Killian Rush

Design Overview:

The goal of my design is to provide a way to remotely monitor temperature data from a radio beacon using an SDR. The board should be expandable for purposes other than weather in the case it needs to be modified, and the data should be able to be displayed in a GUI that provides a graph to show the user.

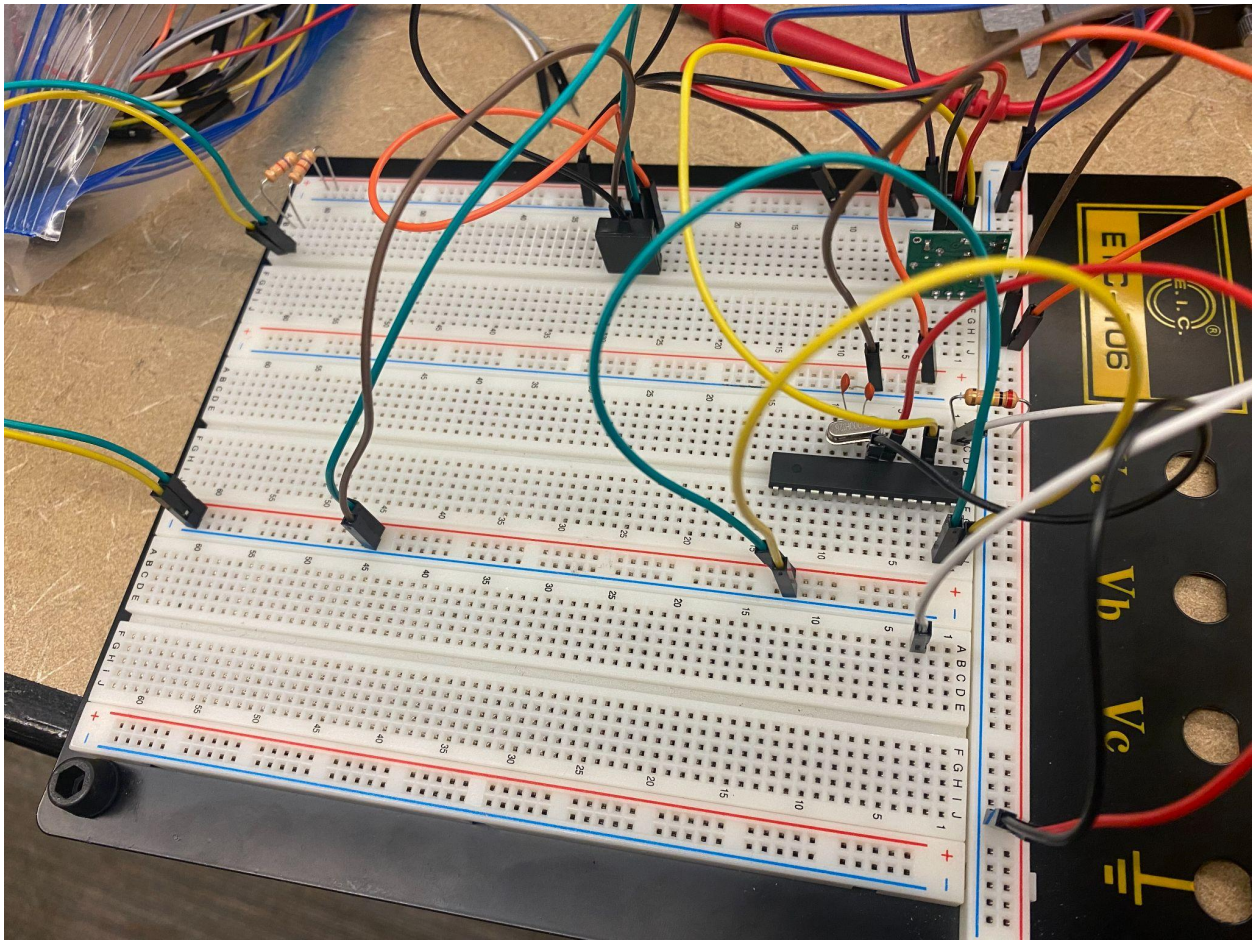
My original concept was to use ASK to transmit a UART signal that could then be transmitted and received using an SDR. I decided to use an I2C bus as this can be expanded in the future to communicate with things such as a Raspberry Pi. This way I can have a swappable and modular design and any changes would only need the Atmega to be reflashed, which is able to be done without removing the chip.

This project builds off of the PCB experience I had from the Bop It project as I must consider how to build a flexible PCB that is built in order to be expanded in the future. As such, a good amount of work was put into designing the PCB. Additionally, the software side of this project took a significant amount of work as I was not able to find code that worked with decoding UART signals coming from GNU radio, meaning I had to code one from scratch.

