Blue2

Subsystem & System Reports

Revision 1
4/30/2022

Table of Contents

List of Figures

Blue2

Jon Flores
Aaron Gavin

PCB/PIC32 Subsystem Report

Revision 1
4/30/2022

Subsystem Introduction

The PCB/PIC subsystem report covers the overall design of the Blue2 device and the PIC32 microcontrollers functionality on our device.

Subsystem Details

Figure 1: Blue2 Altium PCB Layout

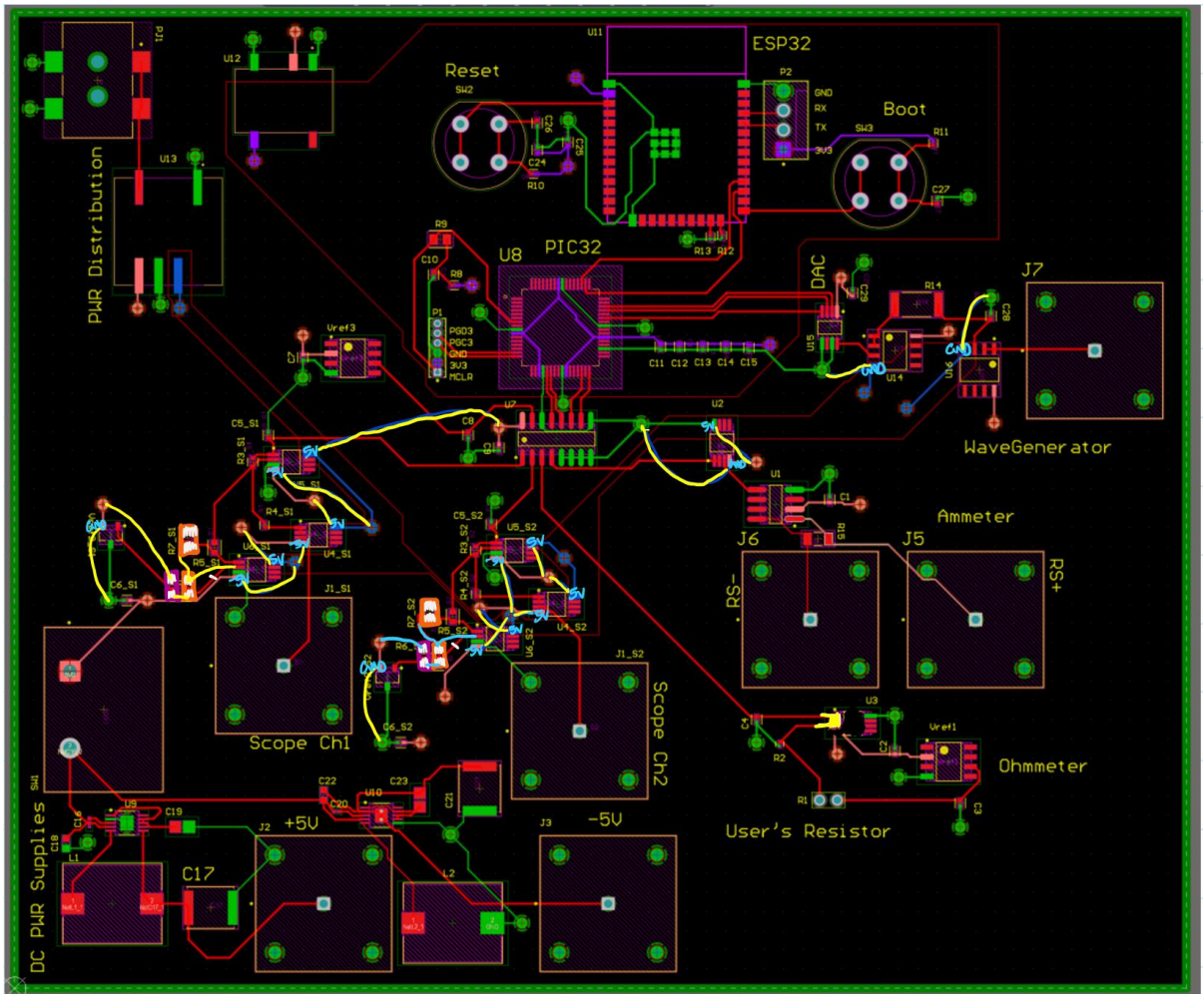


Figure 1 shows the Blue2 device's final Altium PCB layout with the drawn-on design changes. Working with altium to create our PCB layout was a large hurdle in our project because of the process of learning the program. Throughout the process of designing this board, we worked closely with our sponsor to confirm that the device was designed properly before building the physical board.

Figure 2: Blue2 Device and Physical PCB Layout

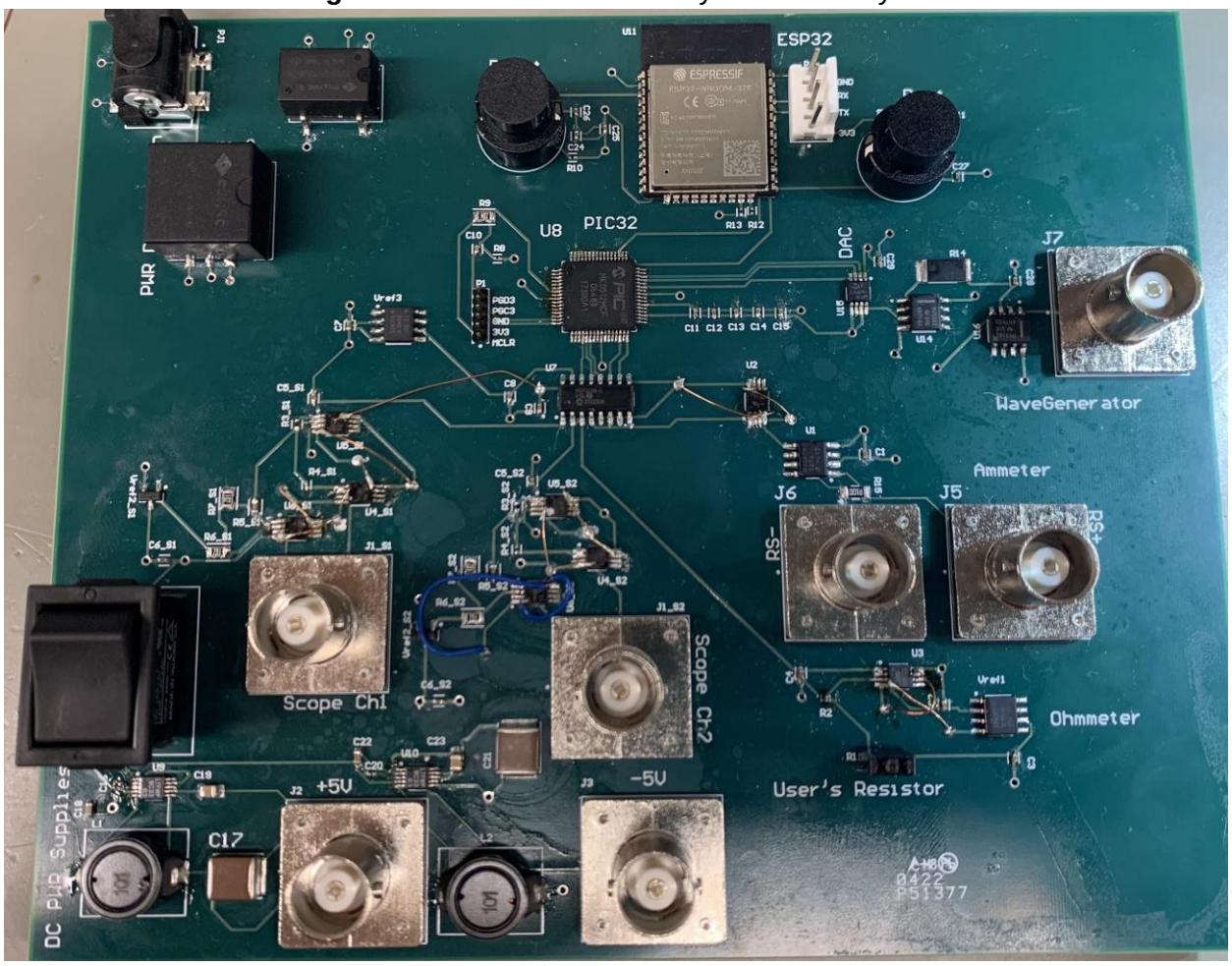


Figure 2 shows the physical Blue2 device. The design was based specifically on the design the team put together in Altium. However, during the process of testing and validating the functions we found that there needed to be redesigns. Those are the external wires on our board. Some of the major components that the user would need to know about on our board would be the probe jacks, the switches, and the power supply female connector. There are all components on our board that the user would need to interact with in order to use our device.

Subsystem Validation

The PIC32 serves as the main microcontroller that controls most of the functions on the PCB. PIC32 code was written in C using the xc32 compiler on MPLAB X. An external programmer was connected to the computer and MPLAB X recognized the device. The programmer converted the C code that compiled on the computer into hex instructions that could be run on the MIPS32 processor. Since MPLAB X did not allow for execution of code, most of the main code development was debugged using Visual Studio Code and its mingw-w64 compiler.

One way in which the PIC32 operated was to take in digital data from the connected analog-to-digital converter, manipulate that data using empirical formulas from simulations and hardware testing, and send that data to the ESP32 where it can be transmitted to the Andriod phone for use in the app. Another way in which the PIC32 operated was to generate digital data for the waveform generator subsystem and send that digital data to the connected digital-to-analog converter. In the figures below, the PIC32 runs through all of its correction calculations for the Voltmeter, Ammeter, and Ohmmeter.

Figure 3: PIC32 Main Loop Menu

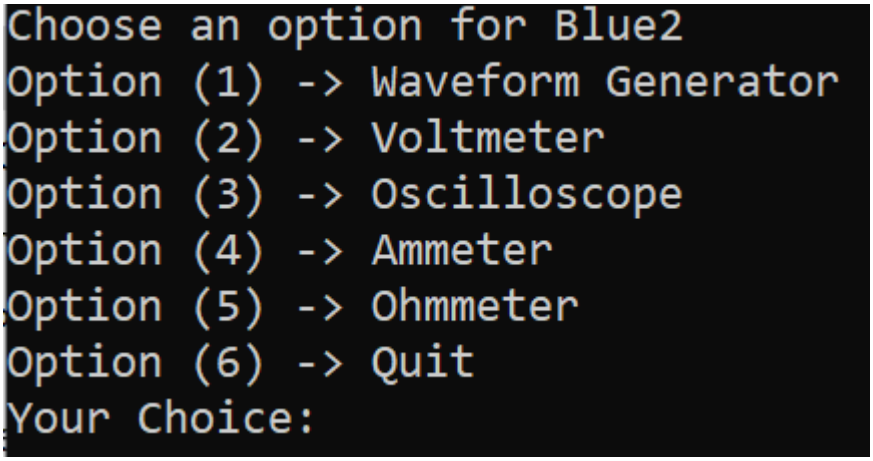


Figure 4: PIC32 Voltmeter Calculations

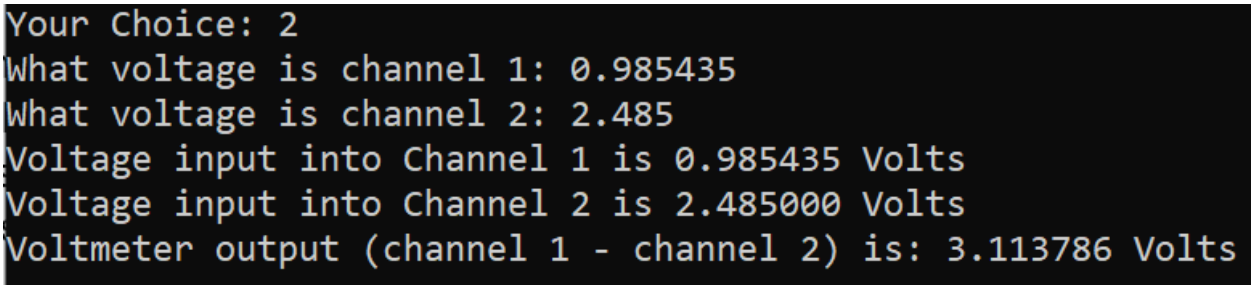


Figure 5: PIC32 Ammeter Calculations

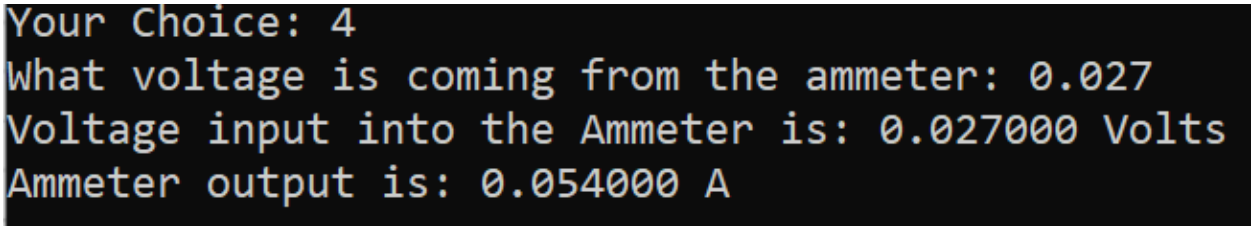
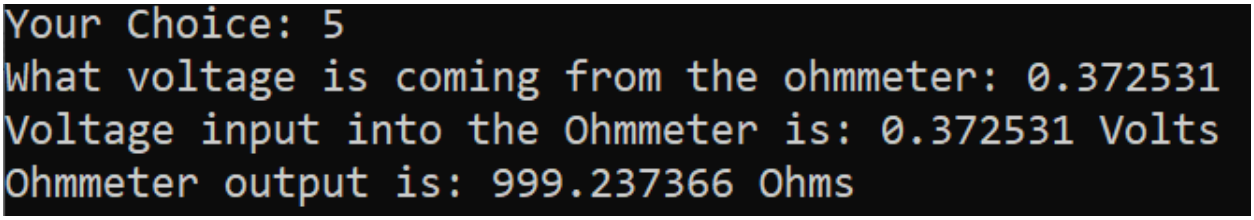


Figure 6: PIC32 Ohmmeter Calculations



Subsystem Conclusion

Through MPLAB X and the programmer, the code that ran the correction calculations was successfully loaded onto the hardware. However, in the design, we did not include a console port to be able to see what the PIC32 was outputting on its own. Since the code compiled on MPLAB X, we assumed that the code was running on the hardware as long as there was power to the board. For the PCB design and fabrication, more care was needed during the schematic design process to ensure that the schematics worked as intended. Many quick-fix remedies were required after the PCB was printed to get it to a working state.

