WEEK 6:

Date: 09/30/2022

Total hours: 12

Description of design efforts:

Continued radio receiver code development.

The first improvements done on the receiver code were proper interpretations of the channel bit data. They were actually 11 bit values rather than the 10 bits stated online. I assume this is just due to the sign bit and that is why it was not included. After creating a packed struct, the bits are easily extracted and the ensuing data could be extracted properly mapped to the correct channel.

To improve the existing radio receiver code, I had to put bounds on the input data streaming in. There were periodic packets coming in with valid frame starts, but with different content than channel data. This resulted in every 50 or so packets being invalid data, creating a useless and choppy input signal. To begin, I put a bound on the length of a packet before it gets parsed. If it fell within the valid range, it can then be processed.

The prior output of the parsing algorithm shows the clear issue:

A picture containing text

Description automatically generated

Fig. 6.1. Original throttle channel data showing periodic data corruption

After properly filtering the packets to the proper data content, the data smoothly varies as expected.

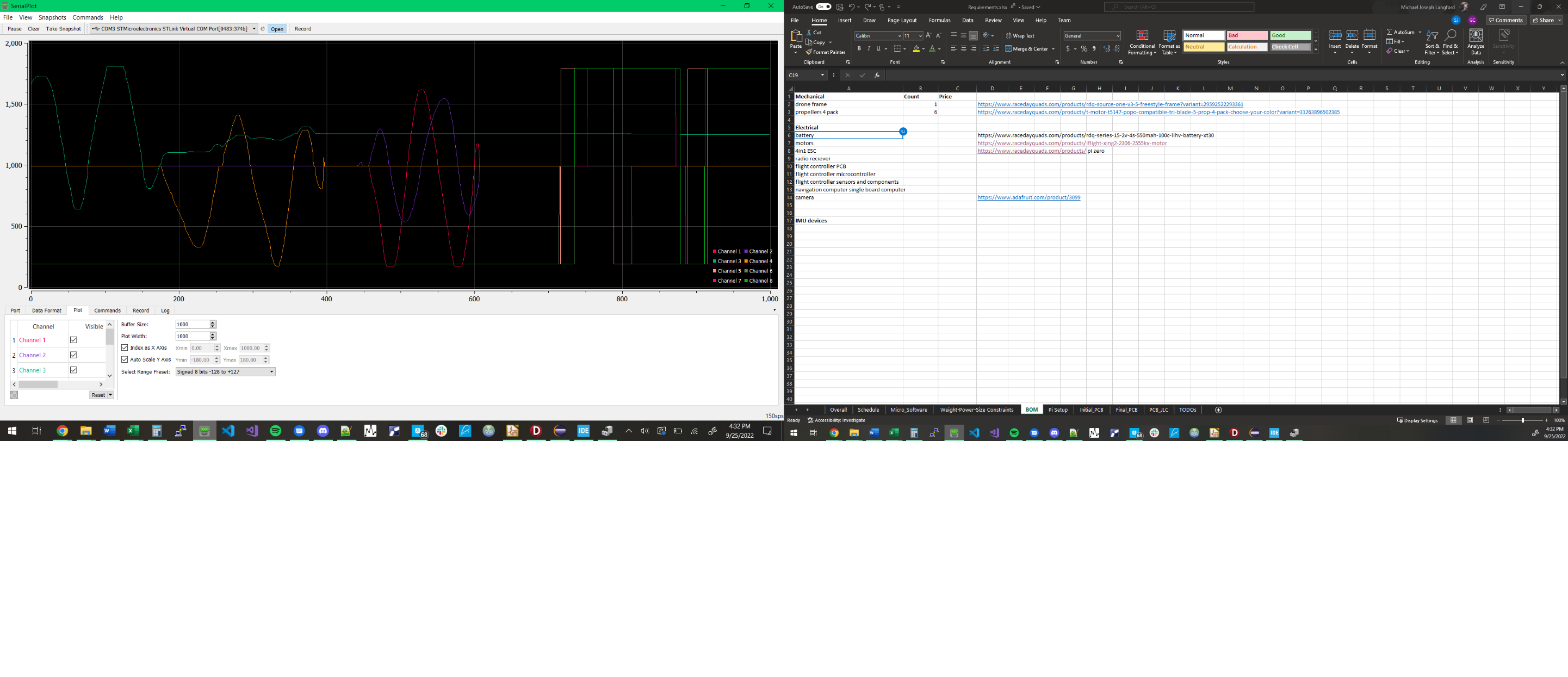


Fig. 6.2. Correct output of all 8 channels after parsing improved.