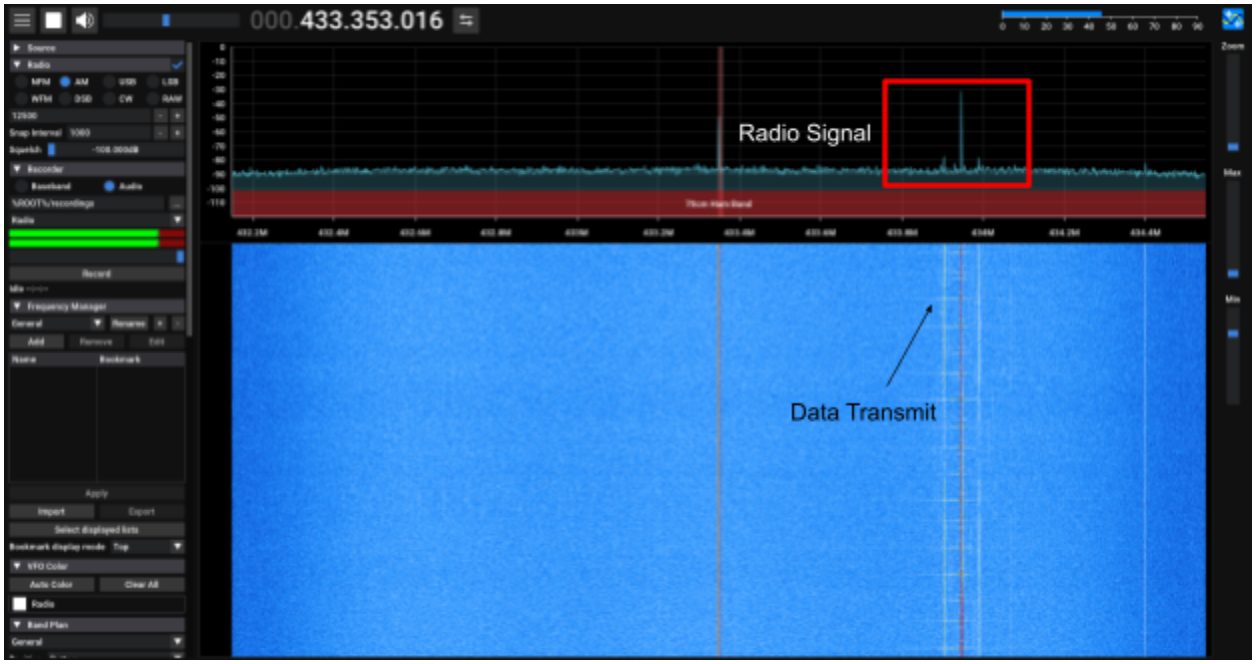
Preliminary Design:

My gameplan for this project was to first test the breadboard, then while it was being milled work on the software to receive the data. Once the breadboard was finished I could then continue developing the software with the new breadboard. Finally after all was done I could build the enclosure and put it all together.

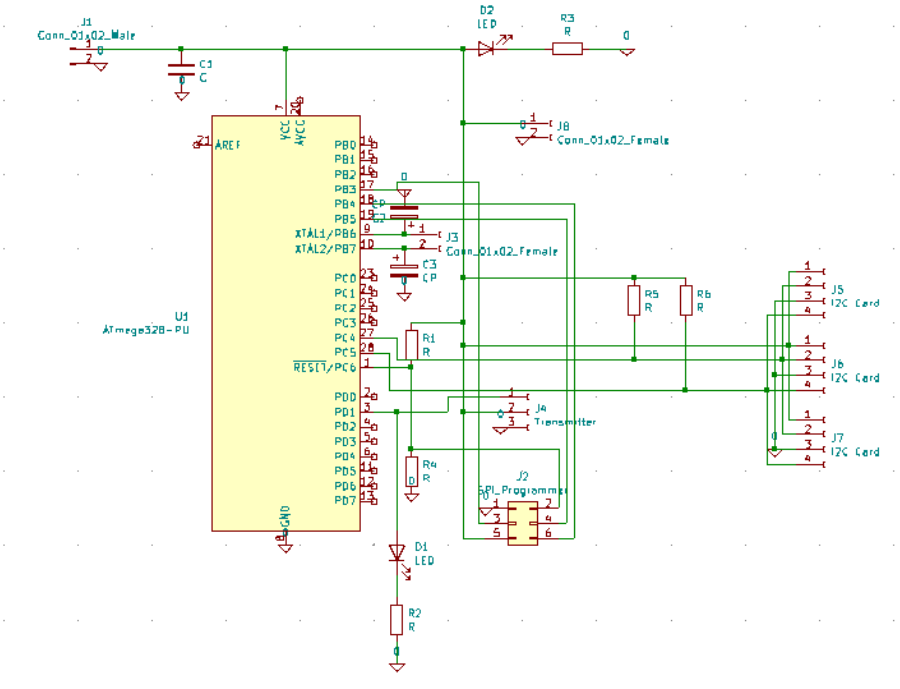
My first step was breadboarding. The actual design was pretty simple, and I took lessons learned from Hello World and the Bop It project to include things such as the 16MHz clock. One thing I did not realize at first was the need for pull down resistors for the I2C Bus.