Blue2
Jon Flores

Voltmeter Subsystem Report

Revision 1
4/30/2022

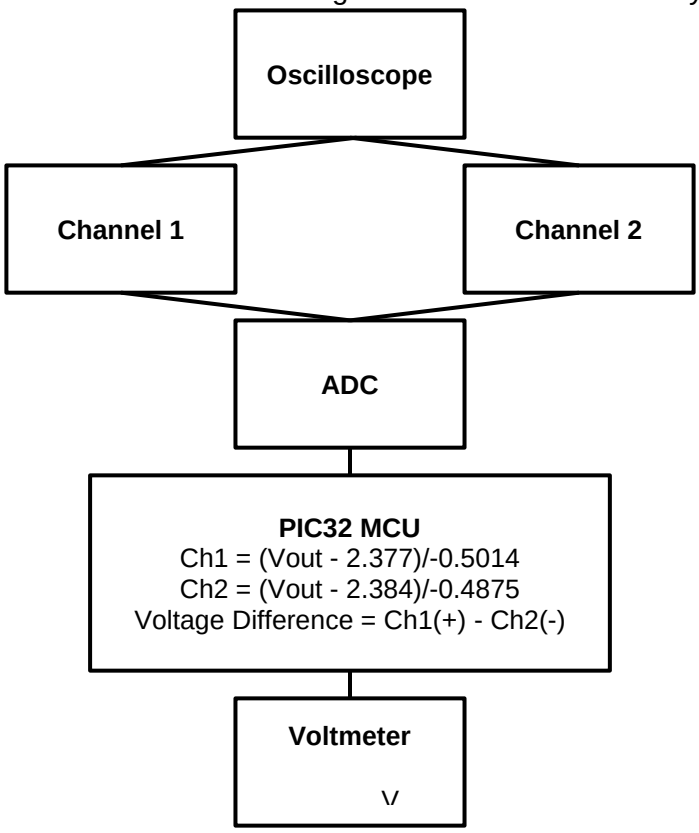
Subsystem Introduction

The voltmeter subsystem is designed to measure the difference in the electrical potential between two places, such as two points within a circuit. The subsystem is using the oscilloscope function to measure the two points of the circuit. It will aim to measure DC input values at specific points in a user's circuit.

Subsystem Details

The voltmeter design shows the user the electrical potential value they are measuring for on the phone application. The output of the oscilloscope design goes to the microcontroller which will handle the channel correction calculation and the calculation of the difference between two channels. A block diagram of the subsystem is shown below.

Figure 7: Functional Block Diagram of the Voltmeter Subsystem



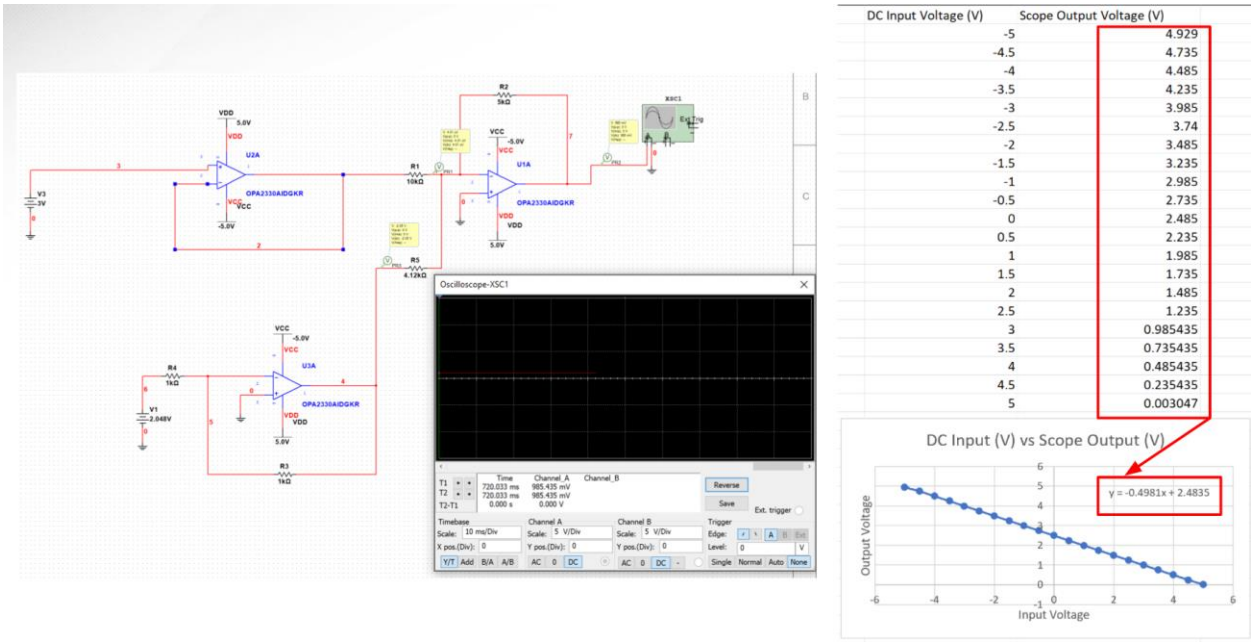
The primary challenge with the voltmeter design was ensuring the design would be able to handle DC and AC input voltages. When testing DC input voltages, the outputs gave an unusual output but we tested several different inputs. With this test, we were able to find the trend of the design. With these trend equations, we will use it in the PIC32 to show the correct value. The voltmeter design shows the user the electrical potential value they are measuring for on the phone application.

In 404, we did have to change the design slightly. First, the supply to our op amps were needing to be flipped. We also needed to replace a 1k ohm resistor with a 10k ohm resistor so that our value was the correct units. In addition we added an additional 10k ohm resistor to the design. This caused us to reroute some of the connections that were going to some of our op amps. In the results section of this report will show figures of the changes made and our results of the new design.

Subsystem Validation

The voltmeter subsystem was validated by using the same oscilloscope design in Multisim but with DC input voltages instead of AC input voltages. As mentioned in the subsystem details section, the output voltages we were receiving were not matching up with the input voltages. We created a table testing the complete range of DC voltages we should expect this device to handle and found a trend equation to use to calculate the correct output voltages. The figure below shows the equation we will use to convert the output values we were receiving. For the voltmeter, we will be using both oscilloscope channels so both of the channels' output values will go through the ADC before going through the PIC32. After the PIC32, the difference will be shown on the phone application.

Figure 8: ECEN 403 Validation of the Voltmeter



Subsystem Results

The voltmeter subsystem worked as intended. We were able to test this function on DC voltages going directly into the channels as well as sample circuits. We were using the oscilloscope to get these measurements so the values that were being outputted were needed to be manipulated. After these manipulations were getting the correct voltmeter values.

Figure 9: Final Oscilloscope Channel 1 and 2 Design in Altium

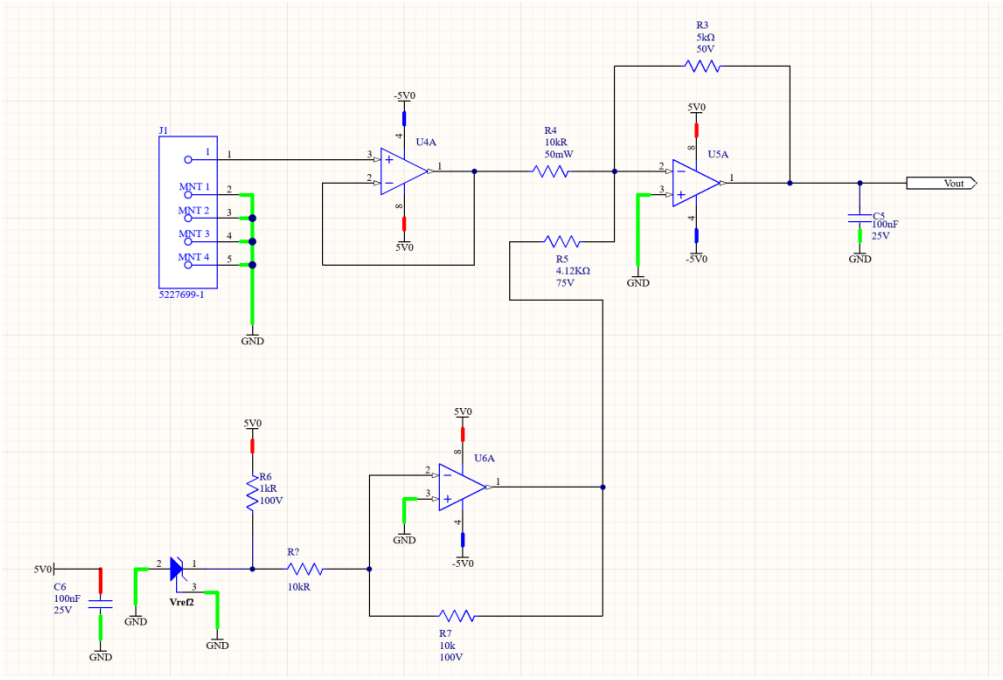
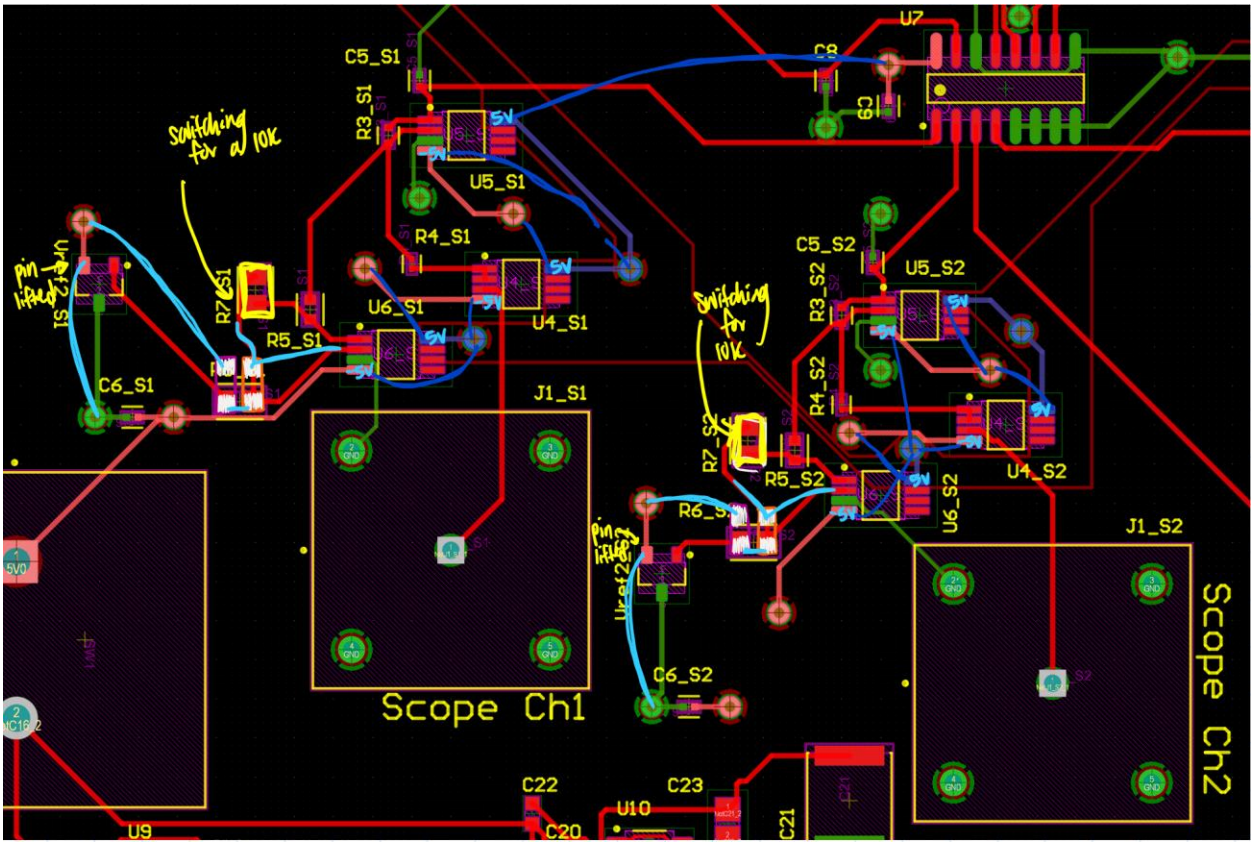
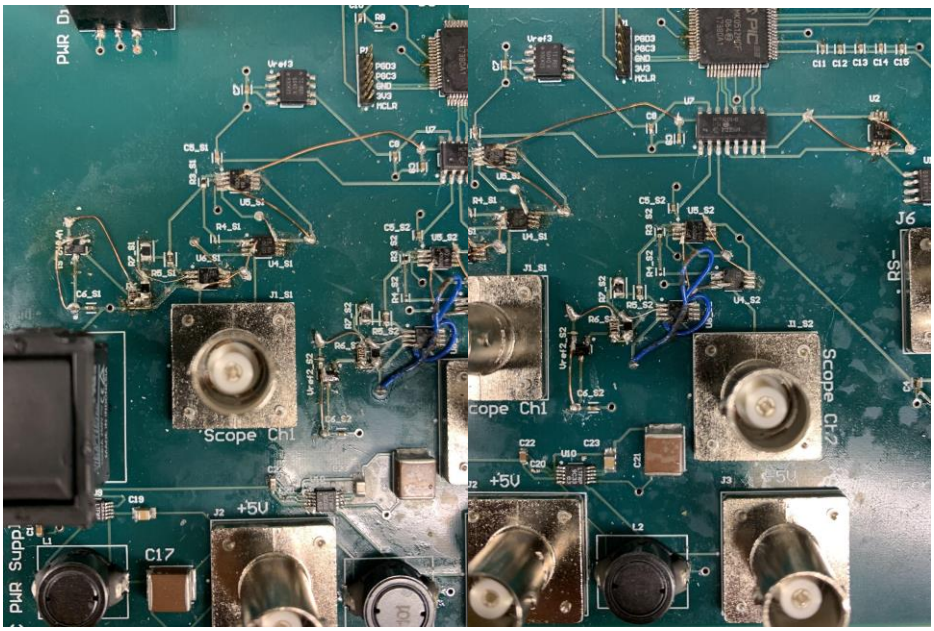


Figure 10: Final Oscilloscope Channel 1 and 2 Layout in Altium



Above is the projects final oscilloscope channels designs. It shows our initial design and then all of the rewiring and new components we had to add. The dark blue lines indicate the op amp power supply corrections. We needed to flip the -5 V supply with the +5 V supply on all of our op amps. We found out that this was an issue because it was giving us a voltage drop on the power distribution to our entire board. After these corrections were made, the power distribution to the board was working properly. The light blue lines indicate the external wires that were needing to placed to reroute the outputs to the correct location. The yellow box indicates the change in resistor value of a 1k ohm resistor to a 10k ohm resistor. The orange box indicates the new 10k ohm resistor we placed in our design. The purple box located next to it indicates the existing 1k ohm resistor the we just changed in our design. Finally, the yellow messages on the figure shows the pins of the components that we needed to lift in order to get the right connections.

