thread = threading.Thread(target=self.write_funct, args=())
        self.write_thread.start()
        #
        self.read_thread = threading.Thread(target=self.read_funct, args=())
        self.read_thread.start()

    def __init__(self, statusCallback):
        """ Initializes the connection object

        Parameters:
            statusCallback (pySignal): signal to use to send updates to the main UI
        """
        self.status = "disconnected"
        self.statusCallback = statusCallback
```

18.6 Channel.py

```
import subprocess
import math
import numpy as np
import pyqtgraph as pg
from pyqtgraph.Qt import QtCore, QtWidgets
from PyQt6.QtCore import pyqtSignal
from PyQt6.QtWidgets import (
   QApplication, QWidget, QLineEdit, QPushButton, QVBoxLayout, QMessageBox, QComboBox
)
from PyQt6 import QtGui
from wavegen import generateSamples
from connection import Connection
from input_field import Input
from wave_drawer import ICON_SIZE

class WaveSettings():
   """Stores a list of settings about the waveform of a channel"""

   def __init__(self, type, freq, amp, offset =0 , duty = 50, phase = 0, arb = None):
       """ Initializes the WaveSettings class.

       type (str): Type of the waveform to generate.
       freq (float): Frequency of wave.
       amp (float): Amplitude of the wave.
       offset (float): DC offset of the wave.
       duty (float): duty cycle of the wave (if wave is square).
       phase (float): phase of the wave
       arb (list of float): A list of samples for the user-defined arbitrary waveforms.
       """
       self.type = type
       self.freq = freq
       self.amp = amp
       self.offset = offset
       self.duty = duty
       self.phase = phase
       self.arb = arb

class Channel:
   """Handles the UI and logic for a single channel"""

   def update_dropdown(self):
       """Called when the wavetype dropdown is triggered. enables/disables input boxes as
needed by wavetype. Calls generate_waveform()"""
```

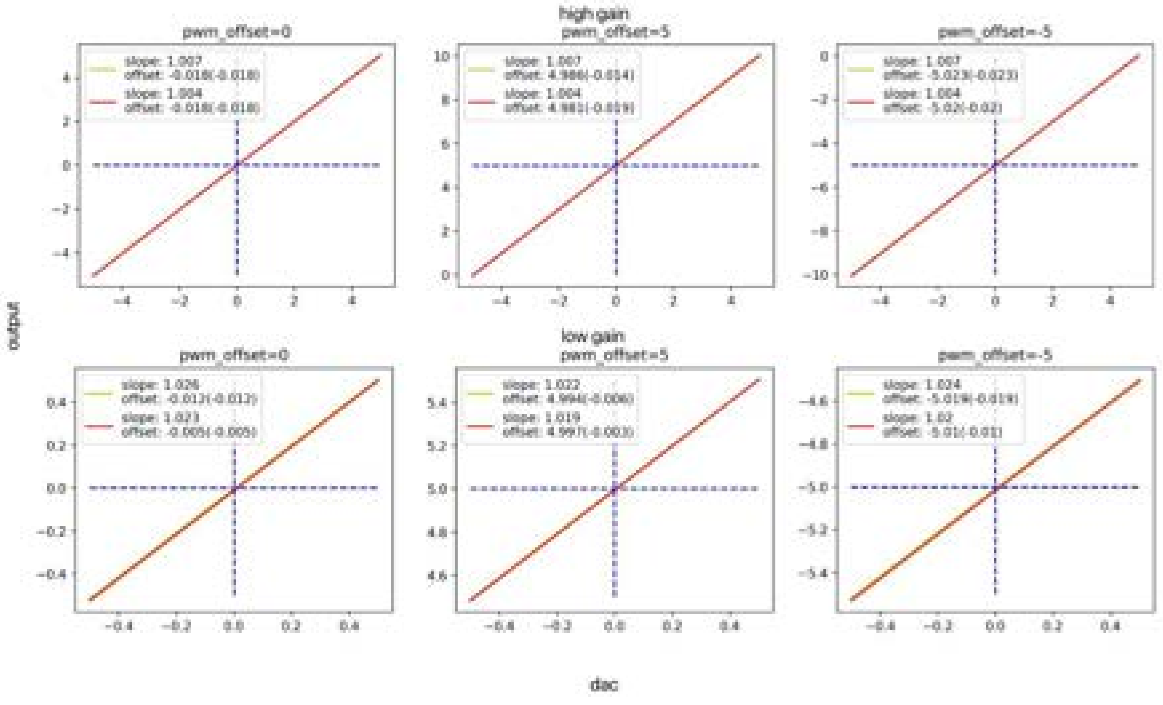
```
        self.waveform_type = self.dropdown.currentText().lower()
        if (self.waveform_type == 'square'):
            self.freqInput.setEnabled(True)
            self.ampInput.setEnabled(True)
            self.offsetInput.setEnabled(True)
            self.dutyInput.setEnabled(True)
            self.phaseInput.setEnabled(True)
        elif (self.waveform_type == 'dc'):
            self.freqInput.setEnabled(False)
            self.ampInput.setEnabled(False)
            self.offsetInput.setEnabled(True)
            self.dutyInput.setEnabled(False)
            self.phaseInput.setEnabled(False)
        else:
            self.freqInput.setEnabled(True)
            self.ampInput.setEnabled(True)
            self.offsetInput.setEnabled(True)
            self.dutyInput.setEnabled(False)
            self.phaseInput.setEnabled(True)
        self.generate_waveform()

