Sub-Gigahertz Arduino Shield

By: Alex Beck, Christopher Baldwin

Team 16

December 2020

Abstract

This document is the final report for the Sub-Gigahertz Arduino Shield project. The report will include the design considerations we made, our tests, and our results.

Table of Contents

1.0 Introduction

There is currently a gap in the market for cheap, low power, and long-range wireless communications. Our team planned to solve this issue by creating a Sub-Gigahertz Arduino Shield. An Arduino Shield is a device that is connected to an Arduino and gives it extra functionality. Our Arduino Shield will enable wireless communication with the Arduino. Once the Arduino receives the wireless signal, it will execute a pre-programmed response. The pre-programmed response can be changed by the consumers to better suit their own needs. Our team decided to base the project around Arduinos because they are easy-to-use tools that can be used by both beginners and experts. Since the Arduino is accessible to a wide range of consumers, our device will also be widely accessible. Our Arduino Shield will operate using wireless signals with frequencies below one gigahertz because they offer longer ranges and lower power consumption than higher frequency wireless signals. Alex designed and soldered the PCB while Christopher created the code for the project.

1.1 High Level Requirements

- The Arduino Shield must be able to convert a 7-17 DC input voltage into 5 volts for the Arduino.
- The Arduino must be able to send and receive wireless signals from a remote and execute a pre-programed action upon receiving a signal from the remote.

2.0 Design

Our Arduino Shield has five main parts. It has a 5-volt buck converter, a 3.3-volt buck converter, an MCU, six load switches, and an antenna. The 5-volt buck converter supplies power to the Arduino. The 3.3-volt buck converter supplies power to the MCU. The MCU sends and receives wireless signals using the antenna. The load switches send power to other devices. The Arduino Shield will also include two LEDs for debugging purposes. The block diagram for our Arduino Shield is shown in Figure 1.

In this project we chose to use the Sub-1Gigahertz protocol from Texas Instruments rather than Bluetooth, Wifi, ZigBee, 6LoWPAN, and other wireless protocols. We chose to use the Sub-1 Gigahertz protocol due to it having lower noise from BLE/Zigbee/WIFI devices and due to its ability to penetrate and bend around obstacles. The energy of the transmission can be described as follows.

$$P_R \propto P_T / d^2 f^2$$

Where P_R is the power at the receiver, P_T is the power at the transmitter, d is the distance between transmitter and receiver, and f is the frequency of transmission. This means we have longer range for the same power or lower power for the same range. This combined with the reduced noise makes Sub-1 Gigahertz satisfy our requirements.

In this project we chose to use load switches instead of relays or discrete transistors due to their small size, low parasitic power draw, and quick output discharge. This means that the user could use a PWM signal with their higher voltage output.

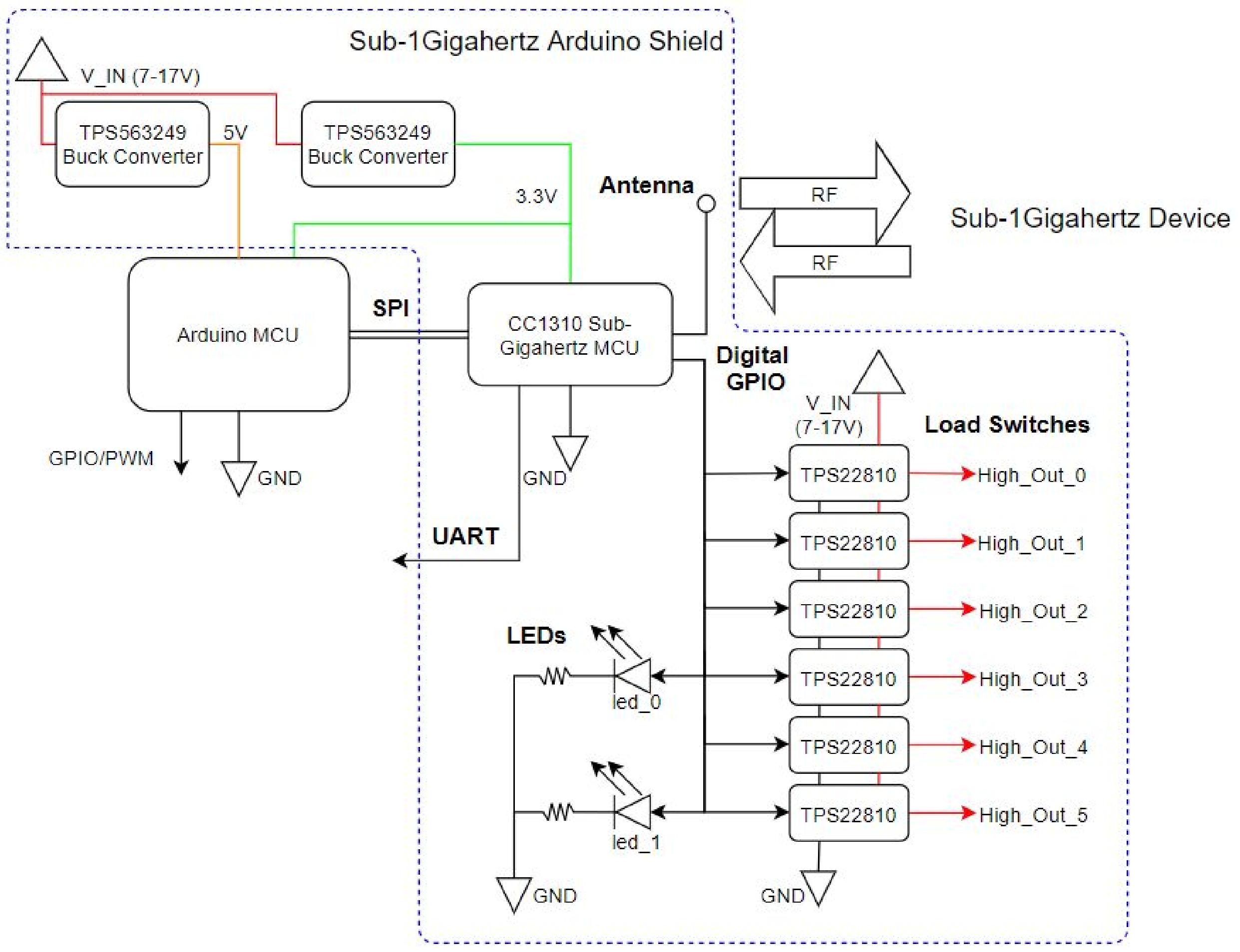


Figure 1: The block diagram for the Arduino Shield.