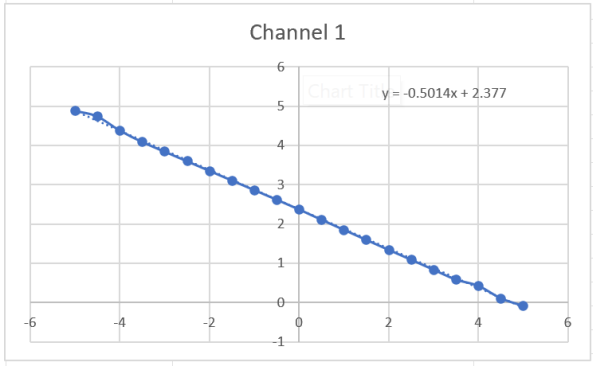
Figure 11: Physical Oscilloscope Channels on Blue2 Device

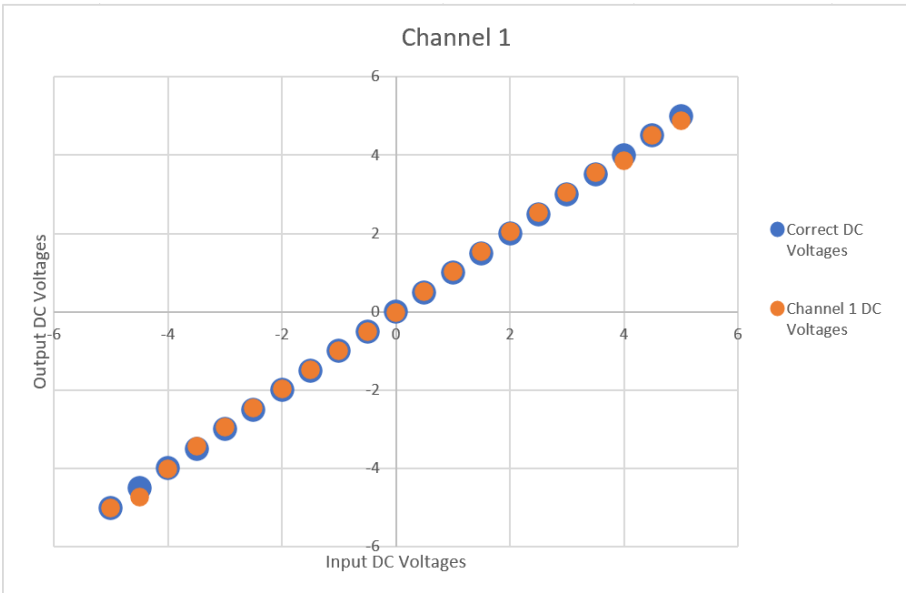


After going through all of the design changes, this report will now go through the data we collected for the voltmeter function as well as show the accuracy of our channels.

Figure 12: Channel 1 Data and Accuracy

DC Input Voltage (V)	Scope Output Voltage (V) - Channel 1	After PIC Calculation	Error %
-5	4.887	**-5.006**	0.12
-4.5	4.747	**-4.727**	5.04
-4	4.39	**-4.015**	0.37
-3.5	4.098	**-3.432**	1.93
-3	3.852	**-2.942**	1.94
-2.5	3.608	**-2.455**	1.79
-2	3.363	**-1.966**	1.68
-1.5	3.118	**-1.478**	1.48
-1	2.874	**-0.991**	0.88
-0.5	2.63	**-0.505**	0.92
0	2.381	**-0.008**	#DIV/0!
0.5	2.1189	**0.515**	2.95
1	1.863	**1.025**	2.51
1.5	1.606	**1.538**	2.51
2	1.352	**2.044**	2.21
2.5	1.101	**2.545**	1.79
3	0.8482	**3.049**	1.64
3.5	0.5957	**3.553**	1.50
4	0.437	**3.869**	3.27
4.5	0.1205	**4.500**	0.01
5	-0.0736	**4.888**	2.25

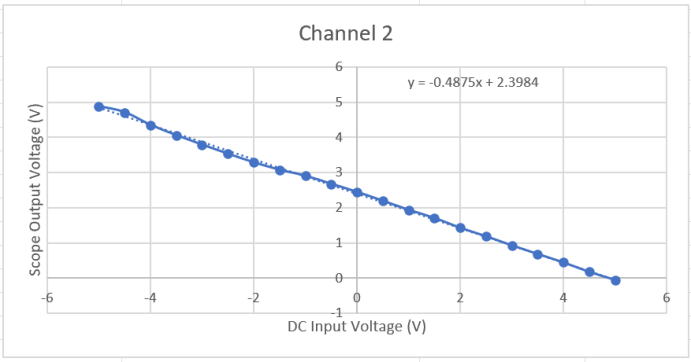


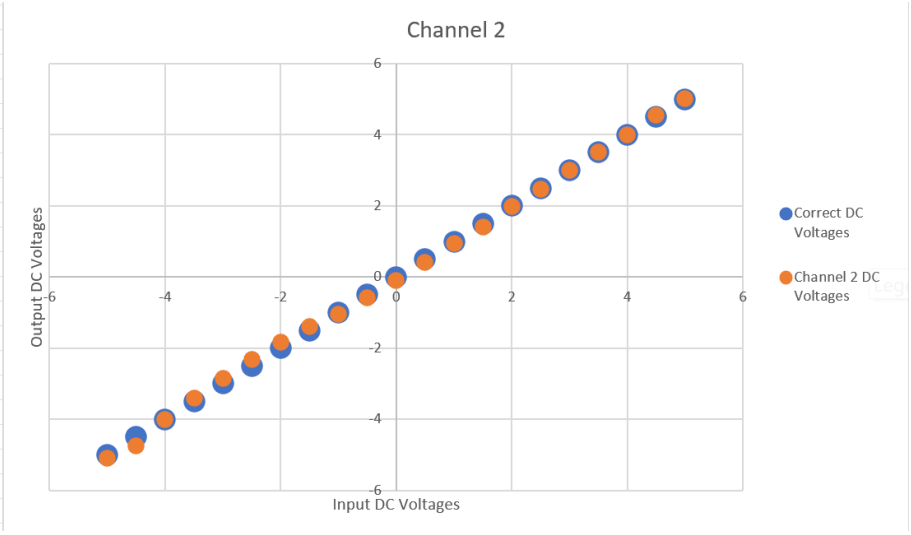


This data runs through a number of tests and already makes the correction for the channel using the trend line equation to solve for the correct voltage value.

Figure 13: Channel 2 Data and Accuracy

DC Input Voltage (V)	Scope Output Voltage (V) - Channel 2	After PIC Calculation	Error %
-5	4.8794	**-5.089**	1.78
-4.5	4.7099	**-4.742**	5.37
-4	4.349	**-4.001**	0.03
-3.5	4.0574	**-3.403**	2.77
-3	3.787	**-2.848**	5.05
-2.5	3.531	**-2.323**	7.07
-2	3.287	**-1.823**	8.86
-1.5	3.076	**-1.390**	7.34
-1	2.906	**-1.041**	4.12
-0.5	2.678	**-0.574**	14.71
0	2.448	**-0.102**	#DIV/0!
0.5	2.197	**0.413**	17.37
1	1.939	**0.942**	5.76
1.5	1.709	**1.414**	5.72
2	1.429	**1.989**	0.57
2.5	1.189	**2.481**	0.77
3	0.931	**3.010**	0.34
3.5	0.686	**3.513**	0.36
4	0.447	**4.003**	0.07
4.5	0.1789	**4.553**	1.17
5	-0.0486	**5.019**	0.39

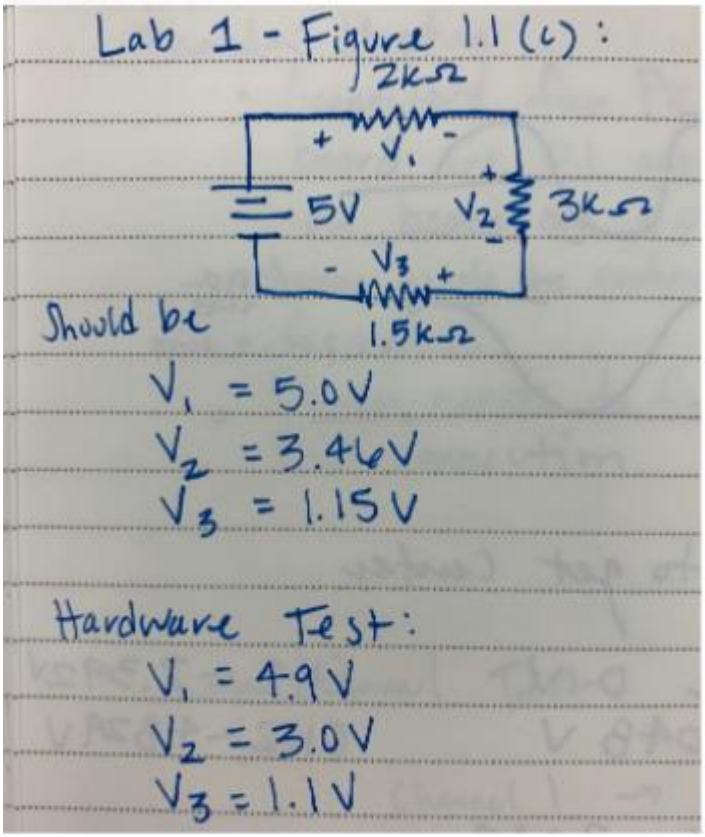




This data runs through a number of tests and already makes the correction for the channel using the trend line equation to solve for the correct voltage value.

We were also able to run lab tests with our hardware to verify that these functions would still work when testing a simple circuit that the user would need to run.

Figure 14: Lab 1 Test and Accuracy of Hardware



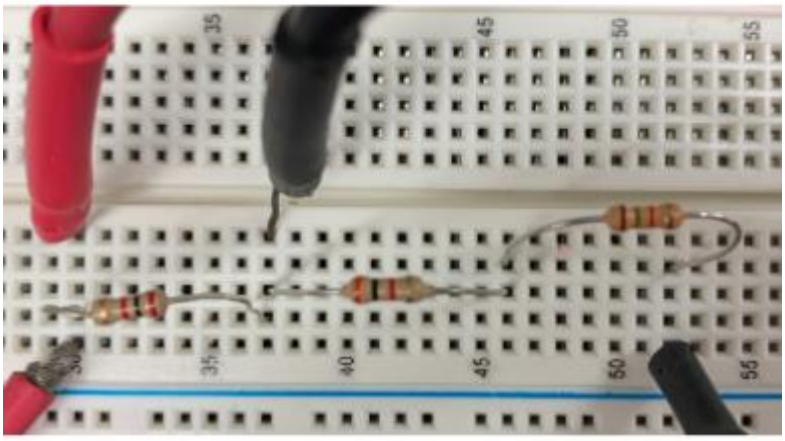
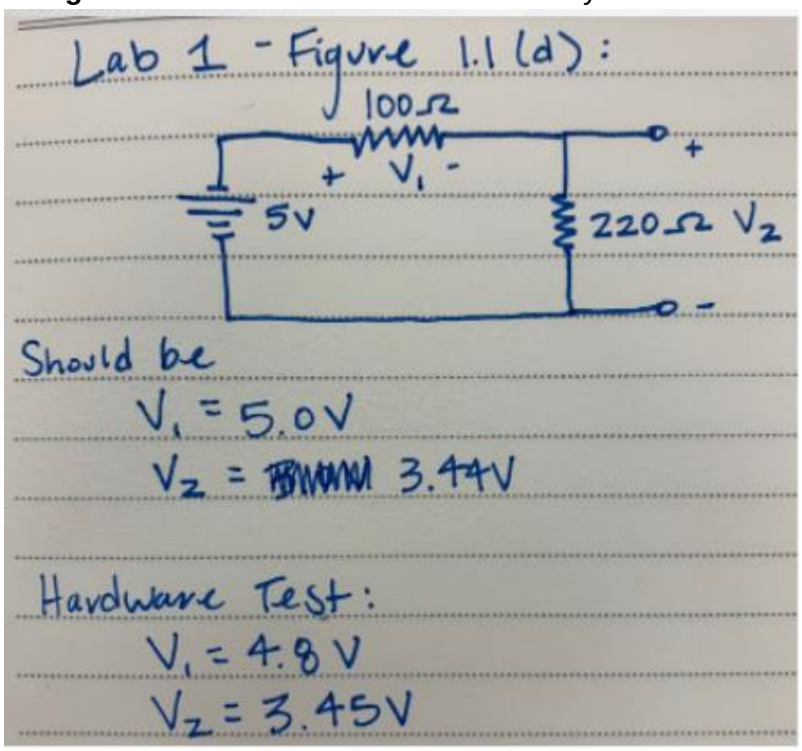
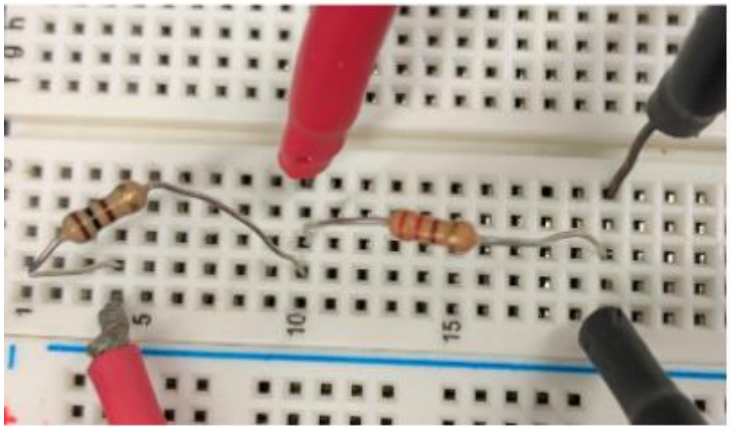


Figure 15: Lab 1 Part 2 Test and Accuracy of Hardware





Subsystem Conclusion

The voltmeter subsystem was shown to be working correctly. The oscilloscope effectively takes in the DC input voltage and output values that we can calculate to match the input values. Using the two channels is required for the voltmeter to work. The user will be required to place each channel at two different locations and the PIC32 will calculate the corrections and the difference between the two. The voltage measurement will be prompted to the user on the phone application. However, because of our lack of integration between the hardware and our phone application we were not able to prompt the result. All of these tests and results were done strictly with the hardware and testbench equipment.