    def updateAWList(self, AWlist, cause, modified = -1):
        """ Updates the list of arbitrary waveforms in the GUI. Calls generate_waveform as
needed.

        Parameters:
        AWlist (list): The list of arbitrary waveforms.
        cause (str): The cause of the update, either "mod", "del", or "".
        modified (int): The index of the modified waveform, defaults to -1.
        """
        last_ind = self.dropdownArb.currentIndex()

        #updates the AW list, updates dropdown of custom waves
        self.listAW = AWlist
        self.dropdownArb.clear()
        ind = 0
        for aw in self.listAW:
            self.dropdownArb.addItem(aw.name)
            if aw.icon:
                self.dropdownArb.setItemIcon(ind, aw.icon)
            ind+=1
```

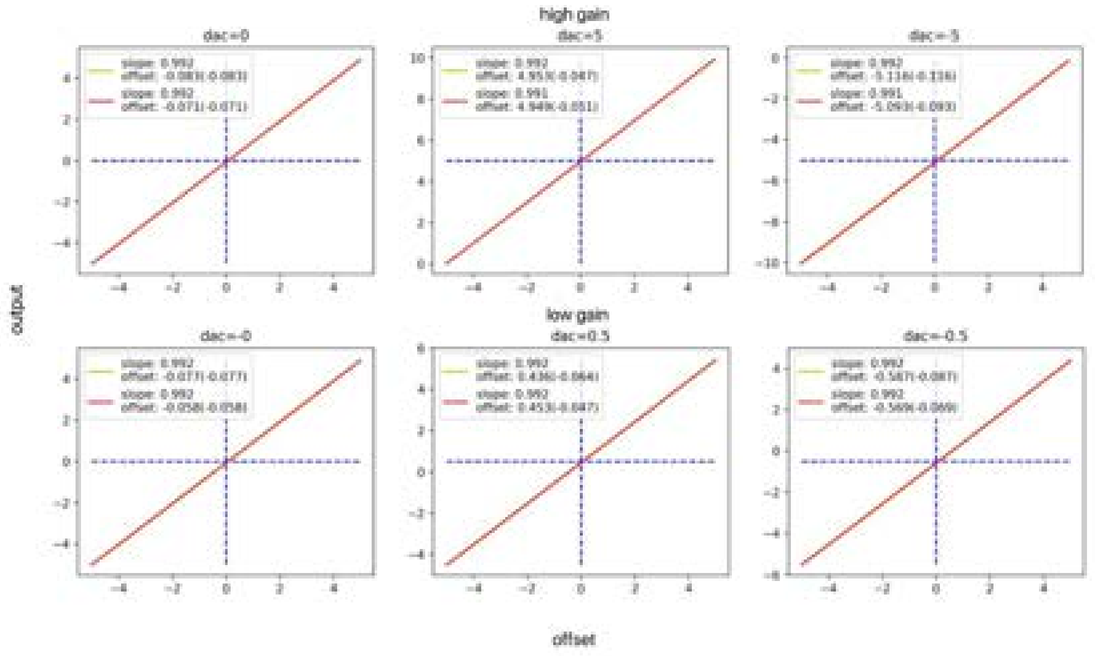
```
       #some of the last_ind!= -1 checks might not be needed here but not enough time to
check if removing them breaks anything

