Blue2

Jon Flores, Aaron Gavin, Bowen Mei

**Advice Report**

Revision 1

4/30/2022

**Introduction**

This report will cover what went right, what went wrong, and what needs to be done for the ECEN 215 Lab-kit project. It has been written as a guide for the next teams to work on this project in the future. The project did not make it to the intended result of a fully integrated and operable system but there are many useful designs, ideas, and tests that we find useful for the next group. The sections below walk through the electrical instrumentation as a whole. This will include the different functions on the device. It will also cover the good and the bad of the microcontrollers. Then will conclude with the information we gathered on the android application. We hope this information will be useful for the next team and will eliminate problems that we ran into.

**Electrical Instrumentation**

For the instrumentation system, we were able to have most of our designs fully functional. The major issue we ran into for all of our designs was the power supply to the op-amps. These were such a small component in all of our designs which were supposed to just buffer the outputs before going into the ADC. However, the small switch between the negative and positive supply to these components caused major issues with our entire device. We needed to raise the pins on these components, solder blue wire to the pin, and then connect it to the correct power supply via that it needed. There were a total of 10 op-amps that needed to be fixed so the time we would have saved by just checking this small issue would have gone a long way.

The ohmmeter design was working as we designed it. It was designed as a voltage divider that had a known resistor and a known voltage reference and then took in the voltage output. With these values, we were able to calculate what the user's resistor was with the voltage divider equation. You can find more details on this in the Ohmmeter subsystem report. However, when demoing our device, we were told that a better design would have been to use probes in place of the user's resistor so that we could measure the resistance of any location of the user's circuit. This would operate similar to a lab multimeter when checking for continuity or using the ohmmeter function.

For the oscilloscope, we knew this design was one of the more complex designs so it was hard to know exactly what would cause the issue when issues did arise. After our corrections, which you can find in the oscilloscope and voltmeter subsystem reports, we found that our design worked properly. However, when demoing, Professor Lusher pointed out that the noise we were getting in our outputs could have been caused by the changes that were made and the environment in which this was being tested. In the lab, there are lots of interferences that may have caused unwanted noise to our function. To fix this, we would have needed to rebuild our board with the changes that were made to eliminate exposure. We could have also added more capacitors to filter the outputs. The voltmeter uses this oscilloscope to measure the values. After corrections would have been made, this function would have been working perfectly. However, the PIC code not being able to test with our hardware caused some uncertainty on what would work and what would not when showing the output to the user.

For the ammeter, this design worked very well. We were able to apply a known voltage over a known resistor and found our outputs were accurate that way. Once we tested on more complex circuits as we would in the lab, our values started to vary when comparing them to our desired value. We were not able to test and see why this occurred, but since we were getting the outputs we were wanting while doing simple tests, we knew this function was working.

The DC power supplies function was working as intended. Since we only needed to provide the user with -5 V and +5 V, we were able to design it strictly so it produced those voltages. Our rocker switch when turned on showed that it was connected to the +5V and would control that. However, for the -5V supply, there was no change when we turned the switch on. To fix this issue, we directly connected the power distribution output to the negative supply jack. This caused that jack to always get -5 V.

The wave generator hardware was not tested because it required complete PIC32 code with peripheral communications. In the PIC32 code that I ran on a compiler on my computer, we were able to generate discrete data points at a certain sampling rate in the shape of a Sine wave, Square wave, or Triangle wave. Late in the semester, we looked through the lab manual more carefully and discovered that one of the labs required a DC signal of 0.5 V. The design of the DC Power Supplies was only capable of delivering -5 V and +5 V. To solve this problem, I wrote code to generate a pseudo-DC signal which outputs the same value of 0.5 V at the discrete sampling rate.

For the Power Distribution to the board design, we just took the input of the wall wart and converted it through two DC to DC converters which would supply the power to the board. The simple design had a huge impact on the functionality of our board. It is what allowed all of our functions to operate as planned. When we ran into issues with the op-amps, it caused the outputs of this function to drop which in turn caused problems for the entire board.

**Microcontrollers**

The PIC32 microcontroller’s main code is written in C. Utilizing a third-party microcontroller programmer, for example, the Pic Kit 3, is necessary to convert the easy-to-write C code into hex instructions that the microcontroller can run. In our design for the PCB, the spacing between the pins, or pitch, used to connect the microcontroller programmer was too small. During the design process, we failed to confirm that the pitch of the male pins on the programmer needed to be equal to the pitch of the female pins on the PCB. We remedied this by carefully soldering small wire extensions so that we could connect the programmer to the PCB. The functionality of the microcontroller’s main code was working as intended, however, I was not able to test the microcontroller’s code separately on the PCB to verify it on the hardware. A solution to this problem would be to add a console port to isolate the PIC32 and debug and test inputs and outputs from just the PIC32.

The ESP32 microcontroller had some issues regarding flashing and booting onto the chip located on the board. This could be due to some pinout or flashing issues, as the result given was similar to using a UART connection that was used for data transfer or debugging. We were able to solve this situation by soldering on some wires to allow for a development board to be used as testing instead of flashing code onto the chip on the board. Another solution would be to add a CP2012 USB to UART to flash the code onto the chip on the board. During the testing of the UART connection between a simulated device and the ESP32, there were issues with noise and transmission of messages that were longer than what was desired. This was due to not timing the transmission of the Bluetooth to coincide with the UART transmission. While the data transmission in UART gave values that we wanted, it also had lots of empty space and junk data. The implementation of the ESP32 code could also be simplified by using the Arduino IDE, instead of the IDF-ESP, as it is simpler to understand and implement functions. If planning to use the IDF-ESP, use one of the examples as a basis of the code to have a starting position as it is quite hard to write everything from scratch.

**Android Application**

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.