Originally, we planned to use the 3.3-volt output of the Arduino to supply power to the MCU. However, we soon realized that the Arduino cannot supply the current that the MCU requires. To solve this problem, we used a 3.3-volt buck converter to power the MCU in addition to using the Arduino to power it. Initially, we were unsure if the 3.3-volt buck converter should convert the input voltage to 3.3 volts or if it should convert the five-volt output of the other buck converter to 3.3 volts. Ultimately, we chose to convert the input voltage to 3.3 volts because it was more power efficient compared to other options such as an LDO. The efficiency of an LDO is

$$\eta = \frac{V\,out}{V\,in}$$

which for this case would be 66% efficient. Comparing this to the buck converter, we believe that the buck converter was the better option.

2.1 5-Volt Buck Converter

The 5-volt buck converter supplies power to the Arduino. It converts an input voltage between seven and seventeen volts down to five volts. It allows a maximum current of 0.6 amps since that is the maximum input current of the Arduino. To meet the low power goal of our project, we decided the 5-volt buck converter must be at least 80 percent efficient. To create the 5-volt buck converter, we used a TPS563249 buck converter [1] because we were already familiar with the part. We generated a circuit for the 5-volt buck converter using Texas Instruments' Webench Power Designer [2] by inputting the TPS563249 part and the circuit requirements stated above. The generated circuit for the 5-volt buck converter is shown in Figure 2.

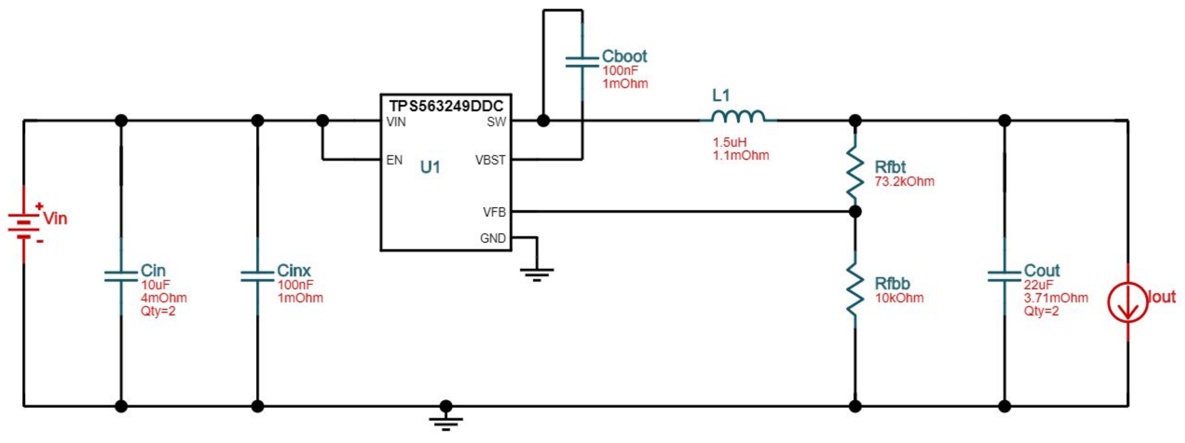


Figure 2: The generated circuit for the 5-volt buck converter using Webench Power Designer.

After we generated the circuit, we then recreated it as an Eagle schematic. We downloaded the ECAD files for the TSP563249 from the Texas Instruments website and imported them into Eagle. For the other parts of the circuit, we found what specific parts we needed to use in the generated circuit's bill of materials. Once we knew what specific parts were needed, we imported their ECAD designs into Eagle. The ECAD design of the inductor caused some confusion for a while. The generated circuit showed the inductor as a two-pin device while the ECAD design of the inductor showed it as a three-pin device. Concerned, we looked at the inductor's data sheet [3] hoping to find out why there was a third pin. Fortunately, the data sheet had an answer. The third pin of the inductor is not connected to the circuit. It exists to increase the mounting stability of the inductor. With that issue solved, we finished the Eagle schematic for the 5-volt buck converter. The Eagle schematic and the PCB layout of the 5-volt buck converter are shown in Figure 3 and Figure 4, respectively.