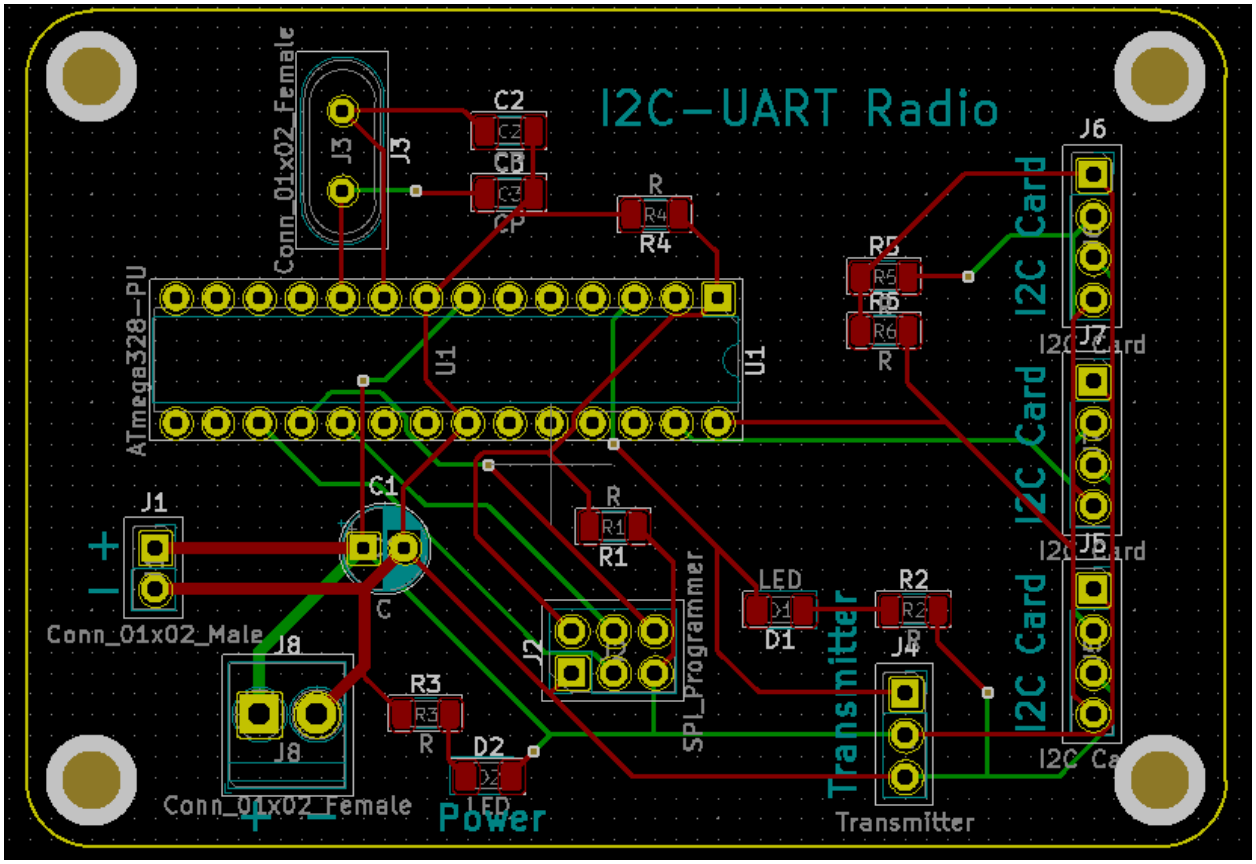


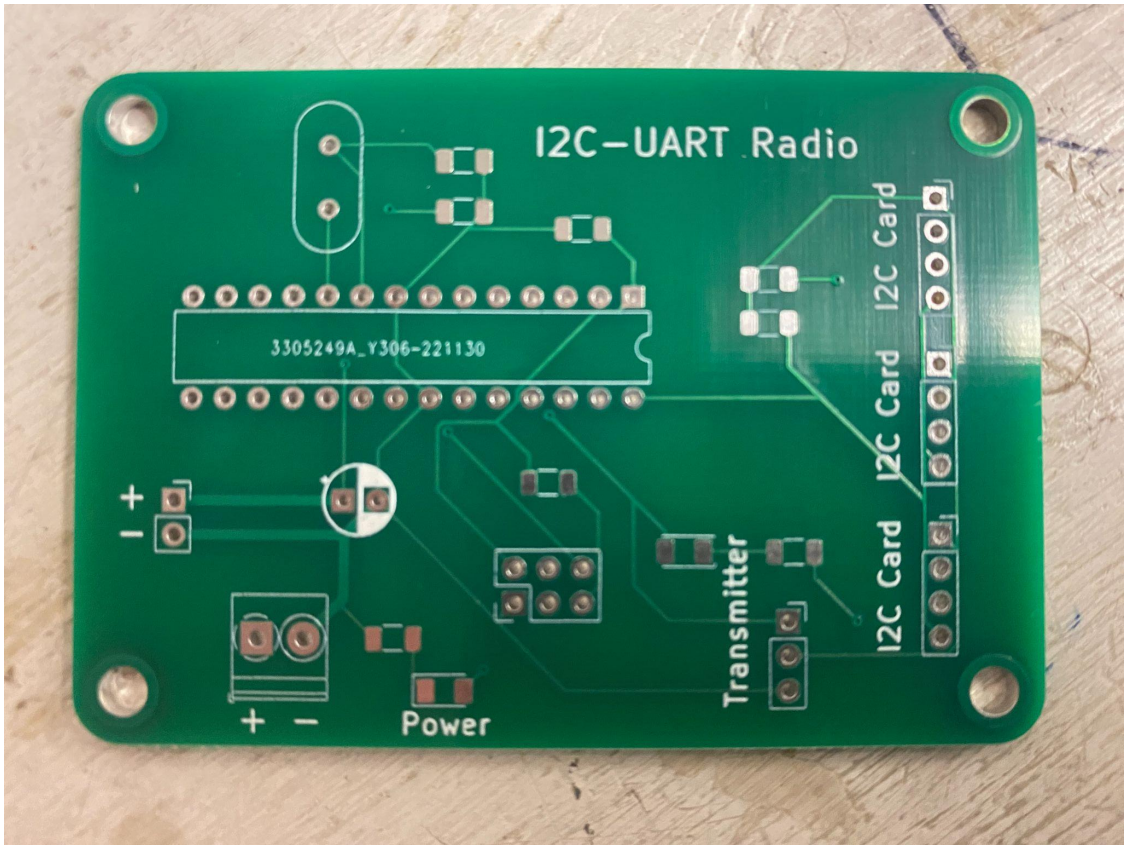
After the breadboard was done I could test to see if I got a signal from the ASK module. Sure enough, in SDR++ the ASK was outputting a signal similar to the one I was expecting of a UART signal.