Blue2
Jon Flores

Oscilloscope Subsystem Report

Revision 1
4/30/2022

Subsystem Introduction

The oscilloscope subsystem is designed to allow the ECEN 215 student to see how voltage changes over time for their circuit. The subsystem is powered by +5V and -5V. It will take in an AC voltage input from the user's probes. The oscilloscope function will have 2 channels with respective ground references. The oscilloscope subsystem was tested to confirm its stability and range of outputs.

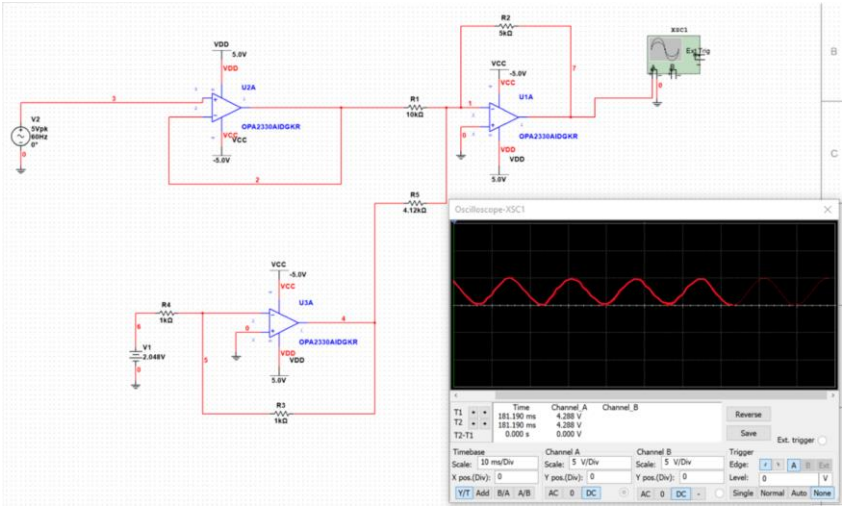
Subsystem Details

The oscilloscope subsystem shows the user measured voltages over time through a waveform of electronic signals on a graph. With the voltage representing the vertical axis (y) and time on the horizontal axis (x). This display will allow the student to determine if the behavior of their circuit is working correctly. It will also show any problems within the circuit such as unwanted noise. All of these details of the oscilloscope will be displayed through the Blue2 phone application under the oscilloscope option.

Subsystem Validation

The oscilloscope design was validated by generating a sinusoidal waveform that stayed between the range of 0V to 5V because of the Analog-to-Digital Converter only being able to take in positive values. This waveform was generated using a 5V AC voltage input that went through a series of voltage follower, inverting summing, and inverting amplifiers. On the bottom inverting op amp, there is a negative voltage reference of 2.048 V for the sole reason of shifting up the waveform to the positive voltage output.

Figure 16: Validation of the Oscilloscope



With this test, we were able to see that the oscilloscope design was able to output the desired waveform. The users will place their probe wires at the desired locations on their circuit so the voltage inputs will change but will stay within the desired range.

Subsystem Results

For the oscilloscope design we had several changes to the function that this report will go into further detail below. For our design, both channels were designed exactly the same which is why we expected to have the same outputs for both channels. When comparing it to the voltmeter which uses the same function, we could see that they were consistent with slight differences. For our oscilloscope outputs going to the ADC only being between 0 and 5 V we needed to shift down the output by half of the voltage reference. For channel 1 we found that the mid point would be about 2.5 V. The channel 2 midpoint was around the same value. This midpoint value would be subtracted from the outputs we were getting from our hardware tests. The shift down would be performed in the PIC32 before the output is sent to the phone application. In the figures below, there is the final design, final pcb layout, final physical design on our device, and then tests performed to justify the accuracy of our oscilloscope function.

Figure 17: Final Oscilloscope Channel 1 and 2 Design in Altium

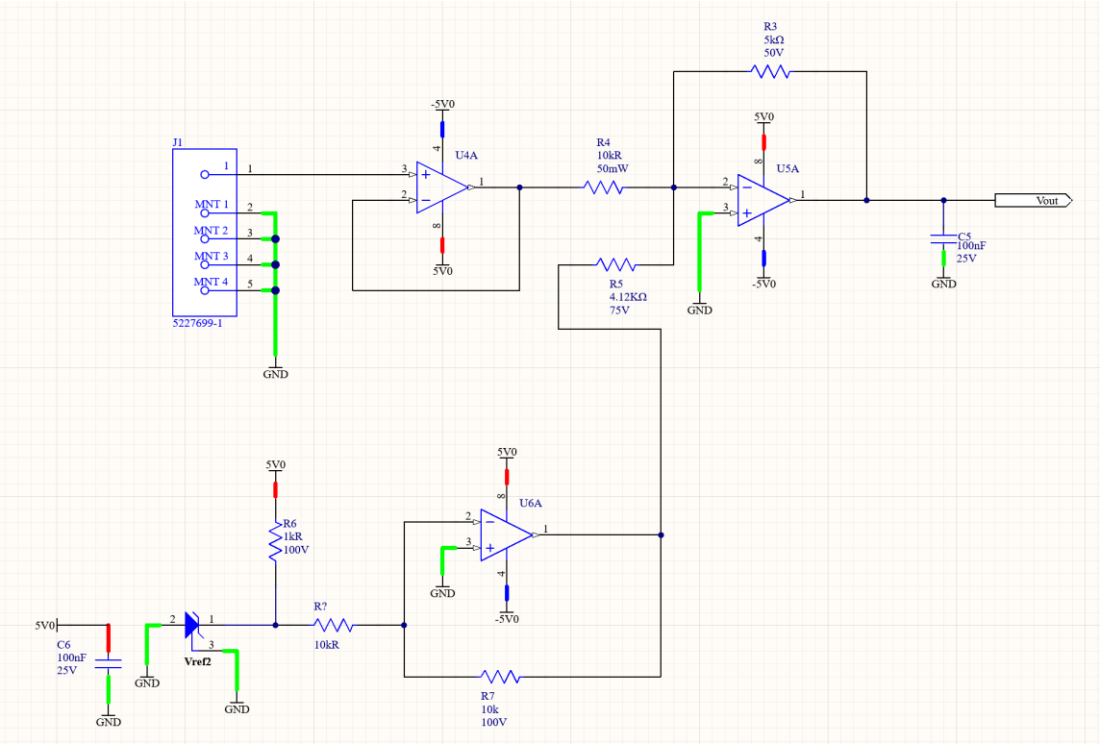
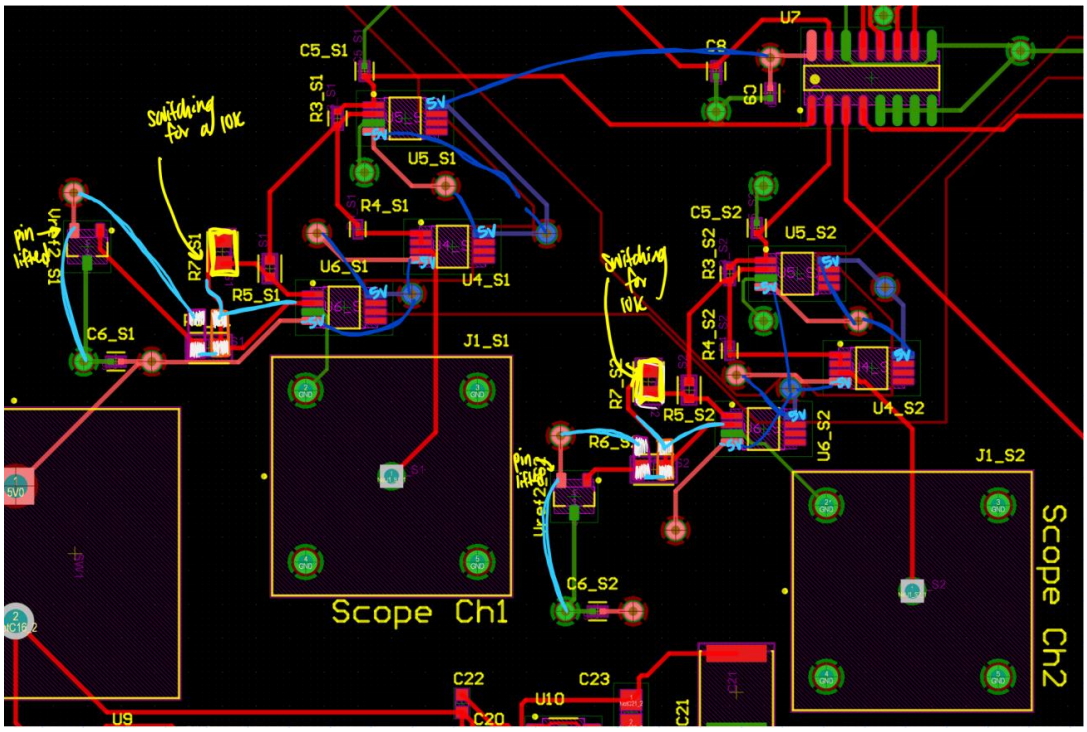


Figure 18: Final Oscilloscope Channel 1 and 2 Layout in Altium



Above is the projects final oscilloscope channels designs. It shows our initial design and then all of the rewiring and new components we had to add. The dark blue lines indicate the op amp power supply corrections. We needed to flip the -5 V supply with the +5 V supply on all of our op amps. We found out that this was an issue because it was giving us a voltage drop on the power distribution to our entire board. After these corrections were made, the power distribution to the board was working properly. The light blue lines indicate the external wires that were needing to placed to reroute the outputs to the correct location. The yellow box indicates the change in resistor value of a 1k ohm resistor to a 10k ohm resistor. The orange box indicates the new 10k ohm resistor we placed in our design. The purple box located next to it indicates the existing 1k ohm resistor the we just changed in our design. Finally, the yellow messages on the figure shows the pins of the components that we needed to lift in order to get the right connections.