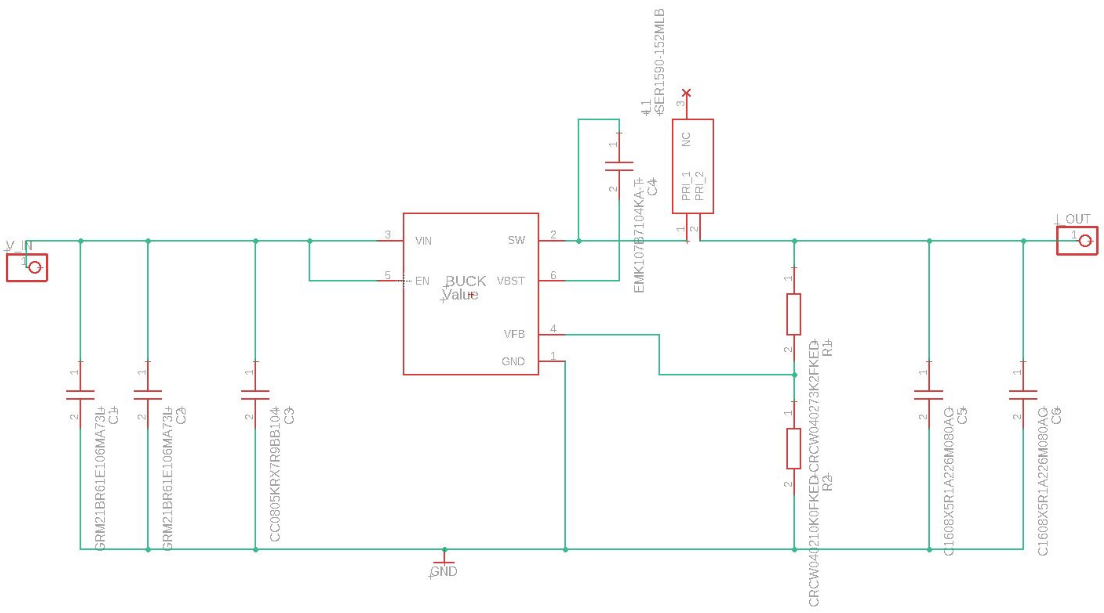


Figure 3: The Eagle schematic of the 5-volt buck converter.

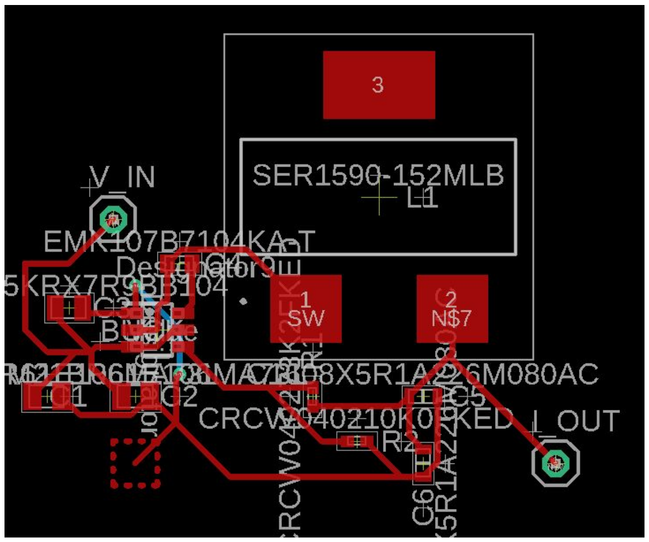


Figure 4: The PCB layout of the 5-volt buck converter.

2.2 3.3-Volt Buck Converter

The 3.3-volt buck converter takes an input voltage between seven and seventeen volts and converts it to 3.3 volts. We used another TPS563249 buck converter since it could also convert the input voltage down to 3.3 volts. Unlike the 5-volt buck converter, the 3.3-volt buck converter can output a much higher current. It was decided the maximum current of the 3.3-volt buck converter would be three amps. The 3.3-volt buck converter needs to be highly efficient to meet the low power goal of the project. We decided the efficiency of the 3.3-volt buck converter must be at least 80 percent. Using the parameters above, we generated a circuit for the 3.3-volt buck converter using Webench Power Designer [2]. The generated circuit for the 3.3-volt buck converter is shown in Figure 5.

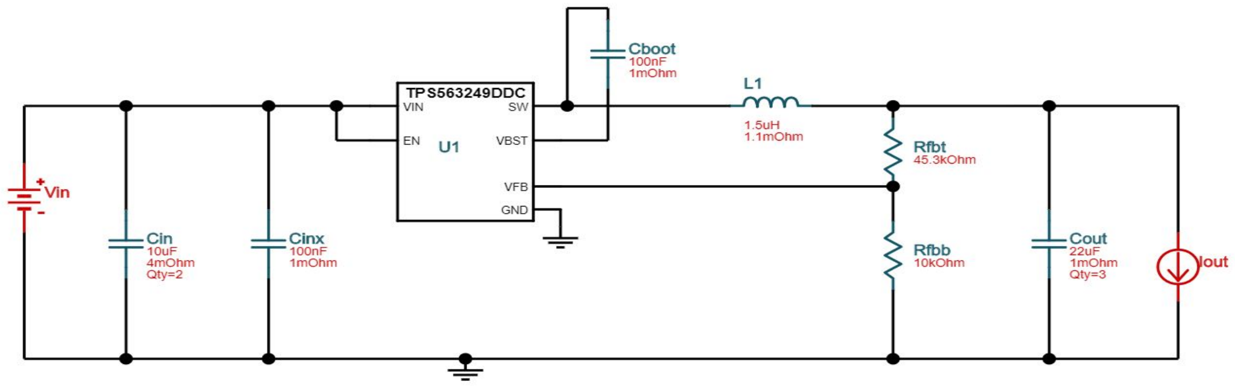


Figure 5: The generated circuit for the 3.3-volt buck converter.

The 3.3-volt buck converter's circuit is very similar to the 5-volt buck converter's circuit. The Rfbt resistor is now 45.3 kilohms and Cout has one more 22 uF capacitor. All the other parts are the same as the 5-volt buck converter. Using the generated circuit, we created an Eagle schematic and a PCB layout for the 3.3-volt buck converter. The schematic and the PCB layout are shown in Figure 6 and Figure 7, respectively.

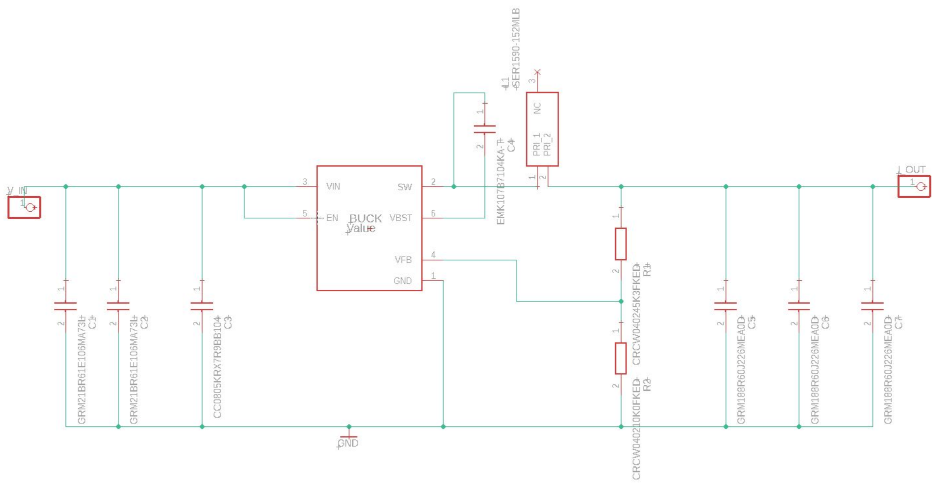


Figure 6: The Eagle schematic of the 3.3-volt buck converter based on the generated circuit.