       # if a arbitrary wave was modified, and that wave is currently selected, we need to
update if the wavetype is arbitrary
       if cause == "mod" and last_ind != -1 and last_ind == modified:
           self.dropdownArb.setCurrentIndex(last_ind)
           if self.waveform_type == "arbitrary":
               self.generate_waveform()
       elif cause == "del" and last_ind != -1:
           #currently selected arbitrary wave was deleted, we need to change index and update
if the wavetype is arbitrary
           if last_ind == modified:
               self.dropdownArb.setCurrentIndex(max(last_ind - 1, 0))
               if self.waveform_type == "arbitrary":
                   self.generate_waveform()
           #decrement dropdown selection index if the deleted wave was prior to the the
selected wave
           elif last_ind > modified:
               self.dropdownArb.setCurrentIndex(last_ind - 1)
       #reset the dropdown to the selection that was before the update
       elif last_ind != -1:
           self.dropdownArb.setCurrentIndex(last_ind)

   def setRunningStatus(self, status):
       """Sets the running status, updates the UI of the run/stop button based on the new
status. Calls generate_waveform()

       Parameters:
       status (bool): The new running status.
       """
       self.running = status
       if self.running:
           self.run_stop.setText("Stop")
           self.run_stop.setIcon(self.icons["stop"])
           #self.run_stop.setStyleSheet("background-color : lightblue")
       else:
           self.run_stop.setText("Run")
           self.run_stop.setIcon(self.icons["run"])
           #self.run_stop.setStyleSheet("background-color : lightgrey")
       self.generate_waveform()
   def generate_waveform(self):
       """Updates the waveform on the graph and calls updateWave() in main.py (via
self.updateWave) so that the new wave can be sent to the AWG device. Called anytime a
relevant setting is changed."""
       #generate_waveform() will be called spuriously while initialization is happening, we
want to ignore these calls until the initialization is done.
       if not self.initDone:
           return

       #get arbitrary waveform samples from the AW list
       tr = 1 / self.freqInput.value
       if self.dropdownArb.currentIndex() != -1:
           arbitrary_waveform = self.listAW[self.dropdownArb.currentIndex()].samples
       else:
           arbitrary_waveform = None

       #generate the samples for the graph
       samples = generateSamples(self.waveform_type, 1000 if self.waveform_type !=
"arbitrary" else 4096, self.ampInput.value, arbitrary_waveform, self.dutyInput.value,
self.phaseInput.value / 360, offset = self.offsetInput.value, timeRange = tr, clamp = [-10,
10])

       #graphs the samples
       self.plot_data.setData(samples[0], samples[1])
       self.plot_widget.setXRange(0, tr)
       self.plot_widget.setLabel('left', text='', units='V')
       self.plot_widget.setLabel('bottom', text='', units= 's')

       #updates the amplitude/offset lines
       self.guide_lines[0].setData([-tr, tr * 2], [self.offsetInput.value,
self.offsetInput.value])
       self.guide_lines[1].setData([-tr, tr * 2], [self.offsetInput.value +
self.ampInput.value, self.offsetInput.value + self.ampInput.value])
       self.guide_lines[2].setData([-tr, tr * 2], [self.offsetInput.value -
self.ampInput.value, self.offsetInput.value - self.ampInput.value])
       self.guide_lines[1].setVisible(self.waveform_type != "dc")
       self.guide_lines[2].setVisible(self.waveform_type != "dc")