Figure 19: Physical Oscilloscope Channels on Blue2 Device

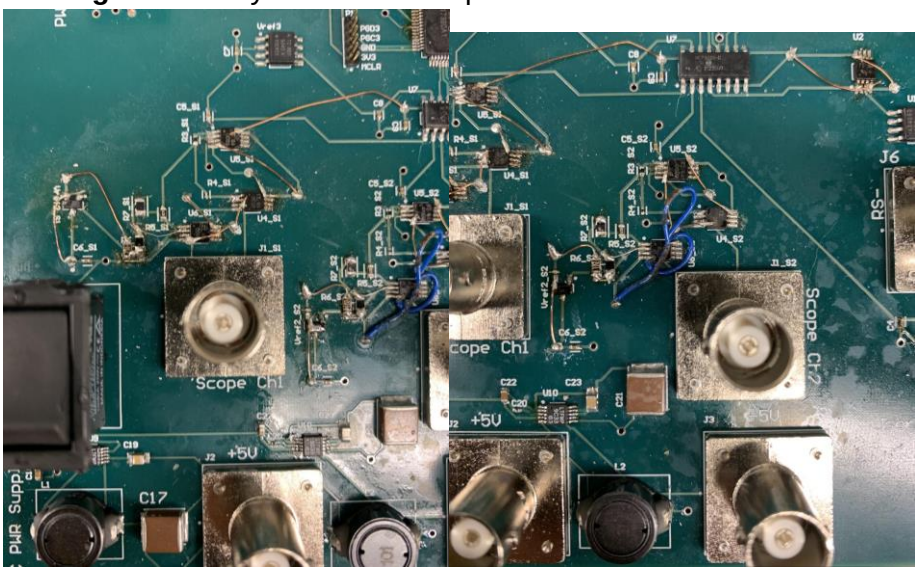


Figure 20: Oscilloscope Simulation vs. Hardware Output Test 1

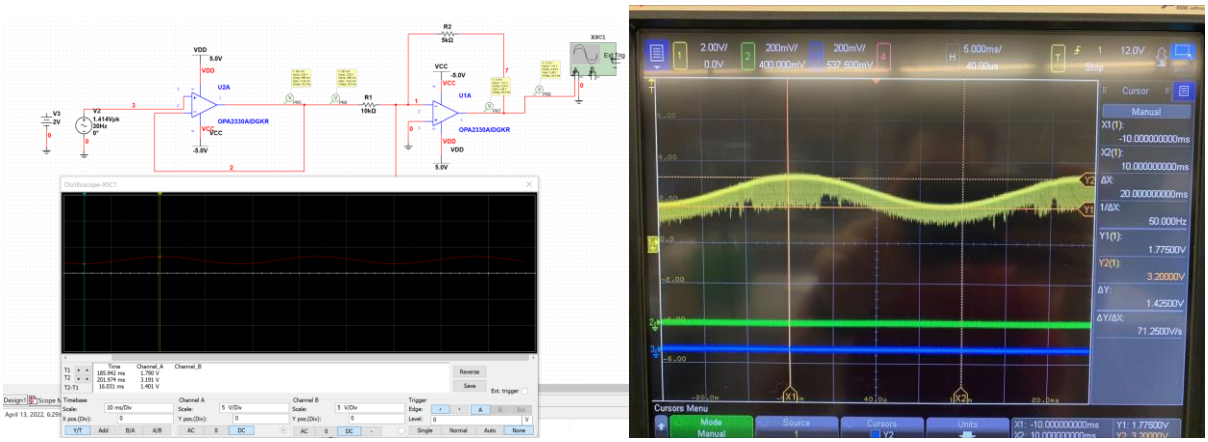


Figure 21: Oscilloscope Simulation vs. Hardware Output Test 2

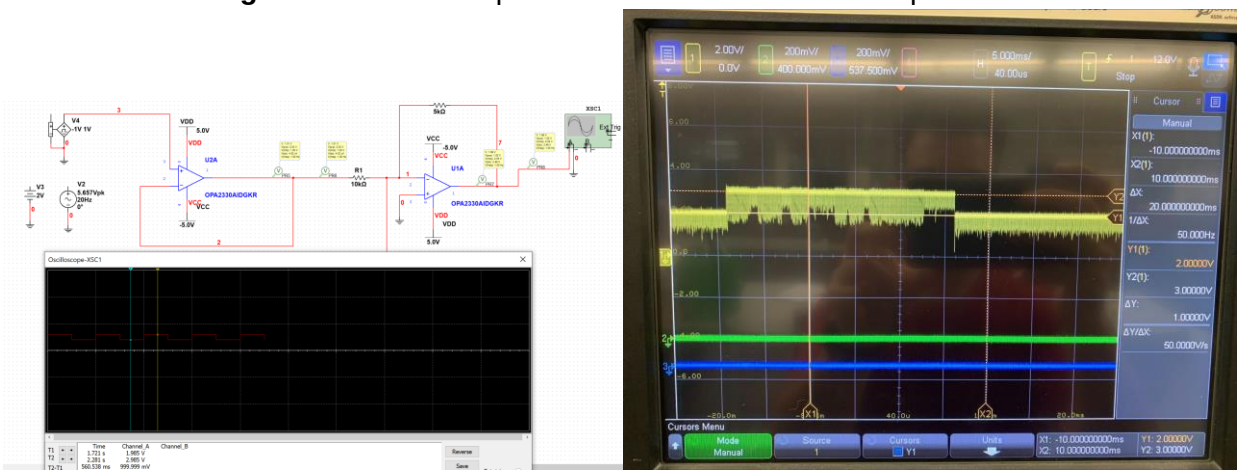


Figure 22: Oscilloscope Simulation vs Hardware Output Test 3

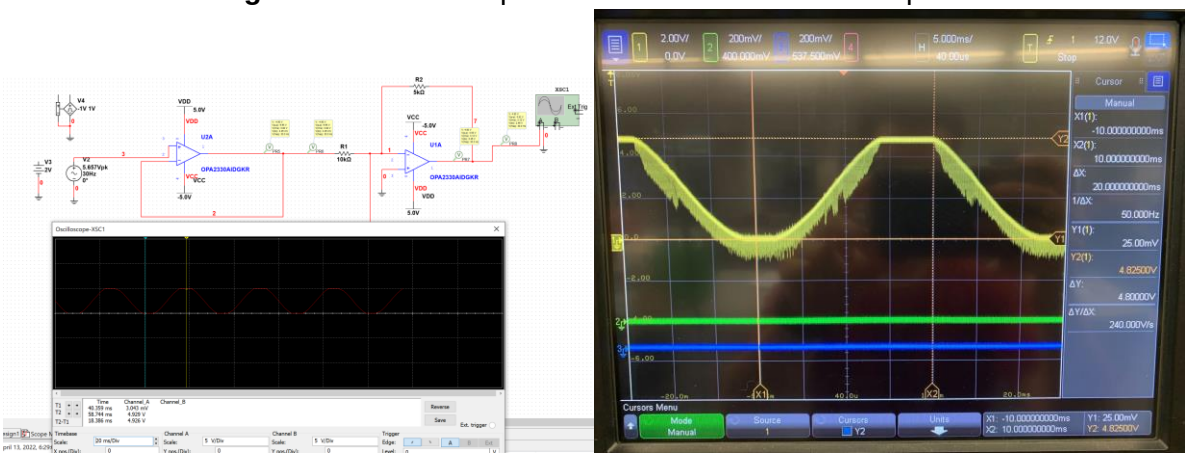


Figure 23: Oscilloscope Simulation vs Hardware Output Test 4

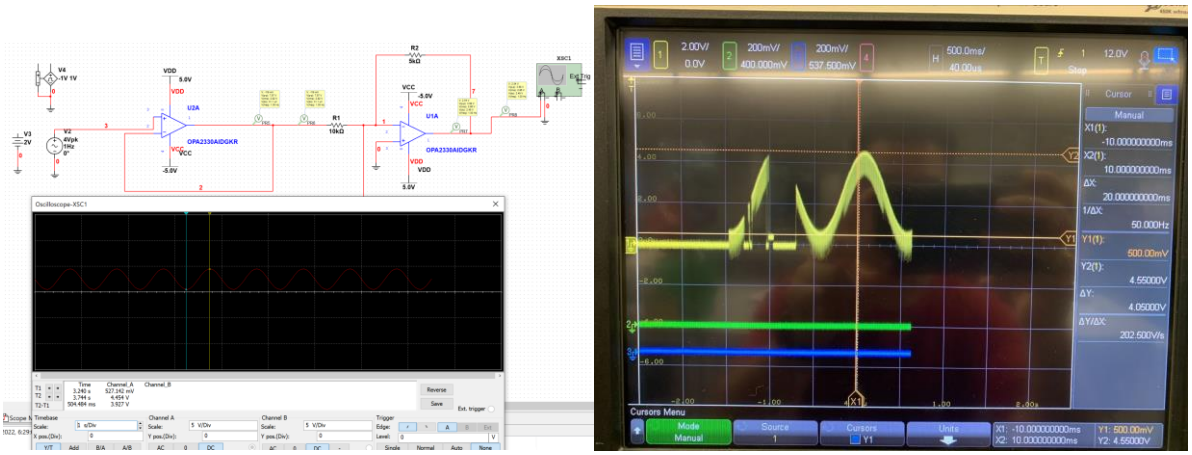
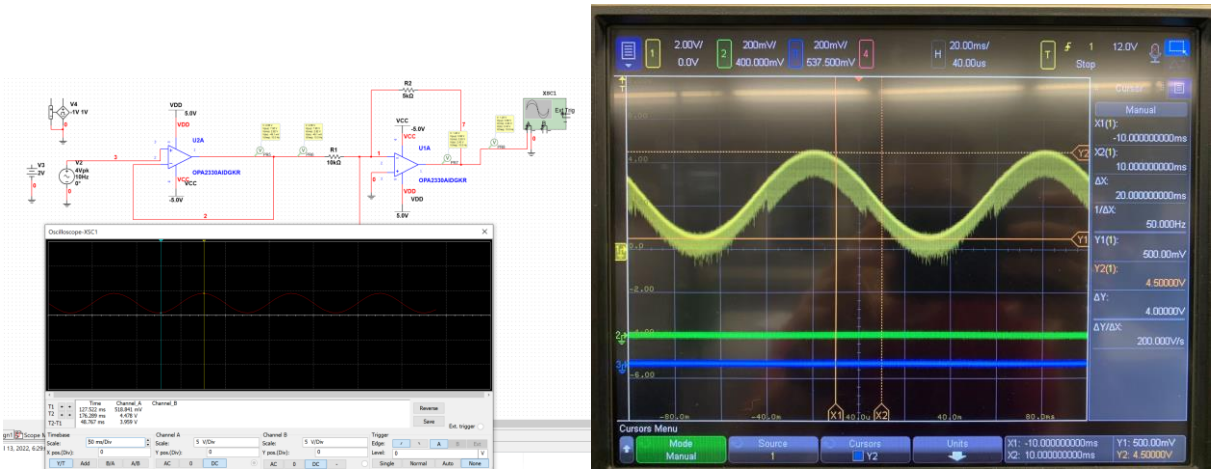


Figure 24: Oscilloscope Simulation vs Hardware Output Test 5



Subsystem Conclusion

The oscilloscope subsystem was shown to be working correctly. However, we were not able to send the outputs to the PIC32 and verify that the corrections being made to the waveform were doing what they were supposed to. The oscilloscope effectively takes in the AC input voltage and outputs the desired waveform showing the voltage change over time. Having two channels, the user will be able to compare voltages over time in two different locations of their circuit if they want to. These waveforms will be prompted to the user on the phone application. With the lack of integration between the phone application and the physical device it was not possible to test if this was going to work properly or not.

Aaron Gavin

Ammeter Subsystem Report

Revision 1

4/30/2022

Subsystem Introduction

The ammeter subsystem is designed to measure current through a resistive load.

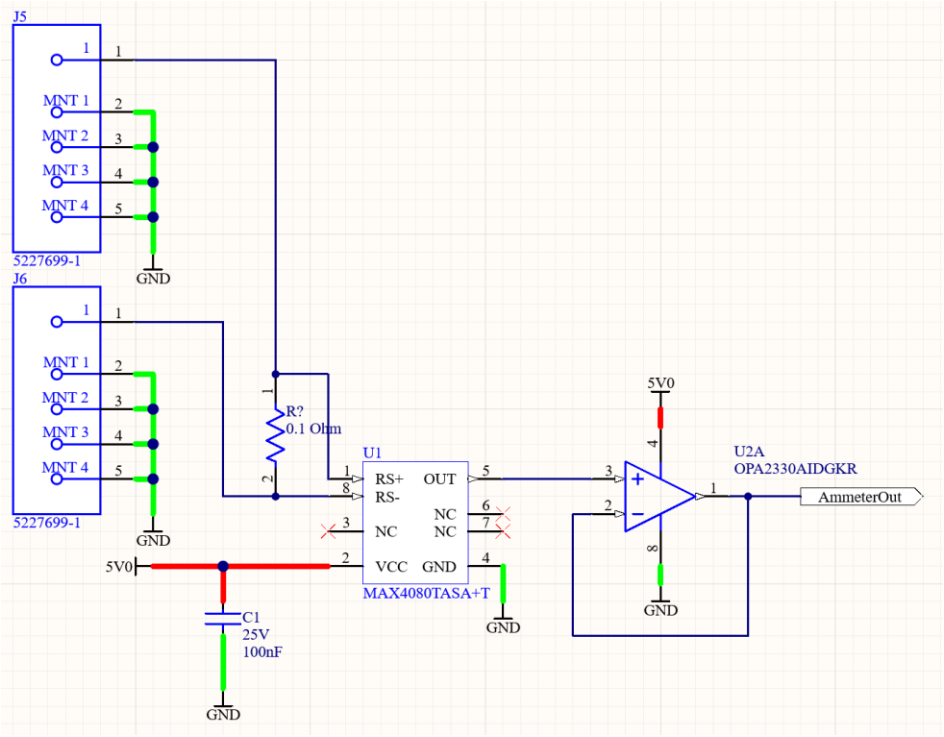
Subsystem Details

Current is determined by measuring the voltage across a known 0.1 Ohm shunt resistance and calculating the current using ohm's law. The current-sense amplifier is designed to divide the voltage on the output pin by 2. Once the voltage signal reaches the PIC32 microcontroller, the voltage is multiplied by 2 to reach an effective current value. This value is then forwarded to the ESP32 for transmission onto the Android app. The current-sense amplifier is used so that the voltage reading is not dependent on the 5 V that is supplied to the chip or the buffer op-amp. This removes any dependence or variation on the user's circuit voltage and allows us to more accurately measure the voltage across the shunt resistance. A noise-reduction capacitor is used on the supply voltage line to ensure a stable 5 V supply.

Subsystem Validation

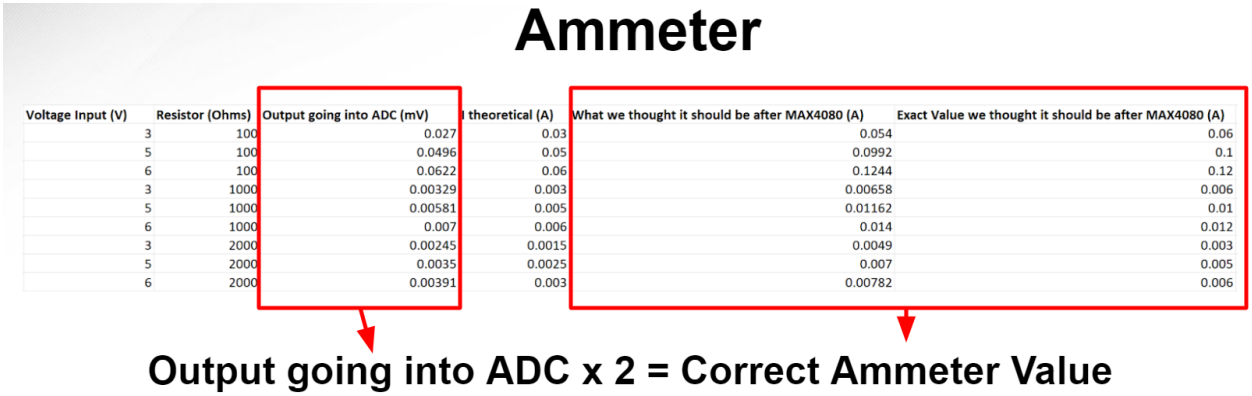
The ammeter was validated by setting up a test resistor and a test voltage source on a breadboard. Two input probes are then connected to each side of the test resistor. Output voltage is measured after the last buffering op-amp. Correction calculations (multiply by 2) are then used to obtain the final measured values of the current.

Figure 25: Ammeter Schematic



A total of nine tests were conducted each with different source voltages and test resistors. Refer to Figure 12 for the table of specific tests. The final current calculations were very close to the theoretical values that were determined through simulations.

Figure 26: Validation of the Ammeter



Subsystem Conclusion

The Ammeter subsystem was shown to be working properly. The ammeter determines the current across a user's resistor by measuring the voltage across a small shunt resistor and calculating current via Ohm's law. The value calculated will be displayed to the user via the phone application.

Jon Flores

Ohmmeter Subsystem Report

Revision 1

4/30/2022

Subsystem Introduction

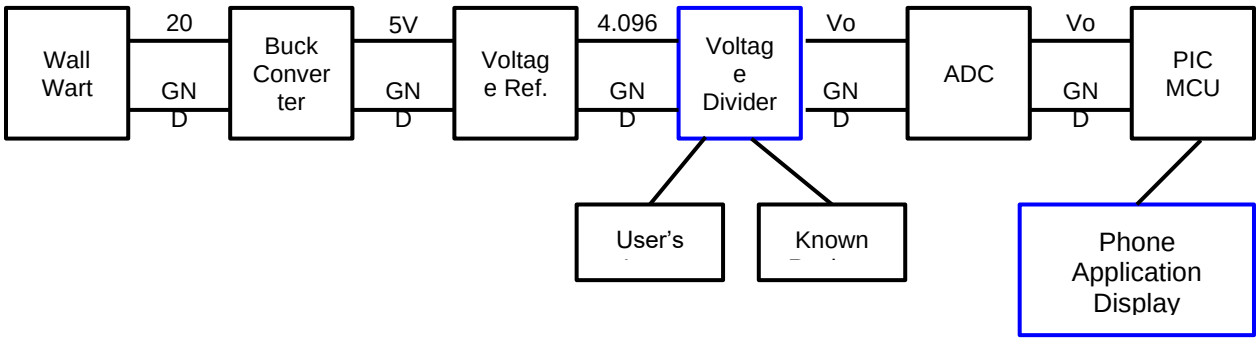
The ohmmeter subsystem is designed to measure electrical resistance. The subsystem is powered by a wall wart that has been stepped down by buck converters to 5V. This function on

our device is designed to be a voltage divider, where we solve for the user's input resistor. The ohmmeter subsystem was tested to show its accuracy to the user's input resistor. Knowing the resistances range the ECEN 215 lab manual requires allowed us to verify that our ohmmeter subsystem will work properly.

Subsystem Details

A block diagram of the subsystem is shown below.