I then used KiCad to build the PCB.

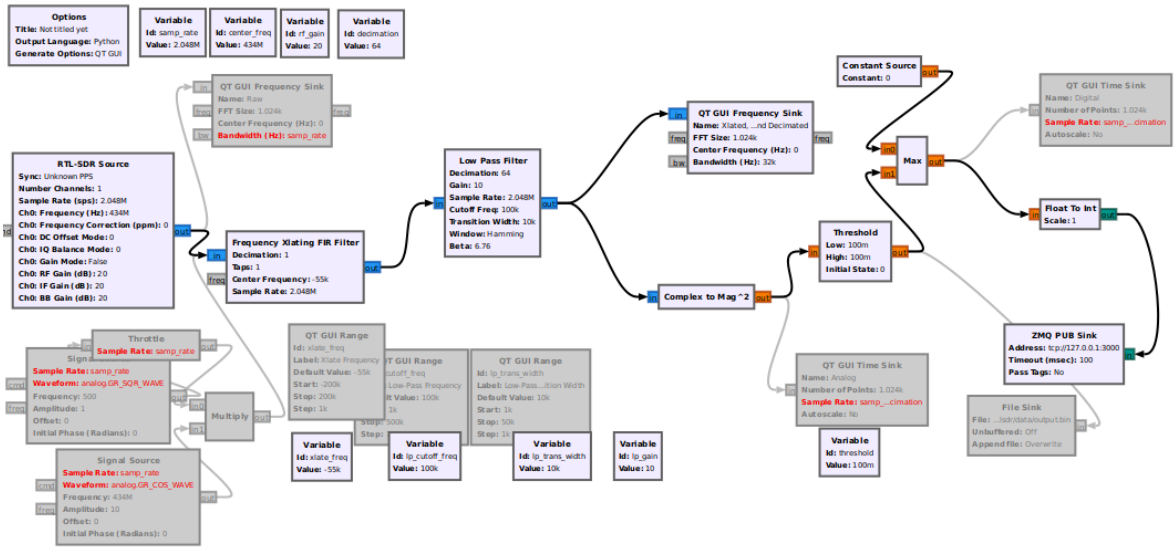



I2C-UART Radio
Conn_01x02_Female
J3
C2
C3
R
R4
ATmega328-PU
U1
R5
R6
J6
I2C Card
J1
C1
Conn_01x02_Male
R1
SPI_Programmer
LED
D1
R2
J4
Transmitter
J8
R3
D2
Power