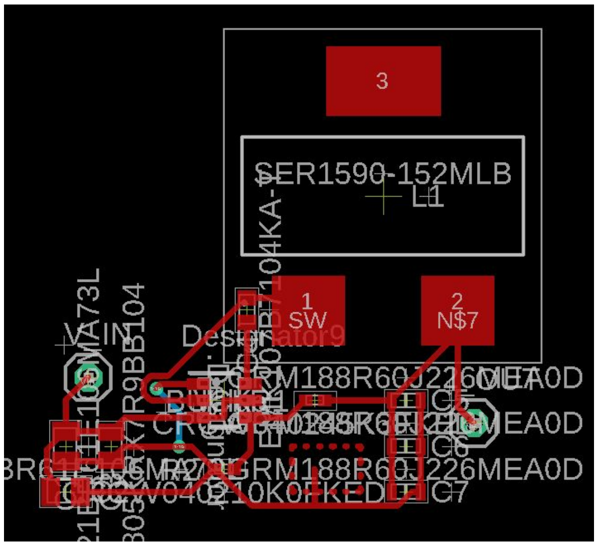


Figure 7: The PCB layout design for the 3.3-volt buck converter.

2.3 MCU

The MCU receives wireless signals for the Arduino shield using an antenna. We used the TI's TIDA-00484 reference design[4] as a template for our RF antenna design shown in Figure 8. Our MCU used a CC1310F128RHBR chip while the TIDA-00484 layout uses a CC1310F128RGZR chip. As shown on the CC1310 datasheet [5], the main difference between the pins of the two chips is that the CC1310F128RHBR chip has less DIO pins than the CC1310F128RGZR chip. We chose this package due to the pitch of the device pads being bigger than the other packages. This was done to assist in hand soldering the board and to help with cost. For a future iteration, this may be substituted for the RGZ package to add additional functionality to the shield.

The MCU features an ARM main CPU as well as a separate RF Core. This allows for much better flexibility with power as the main CPU can enter low power states while the program waits for an RF packet. The main CPU is able to communicate to the RF core through system calls which allow for high levels of programming abstraction.

The CC1310 has a number of features we did not use in this project such as AES encryption, an 8 channel ADC, a constant current source, and an 8 bit DAC. By using the larger package size, we might be able to give the user access to these features at minimal increase in cost.

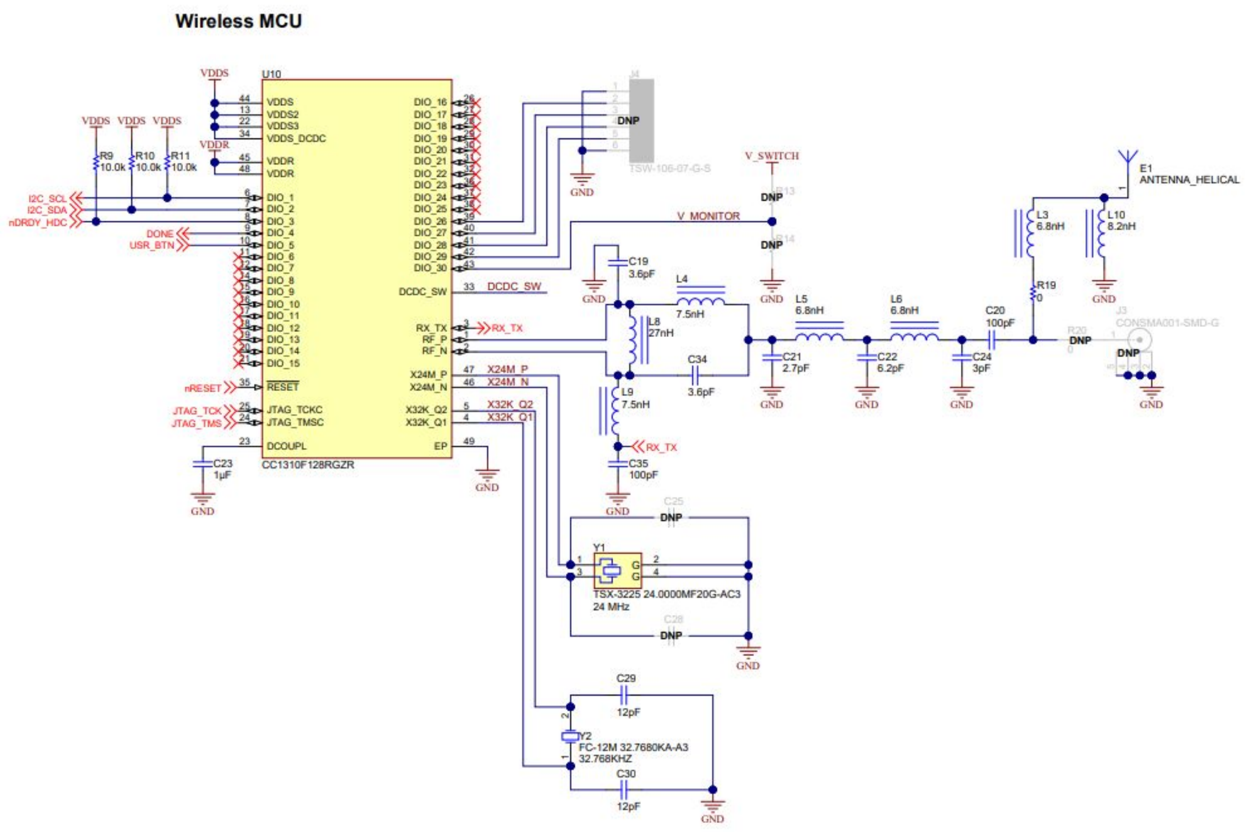


Figure 8: The TIDA-00484 layout. Our MCU uses a CC1310F128RHBR chip instead of the CC1310F128RGZR chip in the layout.