Figure 27: Functional Block Diagram of the Ohmmeter Subsystem



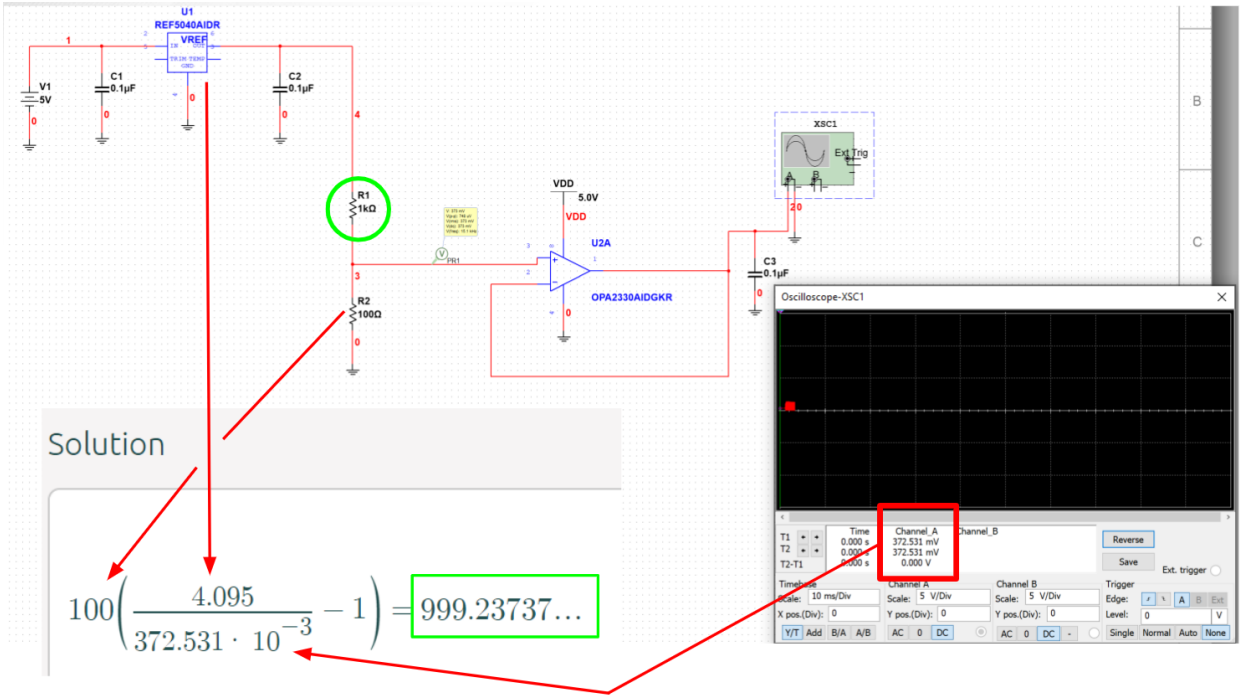
The primary challenge with the ohmmeter design was ensuring that there was a proper connector for the user's input resistor and that the correct output was going through the op amp. The design of the ohmmeter included placing the user's resistor in the right part of the voltage divider so that the ohmmeter was able to work properly. A manipulated voltage divider equation that solves for the user's input resistor will be plugged into the PIC32. The PIC32 will handle the calculations and then send the value to the phone application to prompt the user of the electrical resistance value of the resistor.

Our ohmmeter design will only be required to measure in a range of 47Ω to 10kΩ.

Subsystem Validation

In 403 the ohmmeter subsystem was validated by placing a random input resistor into our schematic and calculating if the output was the correct value. We took a copy of the Altium schematic and placed it into Multisim where we were able to find our voltage output. This voltage output was the last known value we needed to solve for the input resistor. The other known values were the 100 ohm resistor and the voltage reference (4.096 V). We plugged these values into the manipulated voltage divider equation and solved for the random input resistor.

Figure 28: Multisim Simulation Validation of the Ohmmeter



The random input resistor validation shown above points out the known values (red) and the random input resistor we are testing for (green). From this test, we were able to verify that the ohmmeter design would be able to measure the input resistors the user will be testing. The bottom left of the figure shows the calculation that will be done in the PIC32 microcontroller.

In 404, the ohmmeter subsystem was validated by placing a known range of input resistors that are required to measure in the lab into our schematic and calculated if the output was the correct value. The range is from 47Ω to 10kΩ.

Subsystem Results

The ohmmeter design was designed as a voltage divider. We had a known Vref, Known 100 ohm resistor, and the Vout going into the ADC would be the Vout. For our design in 404 we stuck with the design we had in 403 with a few minor changes. We initially had an amplifier buffering the output going into the ADC but there was a significant difference between what was going into the op amp and what was coming after. We decided to remove this op amp and our design just needed corrections of our outputs done in the PIC32 before it is shown to the user on the phone application. Below shows the Altium view of our initial design when we printed our PCB and below that shows the physical function on our board before we removed the op amp.

Figure 29: Ohmmeter Function in Altium

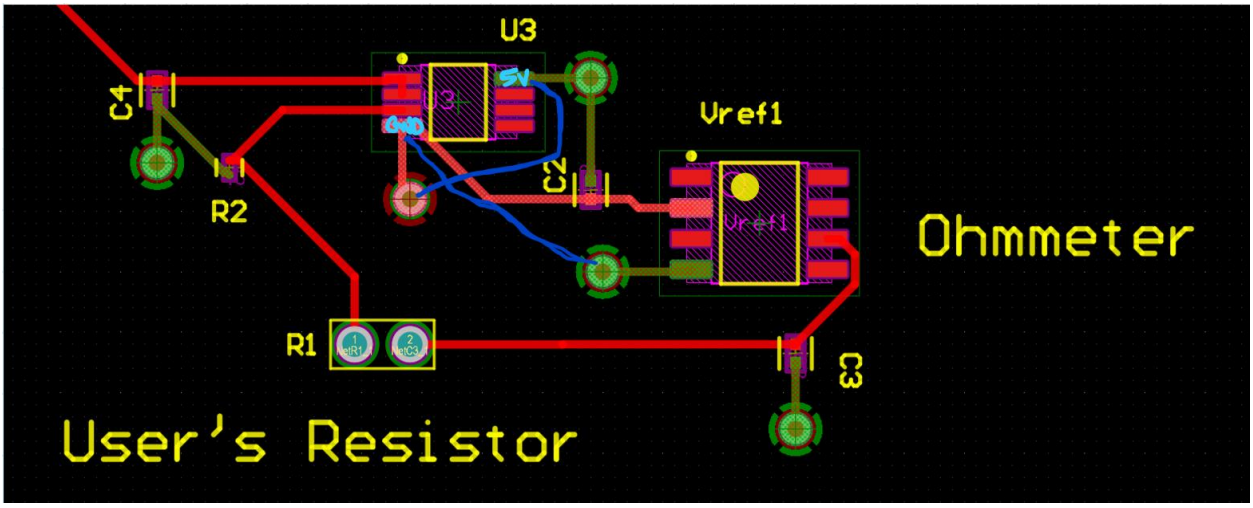
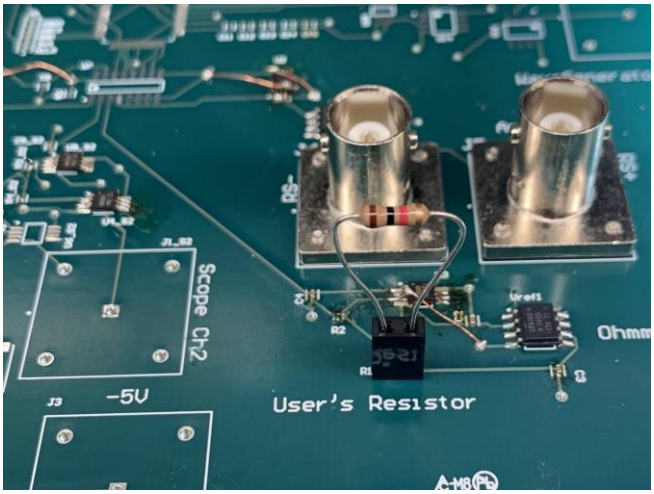


Figure 30: Ohmmeter Function on the PCB



The ohmmeter subsystem was required to measure resistors accessible to the user within the range of 47Ω to 10kΩ.

Figure 31: Validation of the Ohmmeter

Resistor Value (Ohms)	47	100	220	1k	1.5k	2k	3k	5.1k	10k
Ohmmmeter Design Output (Ohms)	129.412	132.31	214.186	974.86	1457.24	1935.67	2949.51	4957.43	9698.57
Ohm Correction Range (Ohms)	128 - 130	131 - 133	210 - 230	950 - 1100	1400 - 1600	1850 - 2100	2900 - 3100	4800 - 5200	9000 - 11000
Ohmmeter Result (Ohms)	47	100	220	1000	1500	2000	3000	5100	10000

The figure above shows the data taken when testing our Ohmmeter function on our device. The first row shows the resistor value of the resistor that is placed in the function for testing. The second row is the output of our design after corrections would have been made in the PIC32. The equation used for that is OhmValue = 100*((4.095/Vout)-1). The Vout value is the value going directly into our anolog to digital converter which then goes into our PIC32. The third row on this table is a range of measurements that will tell our ohmmeter which resistor our design is

outputting. This communication is being done in the PIC32 code. Since we have a set of resistors that the lab manual requires the user to measure and they are not close in value we are able to have these ranges. The last row is the value the ohmmeter function would output to the user.

Subsystem Conclusion

The ohmmeter subsystem is working as intended. The ohmmeter effectively takes in the user's input resistor and outputs the last known value of the voltage divider. With all that we know, the calculations done show the correct value of the input resistor. This value found will be prompted to the user on the phone application. With the lack of integration with the phone application, we are only able to prepare the code for the corrections and show how the output we are getting from our design will be manipulated. It was noted during our demo that some changes could be made for a better design. Essentially, the best way to redesign this function would be to attach probes to the function and show the resistance between the two probes. This will be covered in more detail in our section where we will provide comments for the next team to work on this project.

Aaron Gavin

DC Power Supplies Subsystem Report

Revision 1
4/30/2022

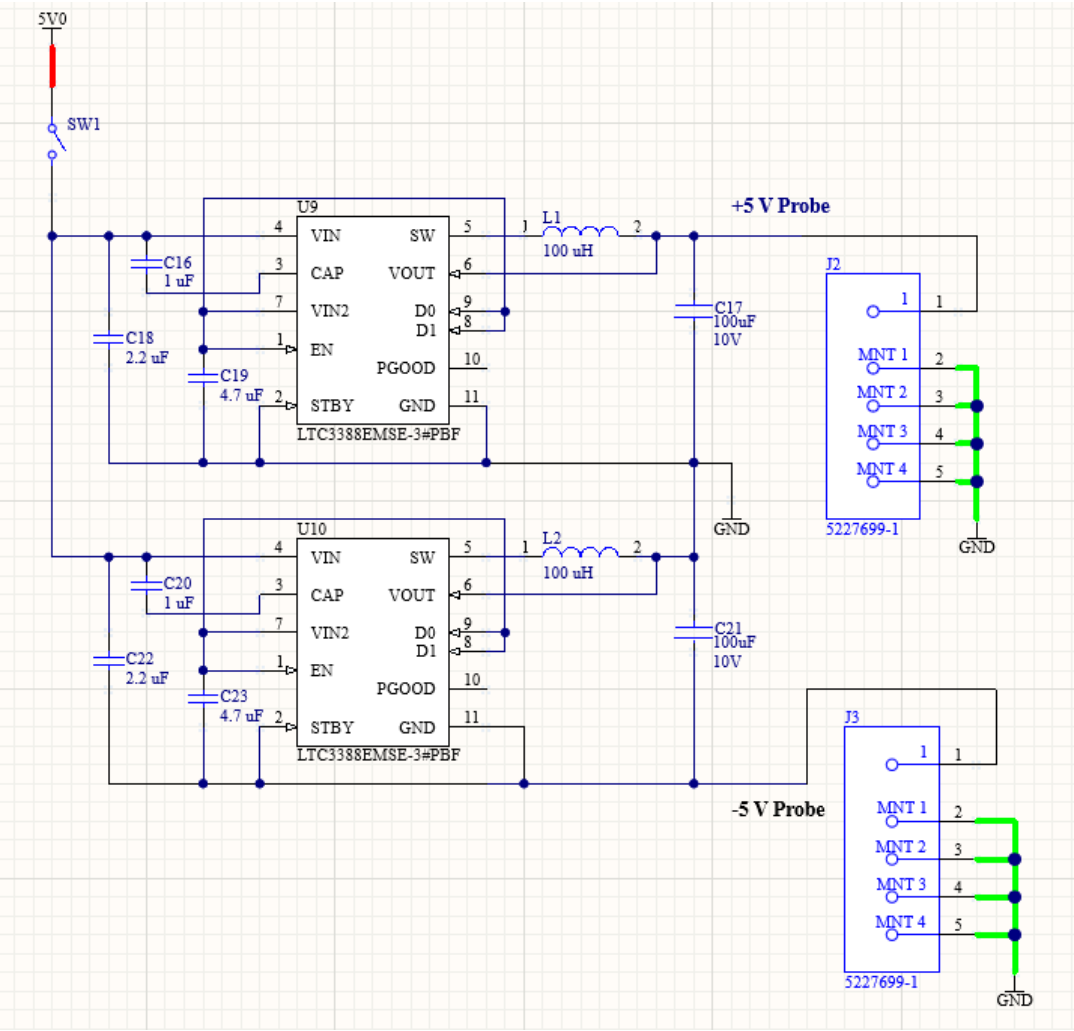
Subsystem Introduction

The DC Power Supplies subsystem provides the user with stable and consistent 5 V and -5 V sources for use in analog circuits, particularly as voltage supplies for op-amps.

Subsystem Details

The DC Power Supplies circuit required a 5 V source to reach the desired outputs of DC regulated 5 V and - 5 V. The subsystem consists of two switching buck regulators, which help stabilize the output voltages at their required levels as well as capacitors and inductors to further facilitate the splitting and stabilization of the output voltages. The circuit utilizes ground at the midpoint of the schematic, allowing the 5 V output from 5 V to 0 V and the -5 V output from 0 V to 5 V. During the design process, the -5 V supply was not working as intended, so a branch of the -5 V output from the Power Distribution subsystem was rerouted to the -5 V output. A rocker switch is located at the head of the design to cut off the voltage supplies when not in use. Due to the -5 V rerouting, the rocker switch only cut the 5 V supply while the -5 V supply remained constant.

Figure 32: Schematic of the DC Power Supplies



Subsystem Validation

The DC Power Supplies subsystem was validated by measuring the output voltage of the 5 V and -5 V outputs and checking whether the correct voltage values were being produced. Both the 5 V and -5 V outputs were shown to produce the correct voltage values.