       #updates the wave settings. The AWG device has no way to turn off output, so "off" is
just a DC wave.
       if self.running:
           self.waveSettings = WaveSettings(type = self.waveform_type, freq =
self.freqInput.value, amp = self.ampInput.value, offset = self.offsetInput.value, duty =
self.dutyInput.value, phase = self.phaseInput.value / 360, arb = arbitrary_waveform)
       else:
           self.waveSettings = WaveSettings(type = "dc", freq = 1e3, amp = 5)

       #we don't want to call main.py's updateWave() until all channels are initialized.
allowUpdates is used to accomplish this
       if self.allowUpdates:
           self.updateWave(self.chan_num)

       #update the running icon UI
       if self.running:
           self.on_off_label.setText("ON")
           self.on_off_label.setColor((0, 255, 0))
       else:
           self.on_off_label.setText("OFF")
           self.on_off_label.setColor((255, 0, 0))


   def enableUpdates(self):
       """We don't want channel to call main.py's updateWave() until all channels are
initialized. This is called when the initialization is done."""
       self.allowUpdates = True
       self.updateWave(self.chan_num)

   def __init__(self, chan_num, grid_layout, icons, updateWave):
       """Initializes the channel.

       Parameters:
           chan_num (int): the number of the channel (0 or 1)
           grid_layout (): the grid layout to add UI elements, TODO, replace the grid with a
hierarchy of vertical/horizontal layouts
           icons (dict of str: qticon): dict mapping icon names to the icon
           updateWave (funct): function to call when we want to send the wave to the AWG
device (this is just main.py's updateWave())
       """
       self.icons = icons
       self.chan_num = chan_num
       self.updateWave = updateWave

       if chan_num == 0:
           GUI_OFFSET = 0
       elif chan_num == 1:
           GUI_OFFSET = 9

       self.initDone = False
       self.allowUpdates = False
       #self.arbitrary_waveform = None
       self.waveform_type = "sine"

       self.plot_widget = pg.PlotWidget()
       self.plot_widget.setMouseEnabled(x=False, y=False)
       self.plot_widget.setLabel('left', text='', units='V')
       self.plot_widget.setLabel('bottom', text='', units='s')
       self.plot_widget.setYRange(-10, 10)
       self.plot_widget.hideButtons()

       #initialize the offset/amplitude lines
       self.guide_lines = []
       for i in range(3):
           if chan_num == 0:
               self.guide_lines.append(self.plot_widget.plot(pen=(255, 255, 0, 96)))
               # self.guide_lines.append(self.plot_widget.plot(pen=pg.mkPen(color='y',
dash=[5, 5])))
           elif chan_num == 1:
               self.guide_lines.append(self.plot_widget.plot(pen=(0, 255, 255, 96)))
               # self.guide_lines.append(self.plot_widget2.plot(pen=pg.mkPen(color='c',
dash=[5, 5])))

       COLORS = ['y', 'c']
       self.plot_data = self.plot_widget.plot(pen=COLORS[chan_num])

       self.on_off_label = pg.TextItem()
       self.on_off_label.setPos(0, 10)
       self.plot_widget.addItem(self.on_off_label)

       self.clabel = QtWidgets.QLabel(f'Channel {chan_num + 1}')

       prefixes_v = {"m": 1e-3}
       prefixes_f = {"K": 1e3, "M": 1e6}

       self.freq_label = QtWidgets.QLabel('Frequency (Hz):')
       self.freqInput = Input(self.generate_waveform, [1, 250e3], float(1000), "hz",
prefixes_f)

       self.amp_label = QtWidgets.QLabel('Amplitude:')
       self.ampInput = Input(self.generate_waveform, [-5, 5], float(5), "v", prefixes_v)

       self.offset_label = QtWidgets.QLabel('Offset voltage:')
       self.offsetInput = Input(self.generate_waveform, [-10, 10], float(0), "v", prefixes_v)

       self.DCLabel = QtWidgets.QLabel("Duty Cycle:")
       self.dutyInput = Input(self.generate_waveform, [0, 100], float(50), "%", [])

       self.phaseLabel = QtWidgets.QLabel("Phase (Deg):")
       self.phaseInput = Input(self.generate_waveform, [0, 360], float(0), "deg", [])