PWM development

PWM is a very common protocol for controlling brushless ESCs. They commonly use a simple protocol of a 20ms period, and a pulse width varying from 1 to 2ms depending on throttle value (0% = 1ms, 100% = 2ms). Santiago had already created the driver code to produce this signal, but we had to implement it with the correct timing.

Next, the calculations had to be done to create a PWM signal working off of Santiago’s code that had the correct pulse width and duty cycle. After refreshing myself about the internal timer operation, I created both code and an excel tool to calculate the PSC and ARR and CCMR values for the PWM driver.

Table 6.1. Table to calculate the registers.

|  |  |
| --- | --- |
| PWM PSC & ARR |  |
| period | 0.02 |
| Fclk | 1.60E+07 |
| freq | 50 |
| PSC | 15 |
| ARR | 19999.00 |
| freq\_out | 50 |
| period\_out | 0.02 |
| period\_ms | 20 |

After calculating the proper register values, the oscilloscope data displays the desired PWM signal. As can be seen in Figure 6.3., the pulse width is the desired 20ms.

A screenshot of a video game

Description automatically generated with medium confidence

Figure 6.3. PWM pulse measured on oscilloscope.

Radio and PWM integration

Now with both PWM and radio receiver working, I thought it would be a good demonstration, and also good work towards system integration, to make a radio channel value control the pulse width of the radio receiver. After changing some pins and channel values, and writing more configurable pin and register setup code, we were able to vary the pulse width by moving the sticks on the radio transmitter.