I2C-UART Radio
3305249A_Y306-221130
I2C Card
Transmitter
Power

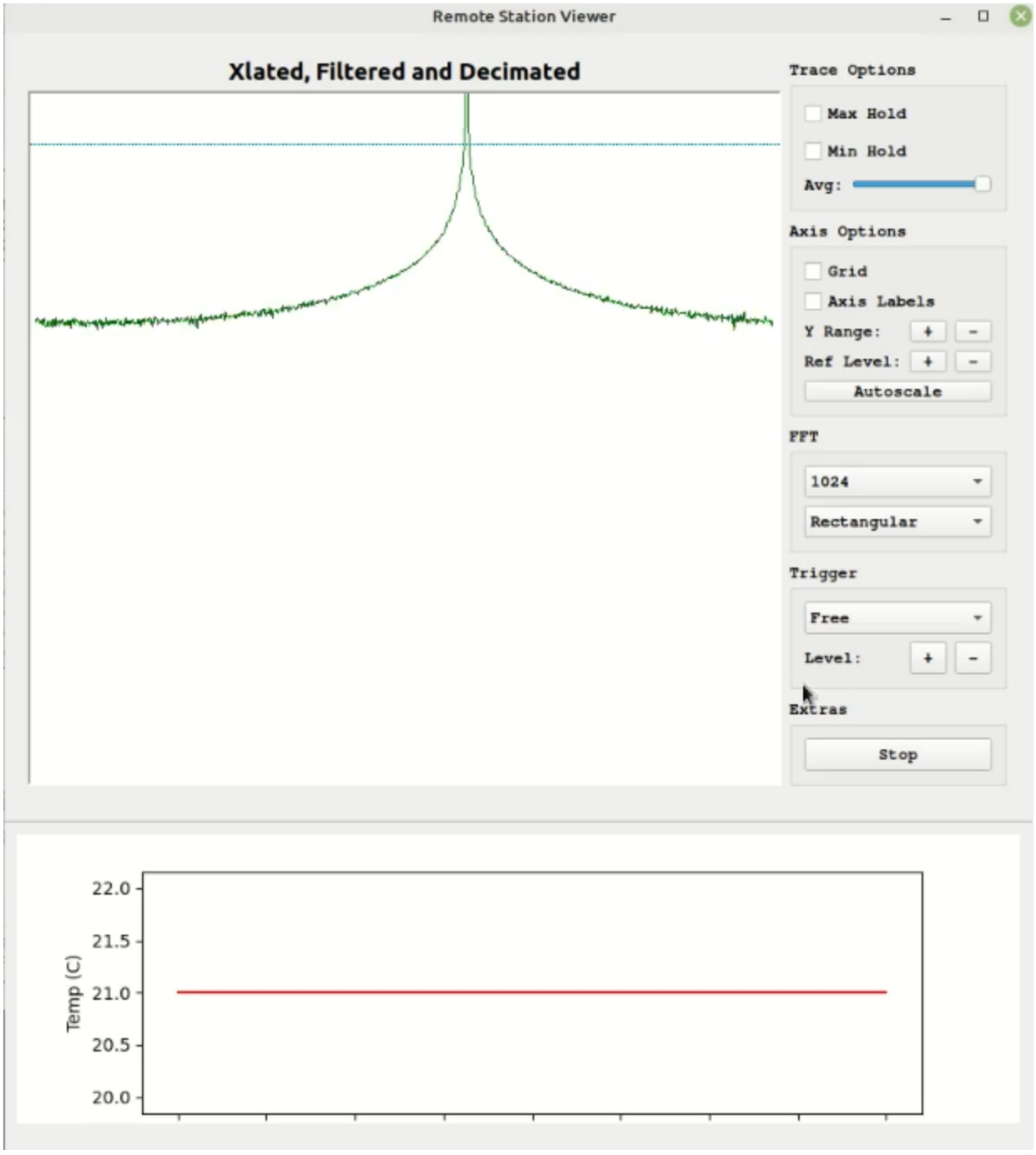
After creating the PCB, I worked on the UART receiver and decoder. This proved MUCH more challenging than expected. First, I had to figure out how to actually receive the signal. While one would think that you could just test if a signal was on or off, this was not the case. Because of sampling limits with computers and consumer SDRs, most data comes into GNU Radio as a numpy array, which means you must first process it in the frequency domain. Eventually, I combined some of my GNU code with some developed by Bodokaiser.