While we were working on the schematic and board design for the MCU, we initially used generic parts for the resistors, capacitors, and the inductors. We used generic parts because the TIDA-00484 reference design document [4] did not include a bill of materials. Christopher believed that there should be a bill of materials for it somewhere online, so we went looking. After a few hours of searching, we finally found a separate document that included the bill of materials for the TIDA-00484 [6]. We then swapped out all the generic parts in the schematic with the parts specified in the bill of materials. We also imported an ECAD file for the Arduino and connected it to the MCU. The MCU's schematic and board design are shown in Figure 9 and Figure 10, respectively.

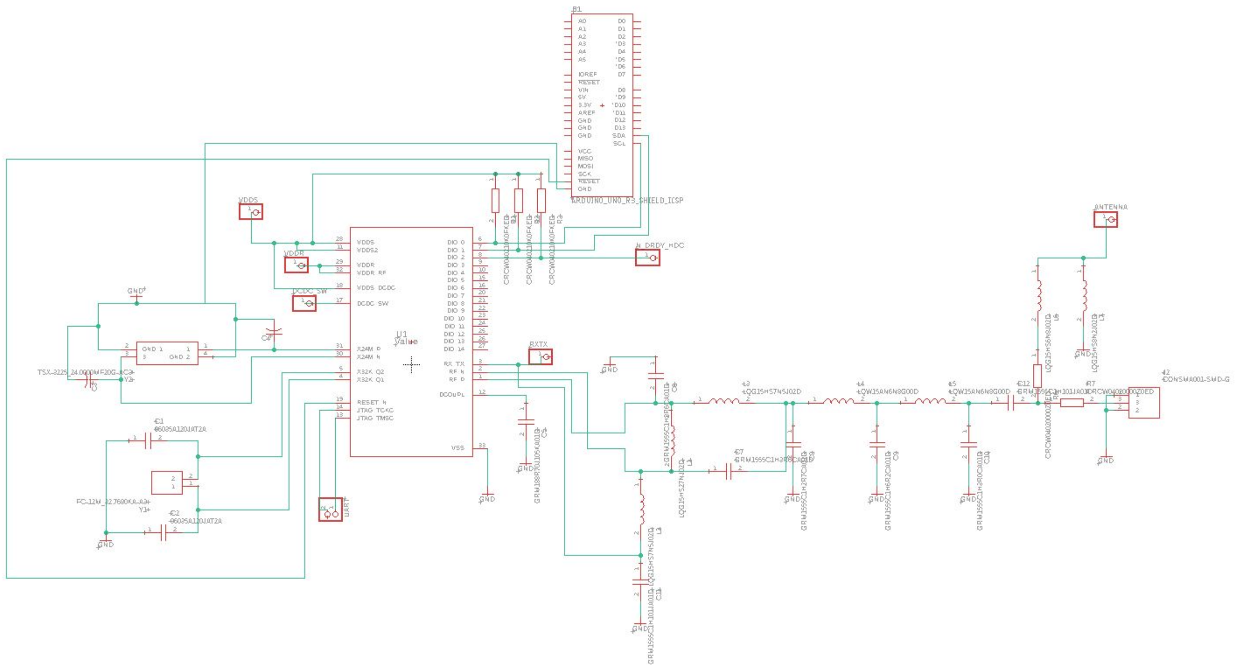


Figure 9: The schematic for the MCU. The Arduino is the part with many pins near the top and the CC1310F128RHBR chip is the part with many pins southwest of the Arduino.

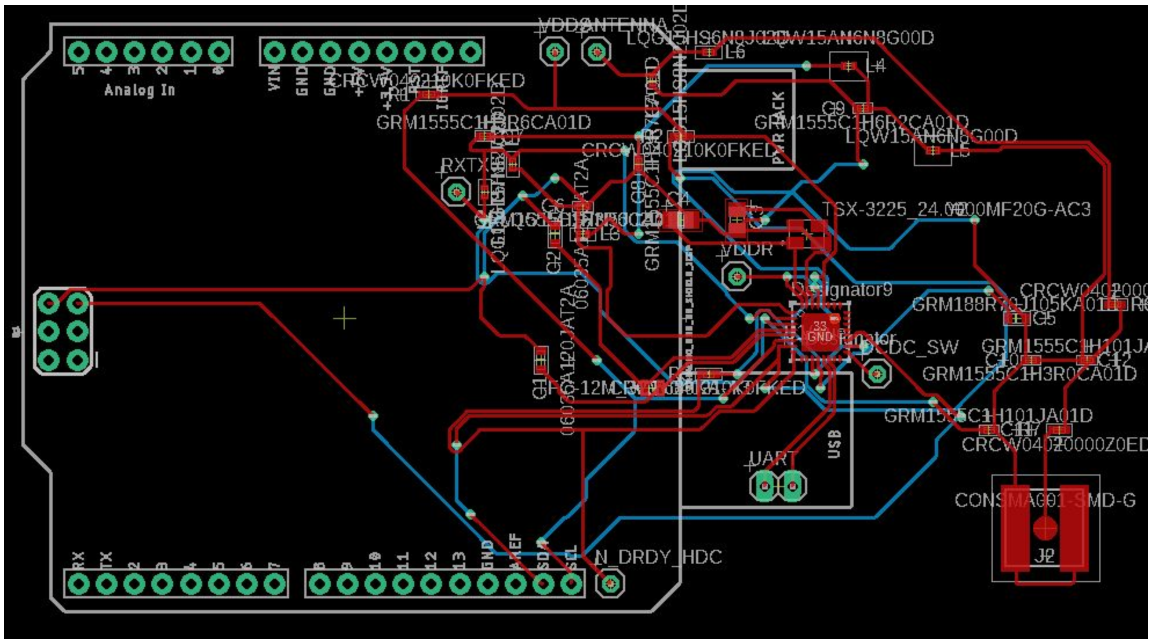


Figure 10: The board design for the MCU.