Figure 33: Validation of the DC Power Supplies

Subsystem Conclusion

The DC Power Supplies subsystem was shown to be operating properly. The users of the Blue2, mainly ECEN 215 students, will be able to use stable 5 V and -5 V supplies required for their op-amp analysis circuits.

Blue2
Aaron Gavin

Power Distribution Subsystem Report

Revision 1
4/30/2022

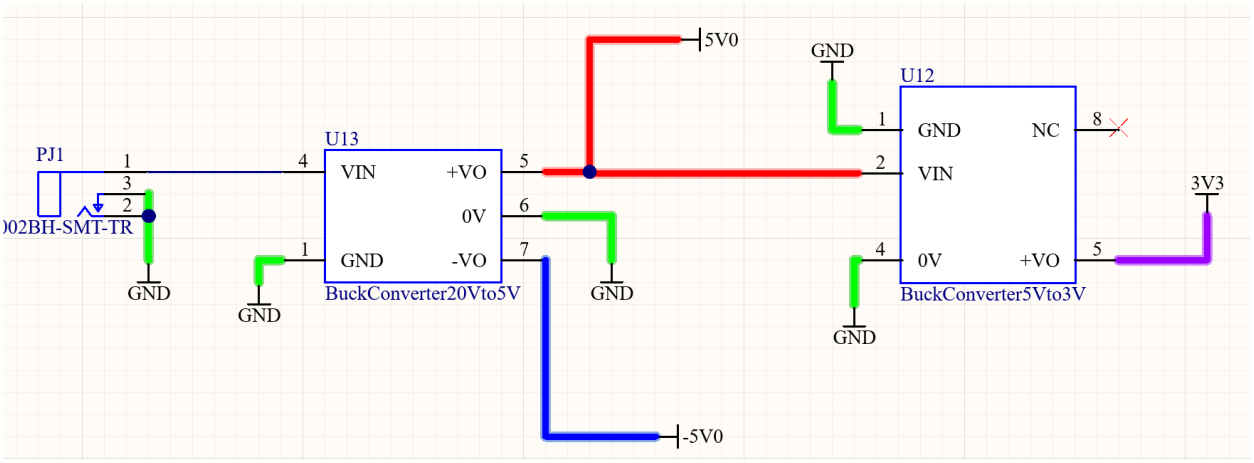
Subsystem Introduction

The Power Distribution subsystem supplies the necessary voltage values to all of the op-amps, IC chips, and the -5 V supply for the DC Power Supplies subsystem.

Subsystem Details

The input to the Power Distribution subsystem is 20 V from the wall wart power supply. The outputs are -5 V, 5 V, and 3.3 V. The 2.5 mm female jack in the design allows connection for the corresponding 2.5 mm male jack on the wall wart power supply. The first buck converter converts the 20 V input into 5 V and -5 V outputs. Both the 5 V and -5 V are sent on an output wire to any other subsystems that require those voltages. One line of the 5 V output from the first buck converter is fed into another buck converter which converts its 5 V input into a 3.3 V output. The 3.3 V is then sent on an output wire to any other subsystems that require that voltage.

Figure 34: Schematic of the Power Distribution Subsystem



Subsystem Validation

The Power Distribution subsystem was validated by measuring the output voltage of the 5 V, -5 V, and 3.3 V outputs with the bench equipment and checking whether the correct voltage values were being produced. After performing these tests, the 5 V, -5 V, and 3.3 V outputs were shown to produce the correct voltage values.

In Figure 13 below, the voltage value of 20.18 V was measured for the 20 V input is shown. In Figure 14 below, 3.40 V was measured for the 3.3 V output, 4.79 V was measured for the 5 V output, and -4.93 V was measured for the -5 V output. The output voltage values are not exactly equivalent to the desired voltages, but after discussing with our project sponsor and professor, Dr. Lusher, we decided that these values were sufficient.

Figure 35: Validation of the Inputs to the Power Distribution Subsystem

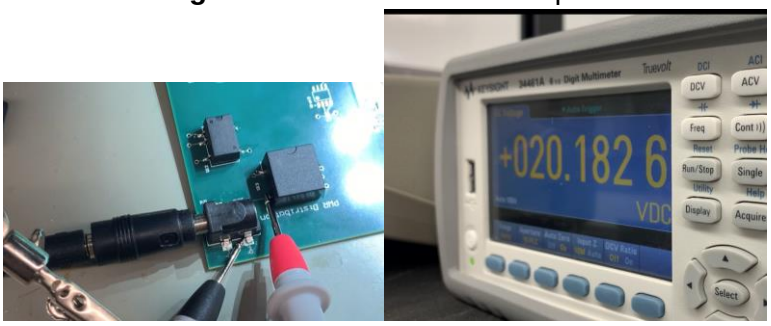
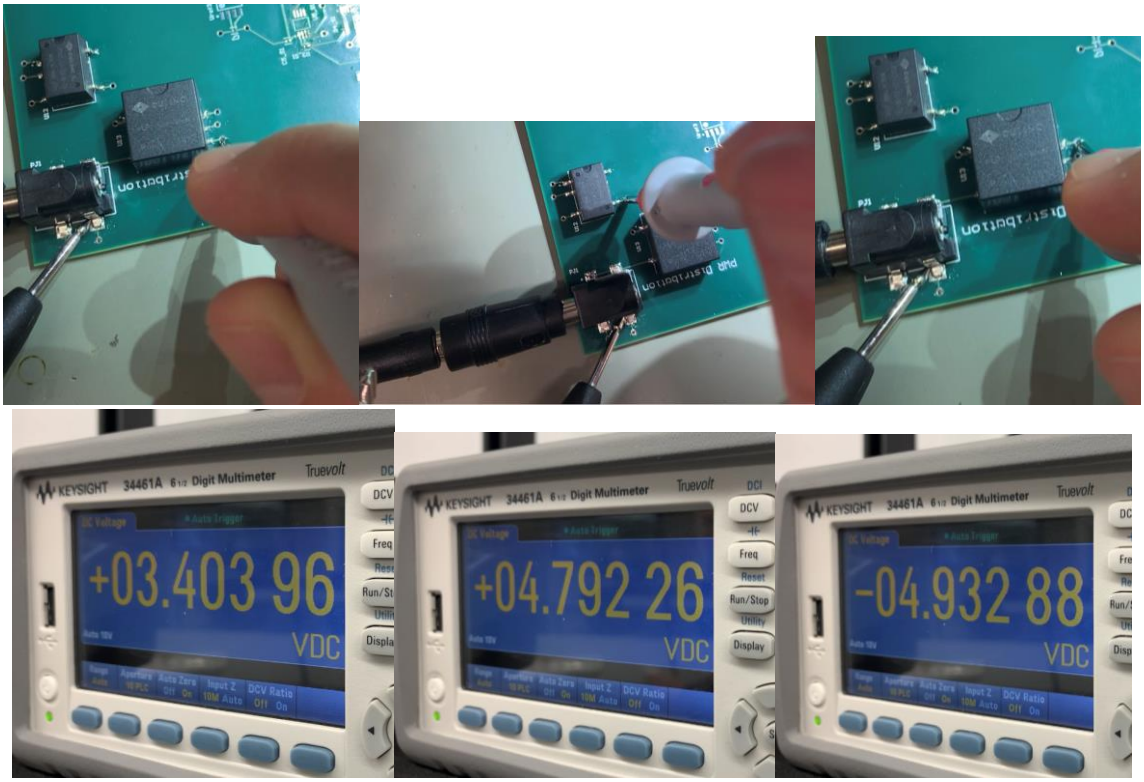


Figure 36: Validation of the Outputs from the Power Distribution Subsystem



Subsystem Conclusion

The DC Power Supplies subsystem was shown to be operating properly in its implementation on the PCB. All of the components on the PCB that required these voltages successfully received power and enabled the use of those components for all other subsystems.

Blue2

Aaron Gavin

Waveform Generator Subsystem Report

Revision 1
4/30/2022

Subsystem Introduction

The Waveform Generator subsystem provides ECEN 215 students with all of the necessary voltage waveforms they will need for the labs.

Subsystem Details

Most of the development of the Waveform Generator subsystem was the C code for the PIC32 microcontroller which would communicate with the MCP4821 digital-to-analog converter and generate analog waveforms. The Waveform Generator only generates the waveforms needed to complete all of the ECEN 215 labs. This includes various Sine waves, triangle waves, square waves, and pseudo-DC signal. The C code is separated into functions (Sine, square, triangle, and DC) and generates 1440 discrete numerical values that resemble the desired waveform. The C code only generates one period of each waveform as repeating periods will occur as long as the user has selected that waveform to generate.

Figure 37: Sine Wave C Code

```c
#include <stdlib.h>
#include <math.h>
#include <stdio.h>

float generateSinewave(float frequency, int samplingRate, float amplitude, float durationOfExecution) {
    // Generates one period of a Sine wave given frequency and amplitude
    int xCounter = 0; // Counter for generating X values
    float period = 1/frequency; // Period is in Seconds
    int dataPoints = samplingRate*durationOfExecution; // Number of data points generated

    float voltageArray[dataPoints][2];

    // Sine Wave Generation
    float time = 0;
    float radians = 0;
    float voltage = 0;
    for (int j = 0; j < dataPoints; j++) {
        time = period * (1/(float)dataPoints) * xCounter; // Calculate time values
        radians = time * 2 * M_PI / period; // Calculate radian values for each x value
        voltage = amplitude * sin(radians); // Modify Y values by amplitude factor
        voltageArray[j][0] = time;
        voltageArray[j][1] = voltage;

        return voltageArray[j][1];
        xCounter++;
    }
}
```

Figure 38: Square Wave C Code

```c
#include <stdlib.h>
#include <math.h>
#include <stdio.h>

float generateSquarewave(float frequency, int samplingRate, float amplitude, float durationOfExecution, int offset){
    // Generates one period of a Square wave given frequency and amplitude
    int xCounter = 0; // Counter for generating X values
    float period = 1/frequency; // Period is in Seconds
    int dataPoints = samplingRate*durationOfExecution; // Number of data points generated

    float voltageArray[dataPoints][2];

    // Square Wave Generation
    float voltage = 0;
    float time = 0;
    for (int j = 0; j < dataPoints; j++) {
        time = period * (1/(float)dataPoints) * xCounter; // Calculate x values
        if (time<(period/2)) {
            voltage = amplitude + offset;
        } else {
            voltage = -amplitude + offset;
        }
        voltageArray[j][0] = time;
        voltageArray[j][1] = voltage;

        return voltageArray[j][1];
        xCounter++;;
    }
}
```

Figure 39: Triangle Wave C Code

```c
#include <stdlib.h>
#include <math.h>
#include <stdio.h>

float generateTrianglewave(float frequency, int samplingRate, float amplitude, float durationOfExecution){
    // Generates one period of a Triangle wave given frequency and amplitude
    int xCounter = 0; // Counter for generating X values
    float period = 1/frequency; // Period is in Seconds
    int dataPoints = samplingRate*durationOfExecution; // Number of data points generated

    float voltageArray[dataPoints][2];

    // Triangle Wave Generation
    float voltage = 0;
    float time = 0;
    for (int j = 0; j < dataPoints; j++) {
        time = period * (1/(float)dataPoints) * xCounter; // Calculate x values

        if (time<(period/4)) {
            voltage = 2*(time/period);
        } else if (time<((3*period)/4)) {
            voltage = (amplitude-(2*(time/period)))+amplitude;
        } else {
            voltage = (2*(time/period))-(4*amplitude);
        }

        voltageArray[j][0] = time;
        voltageArray[j][1] = voltage;

        return voltageArray[j][1];
        xCounter++;
    }
}
```

Figure 40: DC Wave C Code

```c
#include <stdlib.h>
#include <math.h>
#include <stdio.h>

float generateDCSignal(int samplingRate, float amplitude, float durationOfExecution) {
    // Generates one period of a Sine wave given frequency and amplitude
    int xCounter = 0; // Counter for generating X values
    float frequency = samplingRate; // For DC signal frequency is equal to the samplin
    float period = 1/frequency; // Period is in Seconds
    int dataPoints = samplingRate*durationOfExecution; // Number of data points genera

    float voltageArray[dataPoints][2];

    // DC Wave Generation
    float time = 0;
    float voltage = 0;
    for (int j = 0; j < dataPoints; j++) {
        time = period * (1/(float)dataPoints) * xCounter; // Calculate time values
        voltage = amplitude; // Modify Y values by amplitude factor
        voltageArray[j][0] = time;
        voltageArray[j][1] = voltage;

        return voltageArray[j][1];
        xCounter++;
    }
}
```

Subsystem Validation

Validation for the waveform generator on the bench equipment was not possible due to the fact that we did not have the entire system working. Due to bottlenecks with the ESP32 code, the waveform selection and generation was never invoked in the version of the code that was loaded onto the PIC32. The waveforms shown below were generated using the same code as shown above but with a C compiler on a computer. A few tweaks were made to the code since the waveform generator was validated in ECEN 403. Most notably, each waveform generation function generates one period of their specific waveform. That period is saved in a two-dimensional array with one dimension as time and the other as voltage. Since values are passed along peripheral interfaces synchronously at the clock rate of 1440 Hz, the sampling rate of the digital-to-analog converter is set to 1440 Hz and each voltage value of the waveform is transmitted from the PIC32 to the digital-to-analog converter which then converts the digital integer counts into an analog signal. Each period of a waveform is repeated indefinitely until the user presses the stop button on the app, so only one waveform can be generated at a time. The possible waveforms that can be generated are limited to only the waveforms that are required to complete the ECEN 215 labs. Below are the graphs of the discrete data points generated for a Sine wave, a square wave, a triangle wave, and a DC signal.