https://github.com/bodokaiser/arduino-rtlsdr

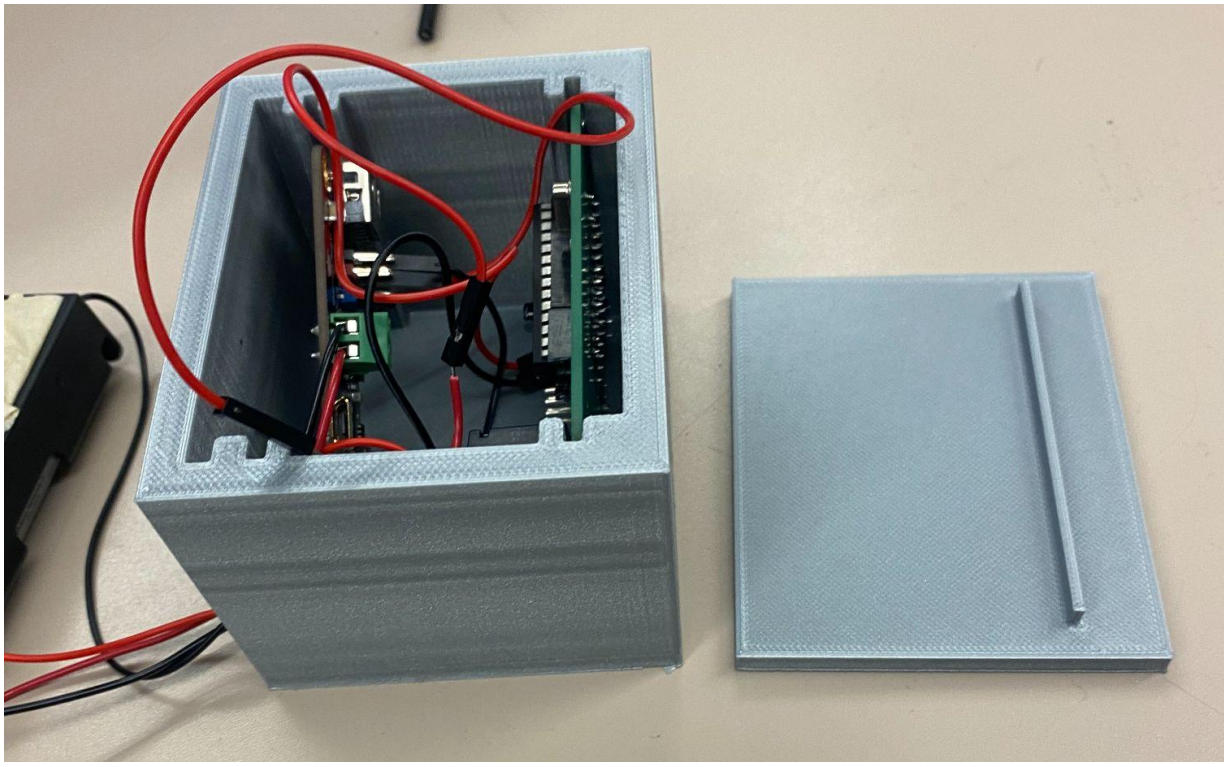


Next was actually taking this signal and decoding. Again for the aforementioned reasons this proved difficult due to the speed of data that was coming in. I had to do some research into sampling rate and processing large numpy arrays. Additionally, this data first needed to be sent through a ZMQ socket, requiring work since the buffer for this socket would often be too large to be processed, requiring some work. However, eventually I got it to process correctly.

Finally on the software side, I needed to build the GUI. This was actually pretty fun. I found a tool that was able to generate some skeleton code for basic GUIs from a template you design. After that, I just connect the GNU Radio interface and matplotlib interface of the logged temp data into the GUI through putting a widget in a widget. It was a bit clunky but after a bit of fiddling with length and width I got it to work.