2.4 Finishing The PCB

The last step was to put everything together onto one schematic and one board. First, we added the load switches to the MCU schematic. We used six TPS22810DBVR [7] switches to make the load switches. We also added the two LEDs to the schematic. Then, we imported the 5-volt buck converter and the 3.3-volt buck converter into the schematic and wired everything together. The wiring was difficult near the right side of the CC1310F128RHBR chip because it was crowded with wires. We successfully wired everything together despite the difficulty. The finished schematic for the entire PCB is shown in Figure 11.

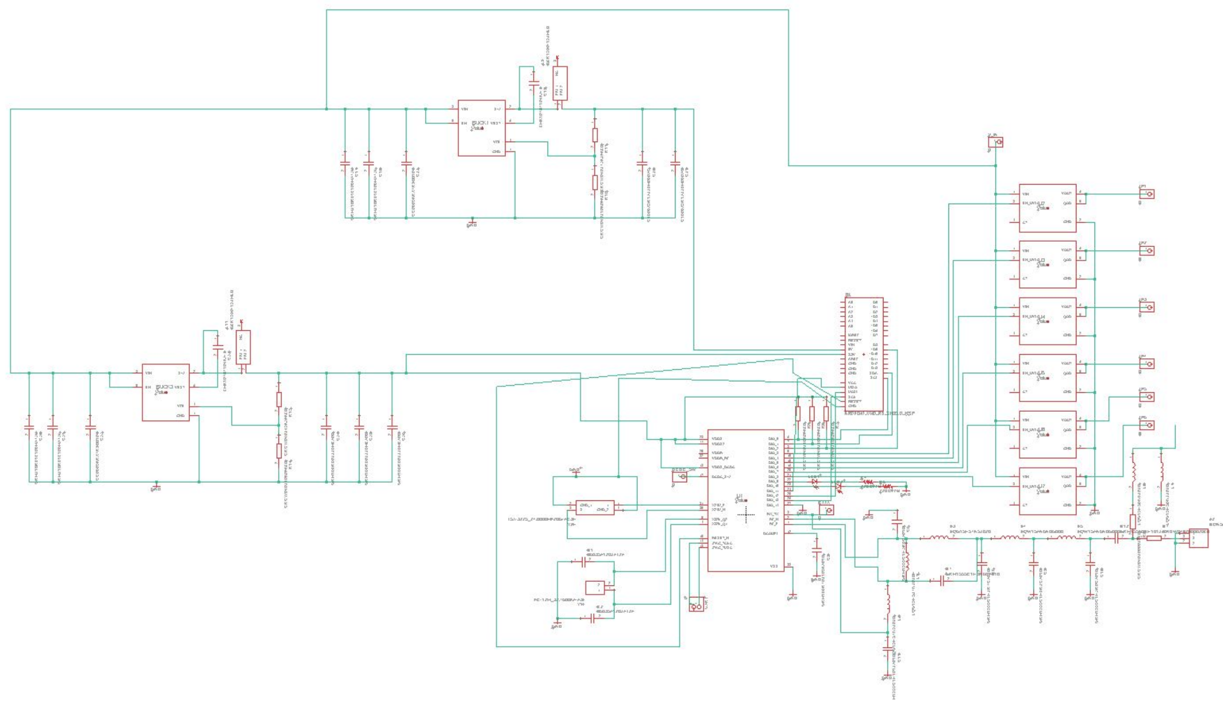


Figure 11: The finished schematic for the PCB.

After finishing the schematic, we had to complete the board design for the PCB. We moved the parts for the 5-volt buck converter, the 3.3-volt buck converter, the load switches, and the LEDs onto the board. We positioned the load switches near the bottom, the 5-volt buck converter near the top, and the 3.3-volt buck converter near the left side of the PCB. We also placed the LEDs to the right of the CC1310F128RHBR chip. Once all the parts were in place, we started working on the antenna. To create the antenna, we curled a 50-nanometer thick wire between the front and the back of the PCB ten times. The antenna we used is shown in Figure 12.

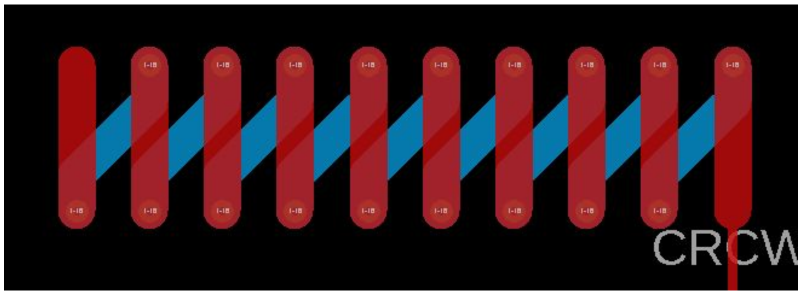


Figure 13: The antenna of our Arduino Shield.

After we made the antenna, we realized that the current PCB design was larger than 100 mm by 100 mm. That was a problem since the Arduino Shield is supposed to fit on a 100 mm by 100 mm PCB. To make the Arduino shield fit on a 100 mm by 100 mm PCB, we rearranged a lot of parts and wires. Once the PCB fit inside a 100 mm by 100 mm area, we ran the DRC test to see if the PCB design had any issues. The DRC found tiny bits of unconnected wires in the design. These were likely left over from when we rearranged a lot of parts and wires to make the design fit inside a 100 mm by 100 mm area. We deleted the left-over wires and looked at the next error. The DRC considered the end of the antenna to be an air wire since it was not connected to anything. The end of the antenna is not supposed to be connected to anything, so we manually approved the error. The DRC gave us no more warnings, so we generated the Gerber files and submitted them to PCBway.