       self.run_stop = QtWidgets.QPushButton()
       self.run_stop.clicked.connect(lambda: self.setRunningStatus(not self.running))

       self.wavetypeLabel = QtWidgets.QLabel("Wave type")
       self.dropdown = QComboBox()
       self.dropdown.addItem('Sine')
       self.dropdown.addItem('Triangle')
       self.dropdown.addItem('Sawtooth')
       self.dropdown.addItem('Square')
       self.dropdown.addItem('Arbitrary')
       self.dropdown.addItem('DC')
       self.dropdown.setItemIcon(0, icons["sine"])
       self.dropdown.setItemIcon(1, icons["tri"])
       self.dropdown.setItemIcon(2, icons["saw"])
       self.dropdown.setItemIcon(3, icons["square"])
       self.dropdown.setItemIcon(4, icons["arb"])
       self.dropdown.setItemIcon(5, icons["dc"])

       self.dropdown.activated.connect(self.update_dropdown)

       self.arbwaveLabel = QtWidgets.QLabel("Arbitrary Wave:")
       self.dropdownArb = QComboBox()
       self.dropdownArb.activated.connect(self.generate_waveform)
       self.dropdownArb.view().setIconSize(QtCore.QSize(ICON_SIZE,ICON_SIZE))

       grid_layout.addWidget(self.plot_widget, GUI_OFFSET + 1, 0, 9, 5)
       grid_layout.addWidget(self.clabel, GUI_OFFSET + 1, 5, 1, 1)
```

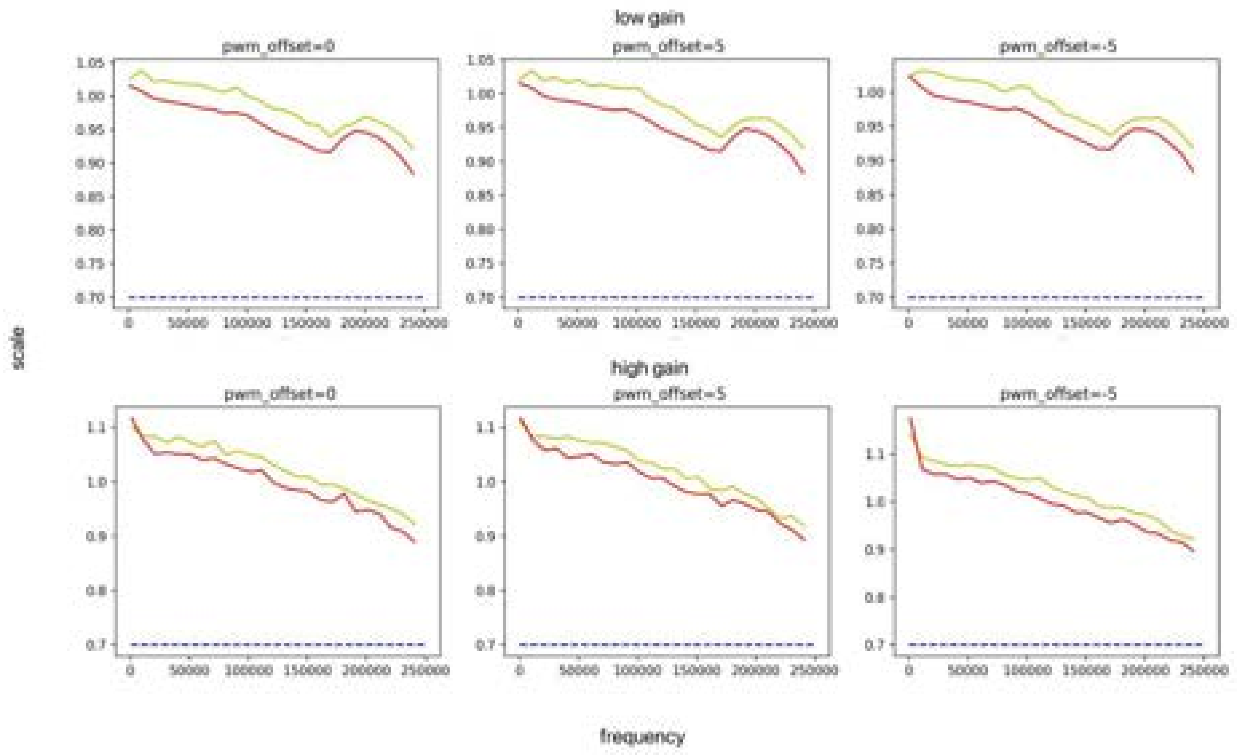
```
    grid_layout.addWidget(self.freq_label, GUI_OFFSET + 2, 5, 1, 1)
    grid_layout.addWidget(self.freqInput, GUI_OFFSET + 2, 6, 1, 1)