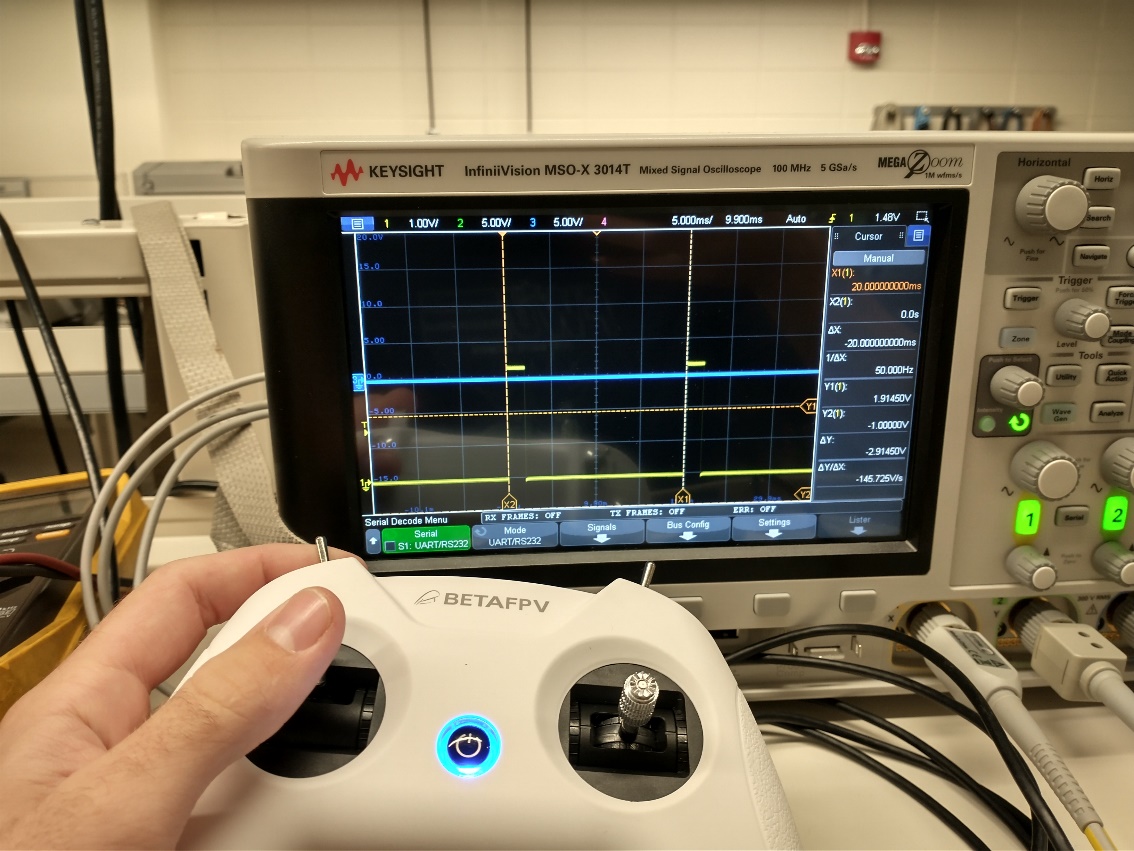
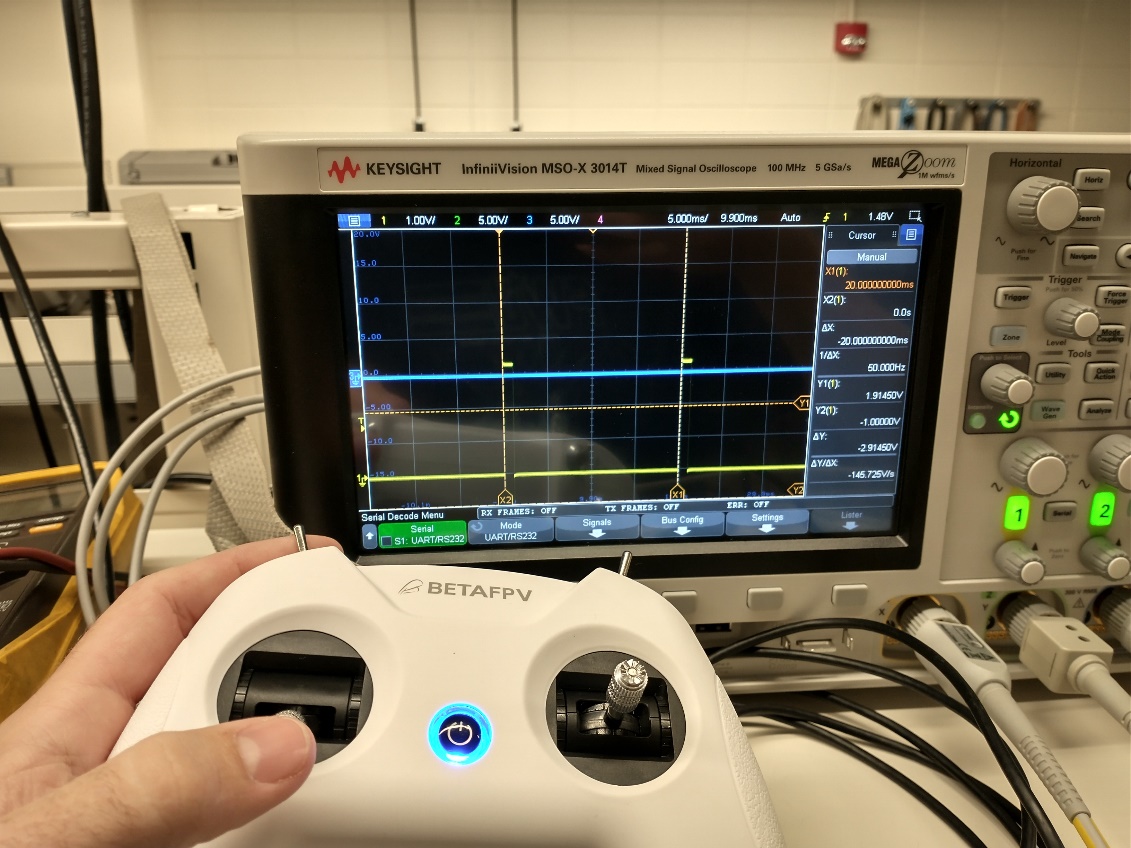


Fig. 6.4. Pulse width actively varied by radio receiver stick input.

[](https://www.youtube.com/embed/zwQ3Dvz2RXk?feature=oembed)

Schematic development and PCB development are also ongoing, although the details are somewhat tedious. Suffice it to say that footprints are being laid out in logical manner, more capacitors added, datasheets checked to ensure compliance with suggested implementation. Current schematic and PCB shown:

Diagram, schematic

Description automatically generated

Fig. 6.5. Current (in progress) schematic.

Graphical user interface

Description automatically generated

A screenshot of a computer

Description automatically generated with medium confidence

Fig. 6.6. PCB (currently), still getting the footprints in the right spots.