The PCB design ended up failing PCBway's audit. They said that the design failed the audit because it did not have a keepout layer. To try and solve this error, we made a small keepout layer box near the bottom of the PCB. However, this did not fix the error. After a lot of trial and error, we realized that the problem had nothing to do with the PCB design. The problem was that we generated the Gerber files without including the layers of the PCB. Making sure to actually include the layers of the PCB, we once again generated the Gerber files and submitted them to PCBway. This time, the PCB design passed the audit. The finished PCB design is shown in Figure 13.

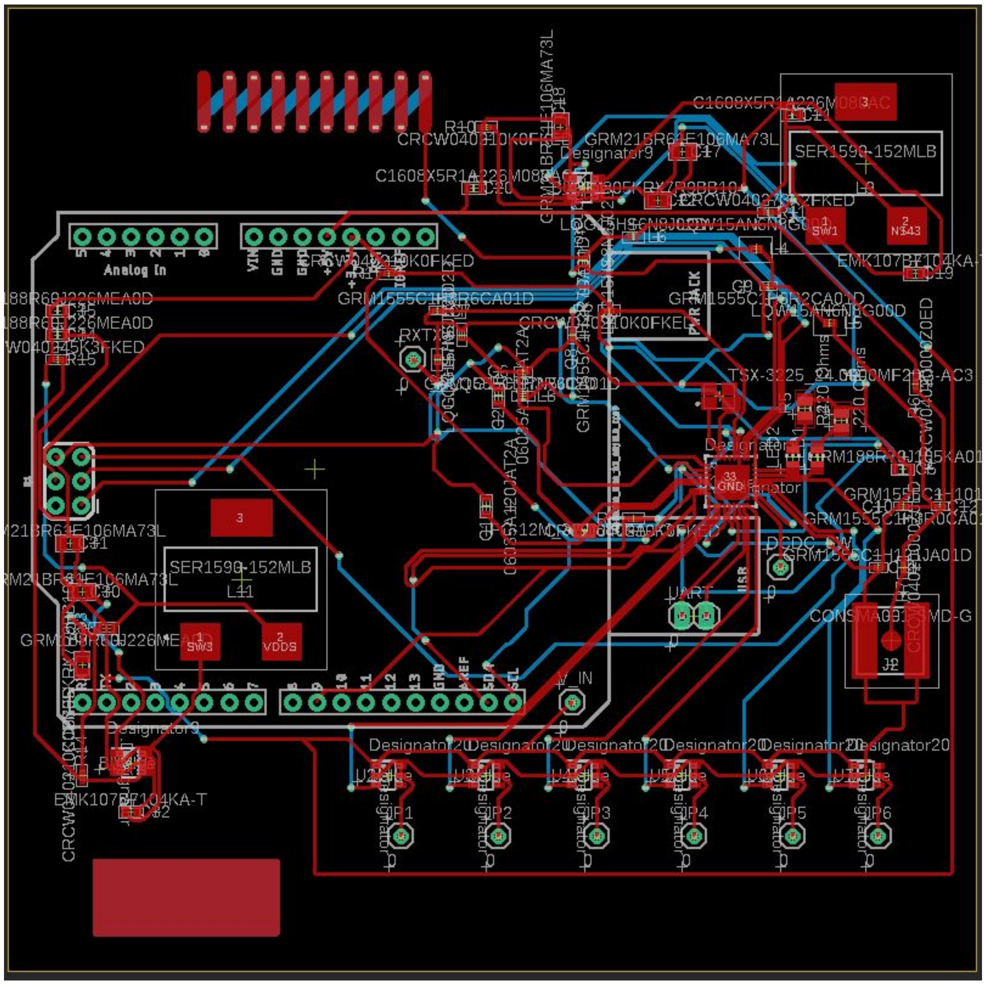


Figure 13: The finished PCB design.

2.5 CC1310 Software

The CC1310 runs on TI's Real Time Operating System (RTOS). This OS is a stripped UNIX operating system which has guaranteed finite execution time for system calls. These RTOS operating systems are useful for low power and system critical applications where starvation of program is not acceptable. These operating systems are common in ARM cores such as microcontrollers and FPGA SoCs where memory space is limited and power is a primary concern.

TI's RTOS supports POSIX threading (aka Pthreads) on the CC1310's single ARM core processor which means that separate functions are able to share execution time of the processor with a set priority. This means that code can be run in a pseudo-simultaneous manner. In practical terms, the threads allow the program to split up code execution in order to best utilize the processor. To synchronize code between threads, we use semaphores, an atomic counter, which allows threads to wait (and yield execution time) until the semaphore is posted (counted up).

2.5.1 SPI Slave Thread

The primary thread of execution on the Sub-1Gigahertz Shield is the SPI Slave Thread. This thread handles setting the SPI Slave transaction buffer, SPI Slave transaction length, and decoding the incoming SPI buffer.