    grid_layout.addWidget(self.amp_label, GUI_OFFSET + 3, 5, 1, 1)
    grid_layout.addWidget(self.ampInput, GUI_OFFSET + 3, 6, 1, 1)

    grid_layout.addWidget(self.offset_label, GUI_OFFSET + 4, 5, 1, 1)
    grid_layout.addWidget(self.offsetInput, GUI_OFFSET + 4, 6, 1, 1)

    grid_layout.addWidget(self.wavetypeLabel, GUI_OFFSET + 7, 5, 1, 1)
    grid_layout.addWidget(self.dropdown, GUI_OFFSET + 7, 6, 1, 1)
    grid_layout.addWidget(self.arbwaveLabel, GUI_OFFSET + 8, 5, 1, 1)
    grid_layout.addWidget(self.dropdownArb, GUI_OFFSET + 8, 6, 1, 1)

    grid_layout.addWidget(self.DCLabel, GUI_OFFSET + 5, 5, 1, 1)
    grid_layout.addWidget(self.dutyInput, GUI_OFFSET + 5, 6, 1, 1)
    grid_layout.addWidget(self.phaseLabel, GUI_OFFSET + 6, 5, 1, 1)
    grid_layout.addWidget(self.phaseInput, GUI_OFFSET + 6, 6, 1, 1)

    grid_layout.addWidget(self.run_stop, GUI_OFFSET + 9, 5, 1, 2)