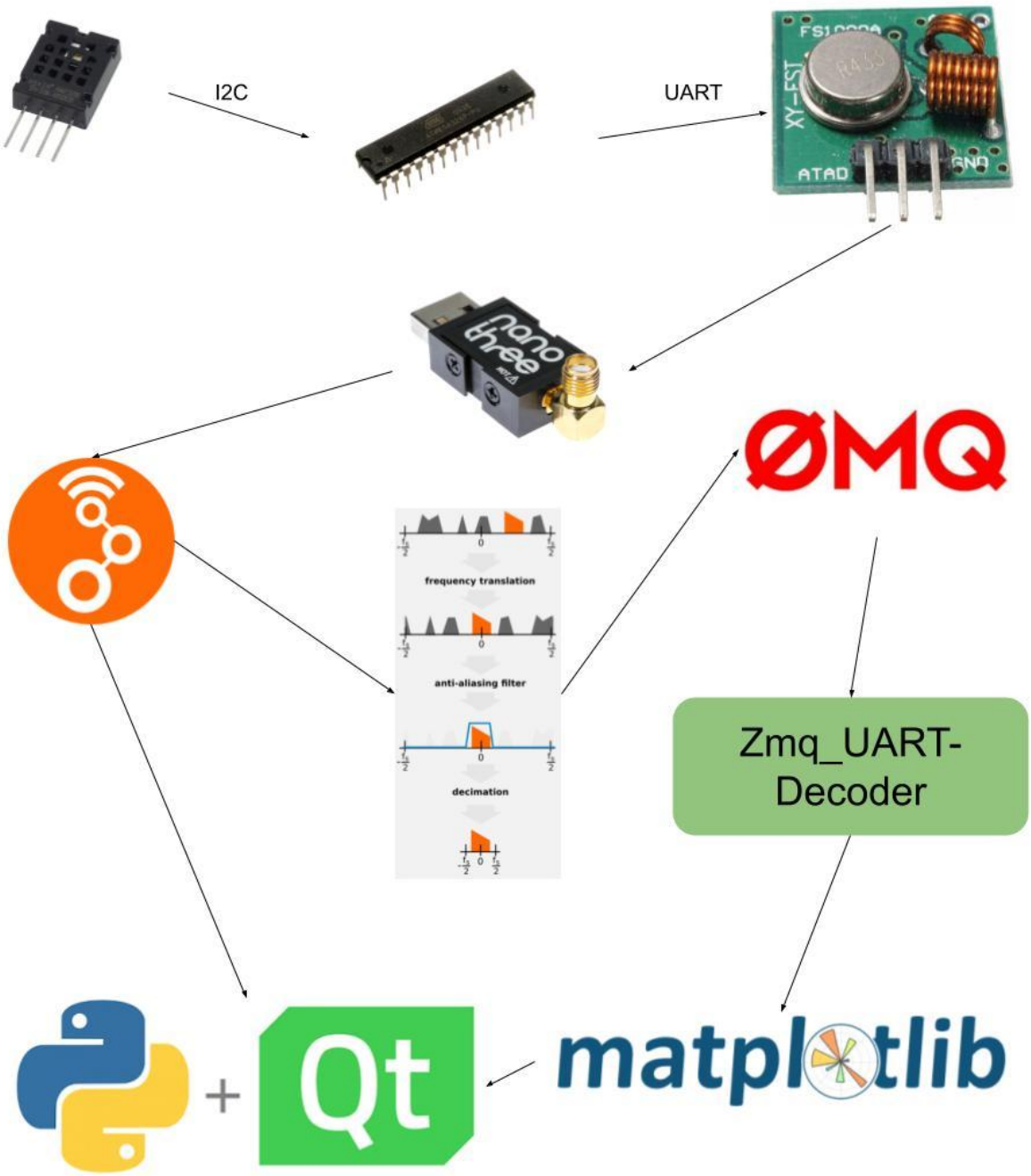


The last thing was to build a simple enclosure. I took inspiration from server racks, and wanted to make it modular so the pcb can be easily taken in and out.



Design Implementation:

A flowchart of the design can be found below.



The temperature is first pulled from the sensor to the Atmega using I2C. It is then sent using UART through the ASK module. It is received via the Nano SDR using GNU Radio. This generated a frequency graph which can be viewed through the GUI. It then passes through a

Frequency Xlating FIR Filter, which acts as a frequency isolator, and downsampled to a reasonable sample rate. This is then sent to my own python script through a ZMQ socket to be decoded and deciphered. This is then graphed in matplotlib and displayed on the GUI.

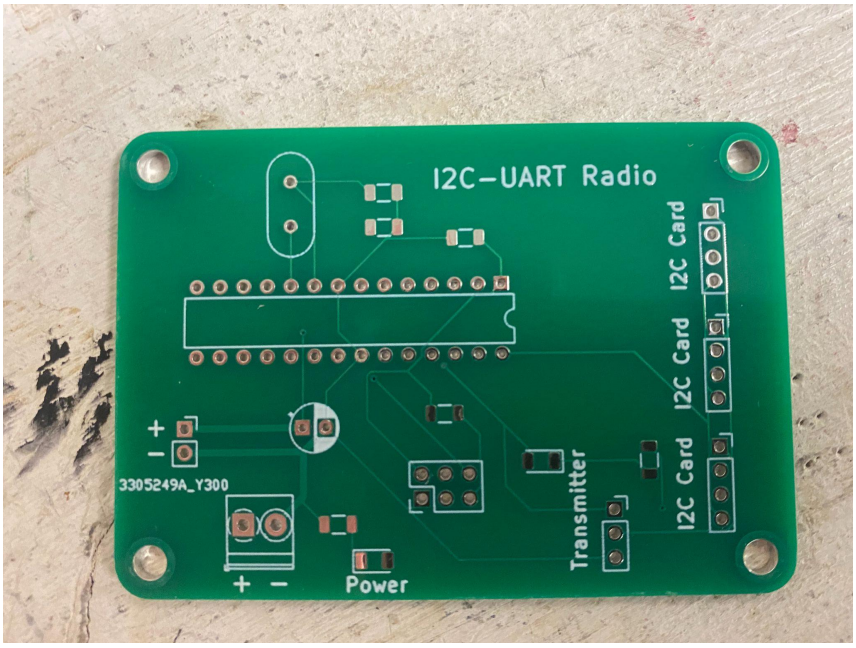
As for peripheral components, I wanted this to be solar powered. As such, a BMS with a small 3.7V Li-ion battery is included in my design, along with a solar panel. Any 12V solar panel will do however.

One challenge I came across was that ZMQ sockets where often slow, creating a bottleneck just before the encoder. I was able to fix this with decimation, but there is still some delay in the system from the ZMQ socket being so slow.