Figure 41: Sine Wave Validation

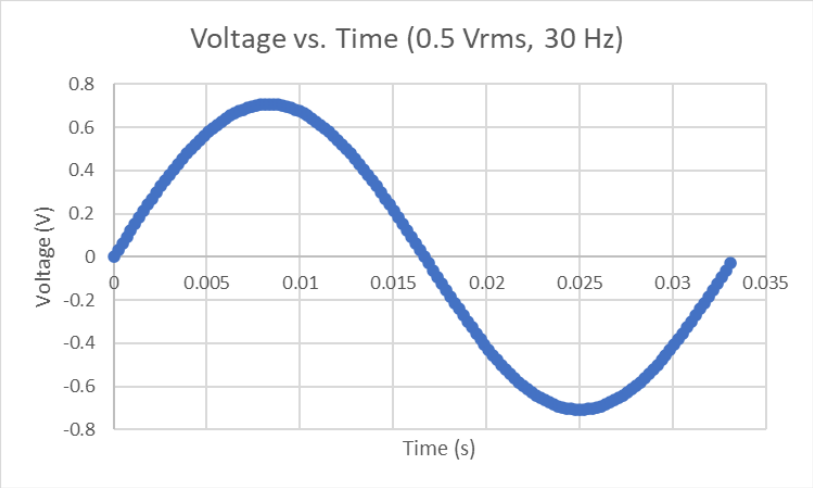


Figure 42: Square Wave Validation

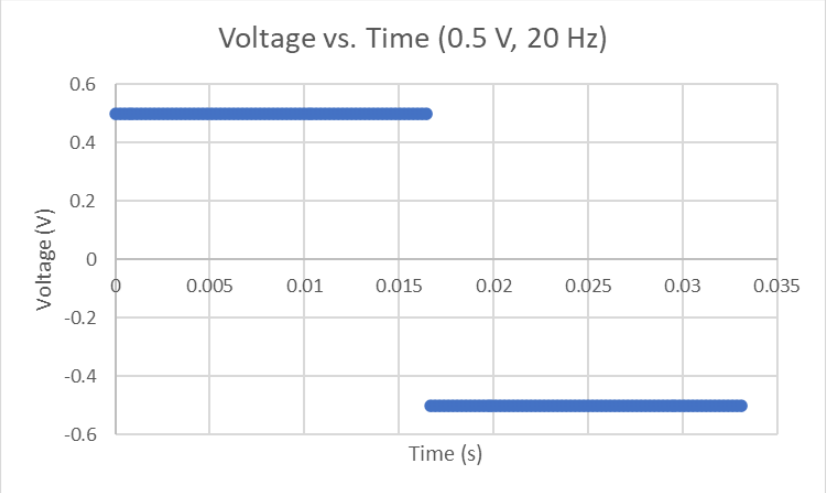


Figure 43: Triangle Wave Validation

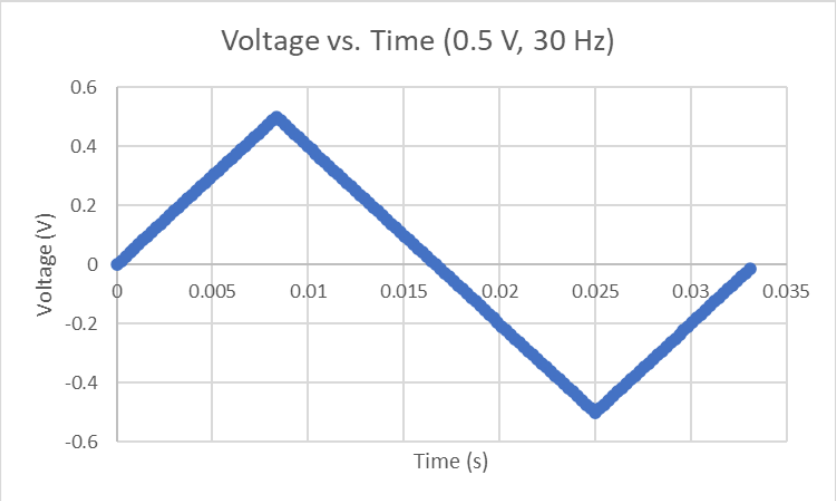
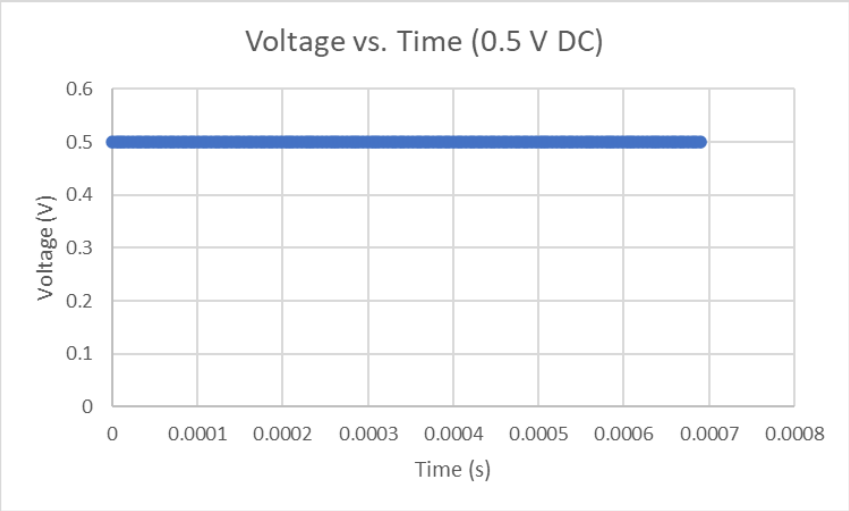


Figure 44: DC Wave Validation



Subsystem Conclusion

The waveform generator was never tested with the bench oscilloscope as planned. However, the tests on the software side proved that the waveform generator code produced discrete data points at the sampling frequency in the shape of each desired waveform. Further testing upon implementation onto the hardware would need to be performed to fully validate the subsystem for its intended purpose.

Bowen Mei

ESP32 Subsystem Report

Revision 1
4/30/2022

Subsystem Introduction

The ESP32 subsystem is designed to act as a bridge connection between the phone application and the PIC32.

Subsystem Details

The subsystem initially was located within the board where most of the hardware was. Due to some errors with flashing the code onto the ESP32 chip on the board. After cutting the trace of the connection between the ESP32 and the PIC32, we added some wire connectors that allowed us to use an ESP32 dev board.

Figure 45: ESP32 UART Code

```
    while (1) {
        if(writeSPP){
            int len = uart_read_bytes(ECHO_UART_PORT_NUM, write_data, 1024-1, 20/portTICK_PERIOD_MS);
            sentToPIC = true;
            receiveFromPIC = false;
            writeSPP = false;
            doNothing = false;
            ESP_LOGI(SPP_TAG, "Read from UART");
        }

        else if(readSPP){
            uart_write_bytes(ECHO_UART_PORT_NUM, read_data, read_data_length);
            receiveFromPIC = true;
            sentToPIC = false;
            readSPP = false;
            doNothing = false;
            ESP_LOGI(SPP_TAG, "Write to UART");
        }
        else if(doNothing){
            //uart_write_bytes(ECHO_UART_PORT_NUM, read_data, read_data_length);
            //ESP_LOGI(SPP_TAG, "DO NOTHING");
            vTaskDelay(1000/portTICK_PERIOD_MS);
        }
        else{
            ESP_LOGI(SPP_TAG, "DO NOTHING");
            doNothing = true;
            vTaskDelay(1000/portTICK_PERIOD_MS);
        }
    }
}
```

The ESP32 we are using requires two types of connections. The first connection is a UART bridge between the ESP32 and the PIC32. The second connection is the Bluetooth interface between the ESP32 and the android application.

Figure 46: ESP32 Bluetooth connection code

```
#if (SPP_SHOW_MODE == SPP_SHOW_DATA)
            ESP_LOGI(SPP_TAG, "ESP_SPP_DATA_IND_EVT len=%d handle=%d",
                     param->data_ind.len, param->data_ind.handle);

        if(((param->data_ind.len) == 1) || (param->data_ind.len) == 2){
            esp_log_buffer_char("",param->data_ind.data,param->data_ind.len);
            if(receiveFromPIC){
                readSPP = true;
                read_data = param->data_ind.data;
                read_data_length = param->data_ind.len;
            }

        }
        //esp_log_buffer_hex("",param->data_ind.data,param->data_ind.len);

        if((param->data_ind.len) == 3){
            read_data = param->data_ind.data;
            read_data_length = param->data_ind.len;
            if(!sentToPIC){
                xTaskAbortDelay(xHandle);
                readSPP = true;
                writeSPP = true;
                if(write_data != NULL)
                    esp_spp_write(param->write.handle, sizeof write_data, write_data);
            }
            else if(!receiveFromPIC){
                xTaskAbortDelay(xHandle);
                writeSPP = true;
                if(write_data != NULL)
                    esp_spp_write(param->write.handle, sizeof write_data, write_data);
            }
        }
```

The ESP32 code used a Bluetooth SPP receiver example as its core. By adding a sending SPP command each time the receiver was sent a message, the Bluetooth connection was created and was able to send/receive information. In addition to that, the UART was created by adding a simultaneous background task that ran in parallel to the Bluetooth connection. It pauses itself when the UART detects no transmission in the Bluetooth connection, so it does not trigger a reset of the ESP32, which happens when a task is done too many times without a response.

Subsystem Validation

The subsystem was tested in conjunction with the android application. This was done by placing test data inside the ESP32 and the android application and checking if there was a Bluetooth connection and data transmission between the two. The UART was also tested with a COM port connected to a PC. The UART would send data from the ESP32 and output the data sent to the PC using putty.

Subsystem Results

The subsystem worked in some capacity with the Bluetooth connection. The issues can be found in the Android application section of the report. The UART connection was tested but had noise issues due to how the Bluetooth connection was set up. The constant transmission of the

Bluetooth meant that the signal that was being transmitted by the UART was a combination of messages.

Subsystem Conclusion

The subsystem had some significant flaws stemming from implementation of bluetooth transmission without proper scheduling and timing. I could have used Arduino to implement the ESP32, but I already had significantly more experience with VS code

Blue2
Bowen Mei

Android Application Subsystem Report

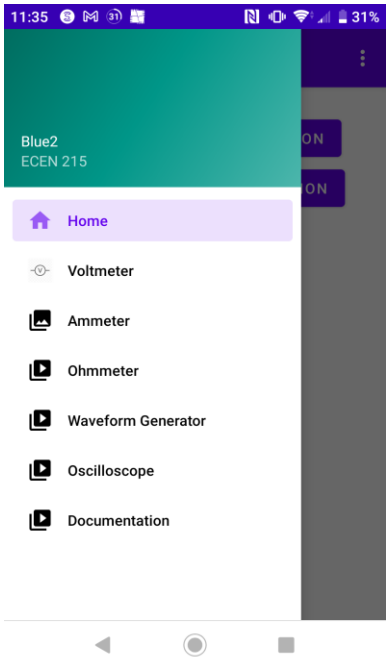
Revision 1
4/30/2022

Subsystem Introduction

The android application subsystem is designed to display the data gathered from all the hardware that is located on the board. It can be started up by any android device and will have different sections for each of the hardware subsystems that can be accessed on the sidebar of the application.

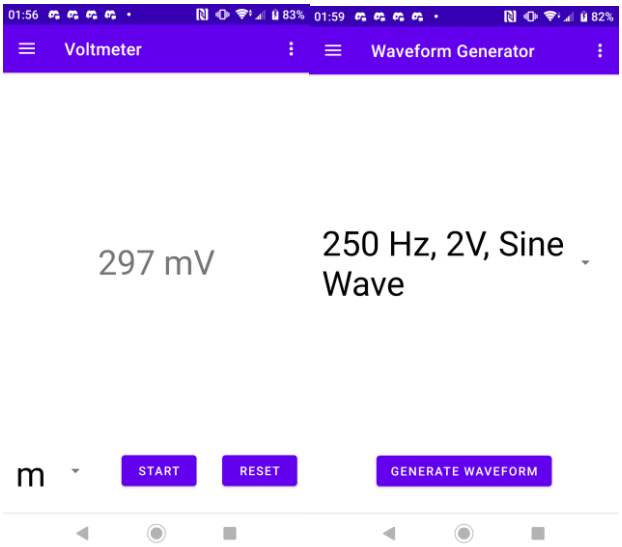
Subsystem Details

Figure 47: Application Main Menu



The subsystem will use a Bluetooth connection to interface with the ESP32 hardware. It will both be able to send and receive values using the connection. The Bluetooth connection is initiated at the boot stage of the application and does not The core of the application is the sidebar which holds all of the different landing pages for all of the different hardware subsystems.

Figure 48: Voltmeter and Waveform Generator Pages



In each of the subsystems, there is a display for the value given from the hardware, two buttons that allow the meter to start and reset, and a drop-down menu that allows the user to select between two different metrics of measurement (mV and V for example). For the oscilloscope page, there is a graphical interface that is updated every millisecond with a voltage value that is plotted and two buttons that allow for the oscilloscope to start and reset. Finally, for the waveform generator page, there is a large drop-down menu allowing the user to select a waveform that is used within the labs of ECEN 215 and two buttons allowing the user to send the choice of their waveform to the hardware while the other button allows the user to stop the waveform that was previously running on the hardware.

Subsystem Validation

Figure 49: Bluetooth Connection Code on Application

```
if(caseVal.equals("ammeter")) {
    if(startStream){
        write(Integer.toString( i: 100),tmpOutStream);
        startStream = false;
    }
    if (tmpInStream.available()!=0) {
        numBytes = tmpInStream.read(mmBuffer);
        String stringStream = new String(mmBuffer);
        String stringVal = stringStream.replaceAll( regex: "[^0-9]",  replacement: "");
        //mmInputStream.skip(numBytes);
        try {
            int e = Integer.parseInt(stringVal);
            Intent mIntent = new Intent( action: "Ammeter-message");
            mIntent.putExtra( name: "Current-message", stringVal);
            LocalBroadcastManager.getInstance(mContext).sendBroadcast(mIntent);
            write(Integer.toString( i: 100), tmpOutStream);
        }
        catch (NumberFormatException e){
            if(tmpInStream.available() != 0) {
                int trash = tmpInStream.read(oldBuffer);
            }
            Log.d(TAG,  msg: "Cleared buffer");
            write(Integer.toString( i: 100), tmpOutStream);
        }
    }
}
```

Validation of the system was completed with test data sent from the ESP32 and sending test data from the application to the ESP32. Initially, the ESP32 was sending ASCII values of test data. However, that was changed to an integer value that encoded the resultant values in the PIC32 and decoded by the application after the integer value is sent to the application.

Subsystem Results

The application was designed to display data that was gathered from the hardware's electrical testbench tools and send information to start, change, or stop a waveform generator. In that regard, the subsystem passed partially when test data was implemented inside the ESP32 and sent to the android application with Bluetooth. There were some roadblocks to the oscilloscope where the application would crash upon moving the graph as well as information being stuck within the Bluetooth data stream when receiving data from the ESP32.

Subsystem Conclusion

The android application works aside from certain edge cases, such as when you shut off a subsystem while a signal asking for the next value is sent or when you move the graph manually in the oscilloscope.

Things that could have been improved include, but are not limited to, the implementation of a better communication system and a better layout design for the elements of each page. The Bluetooth connection used Bluetooth Classic. Bluetooth Low Energy would be an interesting alternative that was initially considered but there were design challenges with its implementation. In the application code itself, the implementation of Bluetooth could be toggled instead of starting on application boot-up. With regards to layout design, it was based on the device that was being tested, which is different from some of the other devices that could be used.