    self.update_dropdown()
    self.setRunningStatus(False)
    self.initDone = True
    self.generate_waveform()
```

19. Appendix F: Python Code Coverage Testing

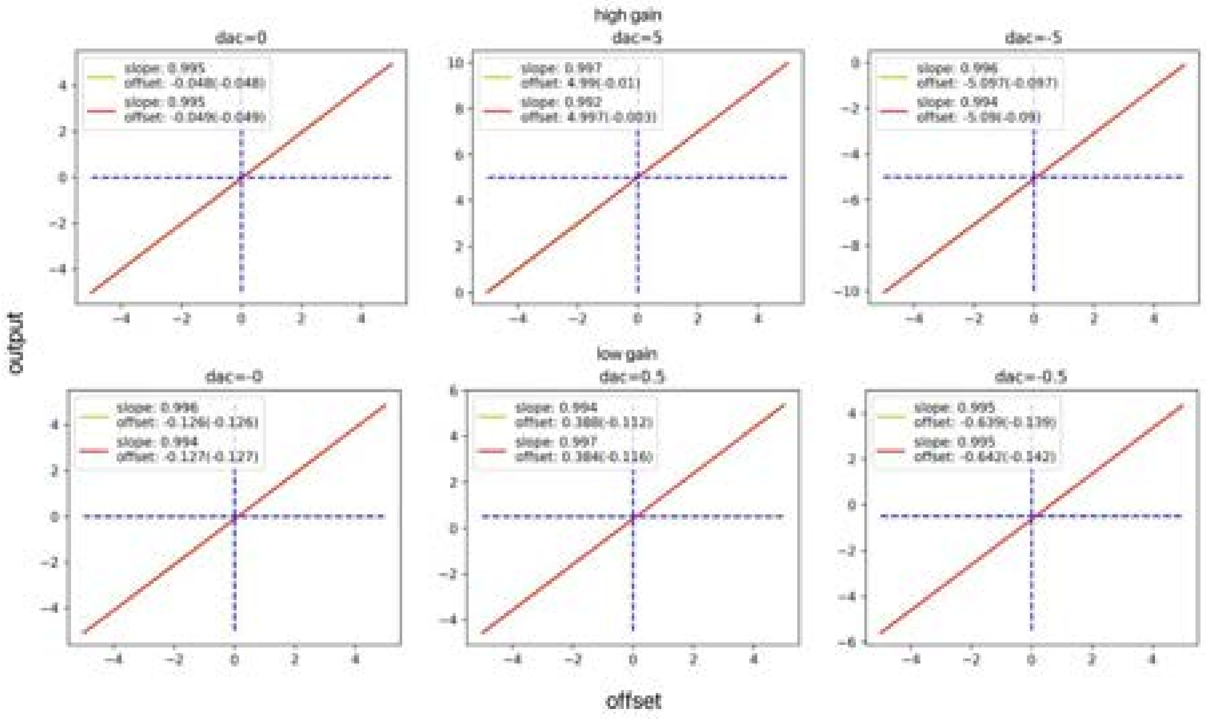
19.1 Board 1 testing result

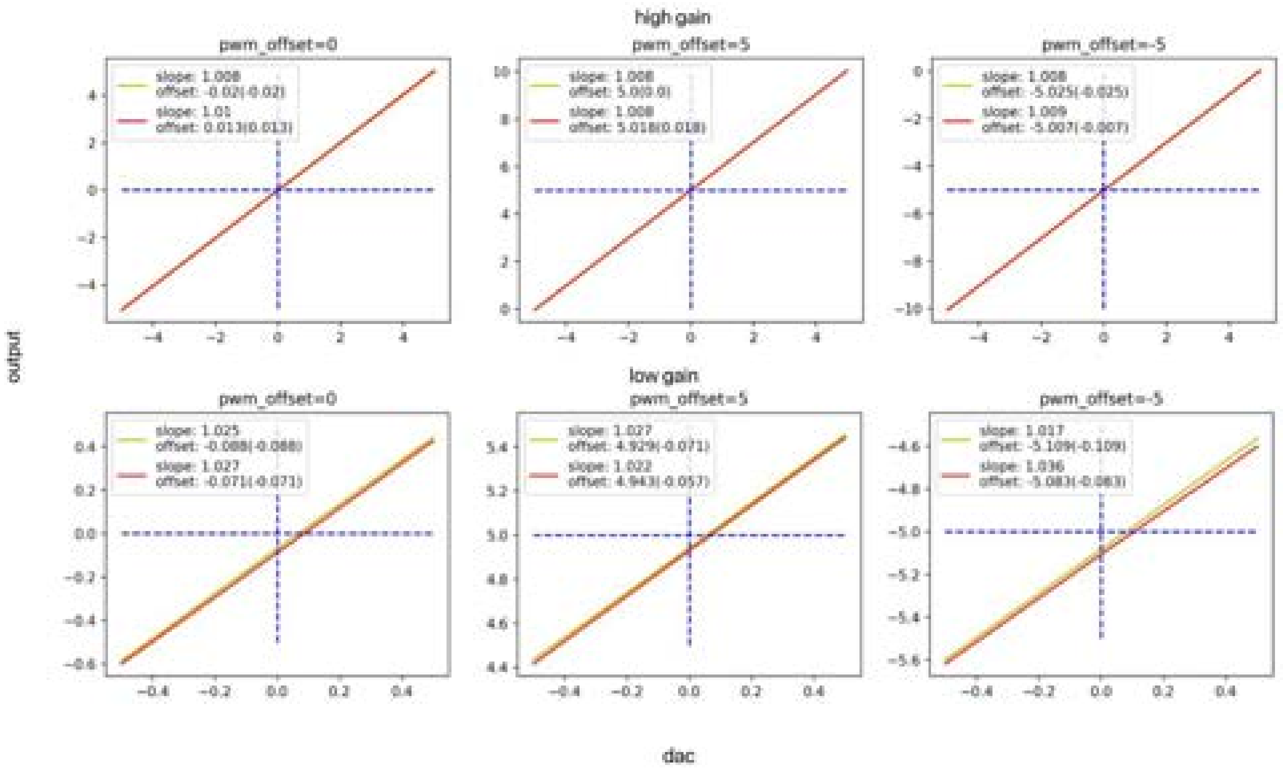


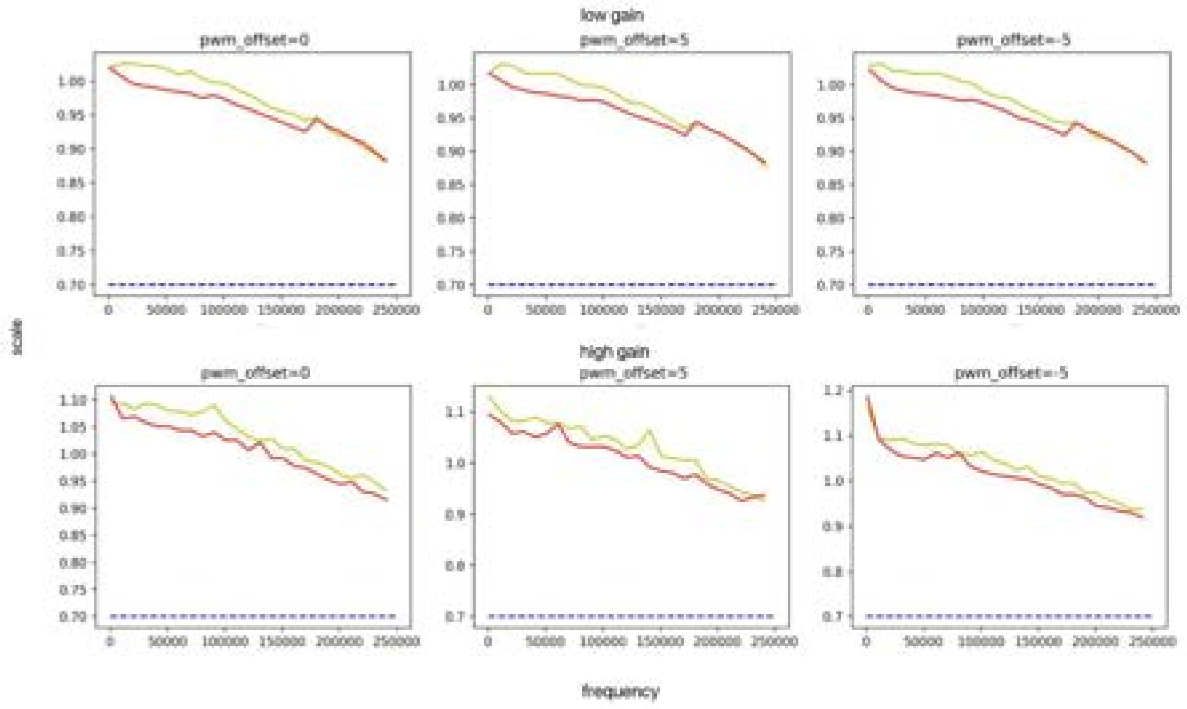




19.2 Board 2 testing result




high gain
pwm_offset=0
slope: 1.008
offset: -0.02(-0.02)
slope: 1.01
offset: 0.013(0.013)
pwm_offset=5
slope: 1.008
offset: 5.0(0.0)
slope: 1.008
offset: 5.018(0.018)
pwm_offset=-5
slope: 1.008
offset: -5.025(-0.025)
slope: 1.009
offset: -5.007(-0.007)
low gain
pwm_offset=0
slope: 1.025
offset: -0.088(-0.088)
slope: 1.027
offset: -0.071(-0.071)
pwm_offset=5
slope: 1.027
offset: 4.929(-0.071)
slope: 1.022
offset: 4.943(-0.057)
pwm_offset=-5
slope: 1.017
offset: -5.109(-0.109)
slope: 1.036
offset: -5.083(-0.083)
output
dac


low gain
pwm_offset=0
pwm_offset=5
pwm_offset=-5
high gain
pwm_offset=0
pwm_offset=5
pwm_offset=-5
scale
frequency