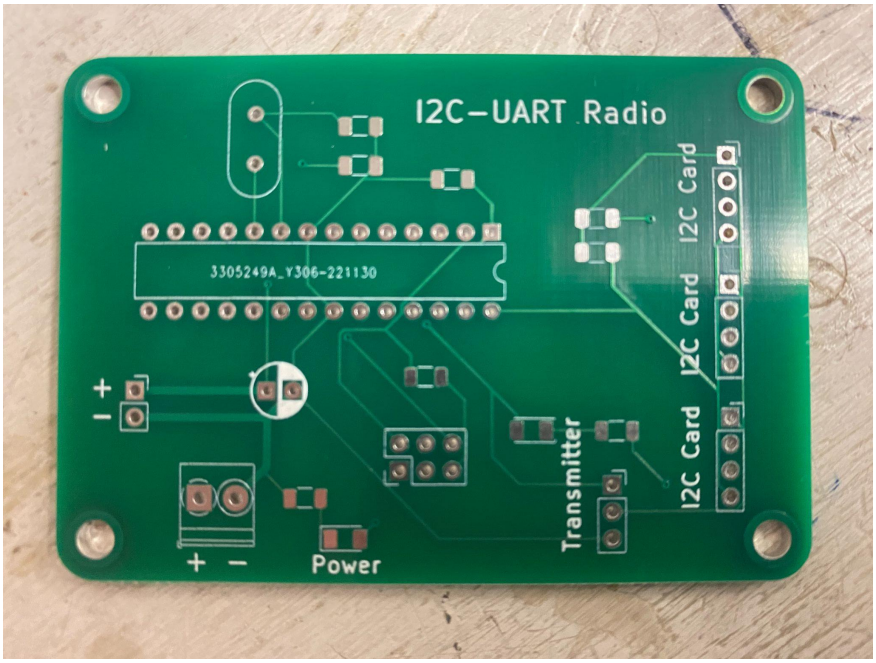
Testing Procedure:

My testing procedure was pretty simple. First, I would upload modified firmware to the arduino that transmitter a constant value, then test with that first. Then, I uploaded the temperature code to test the I2C bus. One problem I faced was that the first breadboard I made had a dud I2C bus due to the lack of pulldown resistors, so the board had a simple modification to make the I2C bus work.

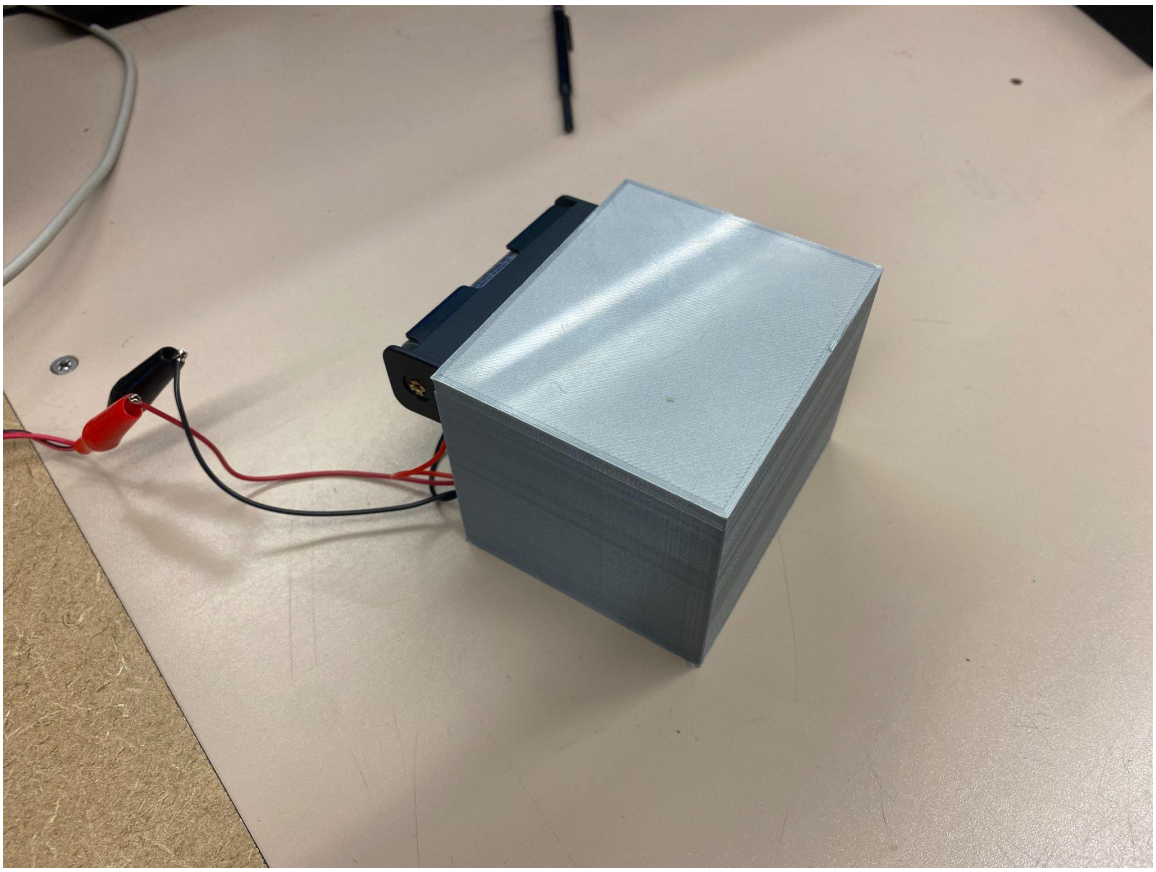
Old PCB



New PCB



Finally I printed the enclosure and put together the design.



Summary:

Overall, while this project was successful, going forward I would have liked to use a hardware decoder as I found the software decoding unnecessarily complex. However, I did this project to learn more about SDRs, and overall am very happy with the final product. I plan to take the temperature sensor out and utilize the board for other purposes as it makes for a great short range radio beacon.