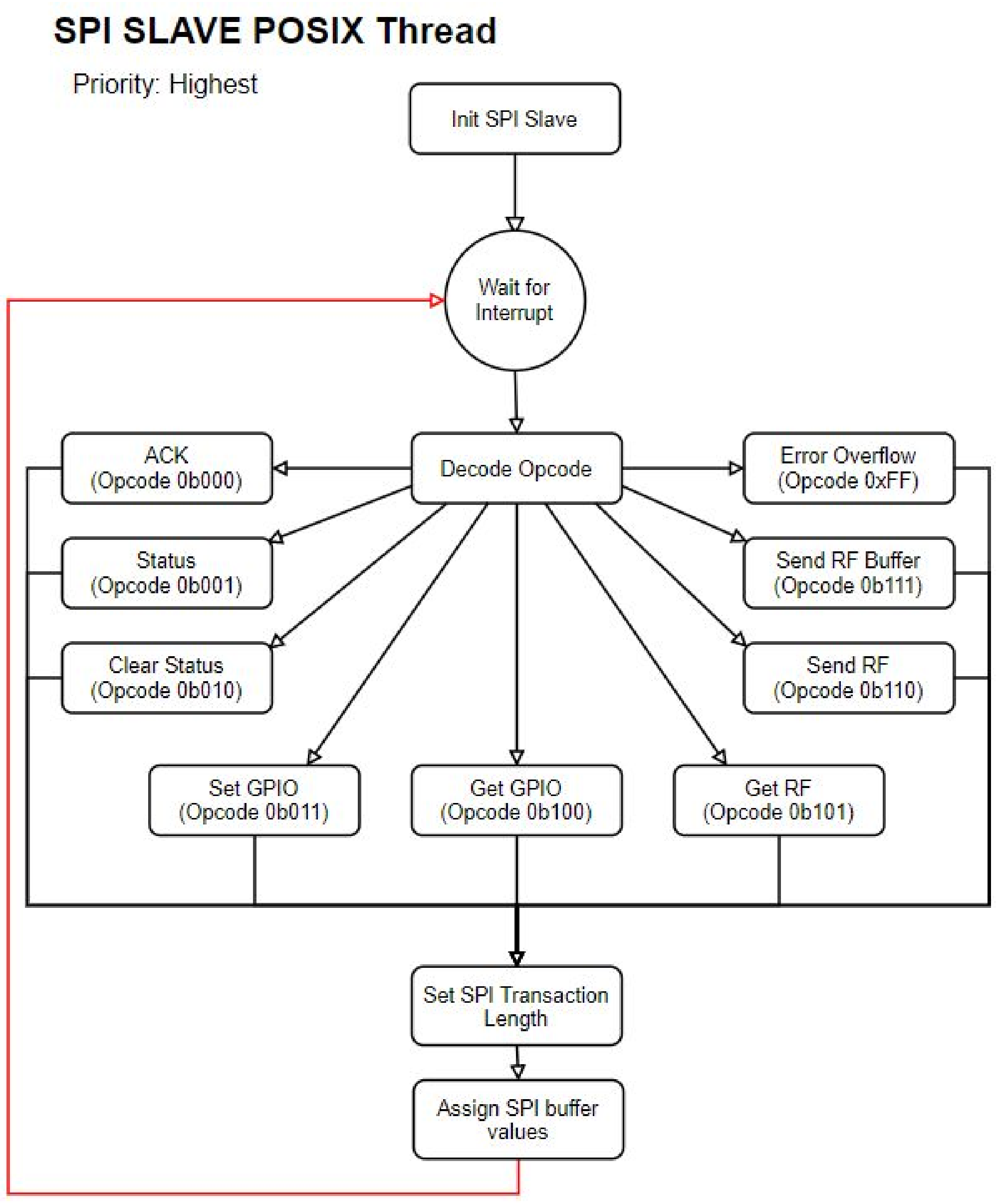


Figure 14: SPI Slave Pthread Flowchart

2.5.2 RF Tx Thread

The Tx thread spends a majority of its time waiting for a signal (semaphore post) from the SPI thread. This semaphore means the thread will only run when a command is sent from the Arduino along with a new buffer of data. Once a signal is received, the thread cancels the RF Rx call, runs the Tx command, then signals the RF Rx thread to continue. This synchronization is needed so that the Rx thread does not run the Rx command before the Tx thread runs the Tx command.

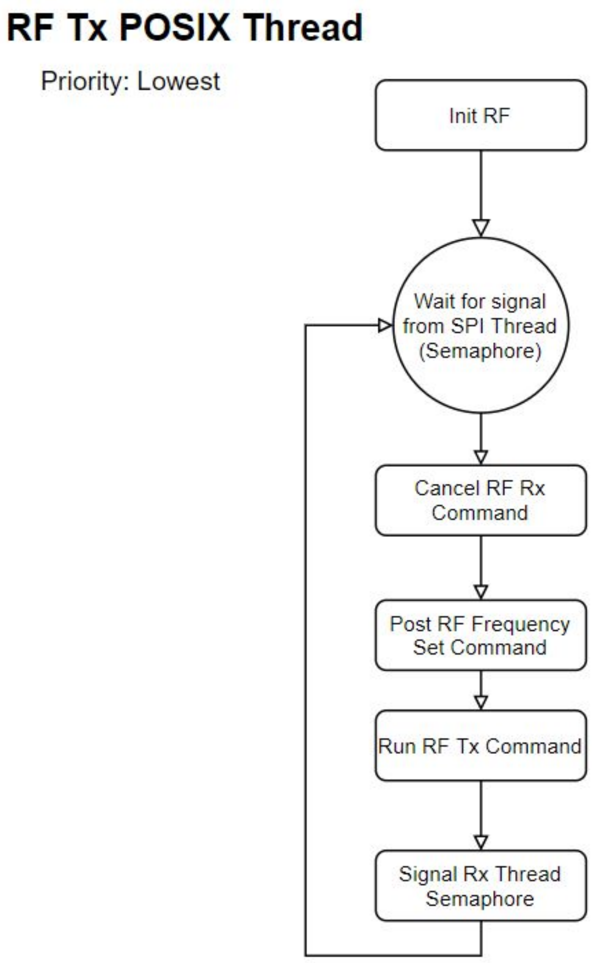

RF Tx POSIX Thread
Priority: Lowest
Init RF
Wait for signal from SPI Thread (Semaphore)
Cancel RF Rx Command
Post RF Frequency Set Command
Run RF Tx Command
Signal Rx Thread Semaphore

Figure 15: RF Tx Pthread Flowchart

2.5.3 RF Rx Thread

This thread runs the Rx command until cancelled by the Tx thread. Running the Rx command stops execution of the thread which is the main reason why threading was required by the project. When an RF packet is received, the callback function is run which gets the RF data and updates the shield status with RF_Received.

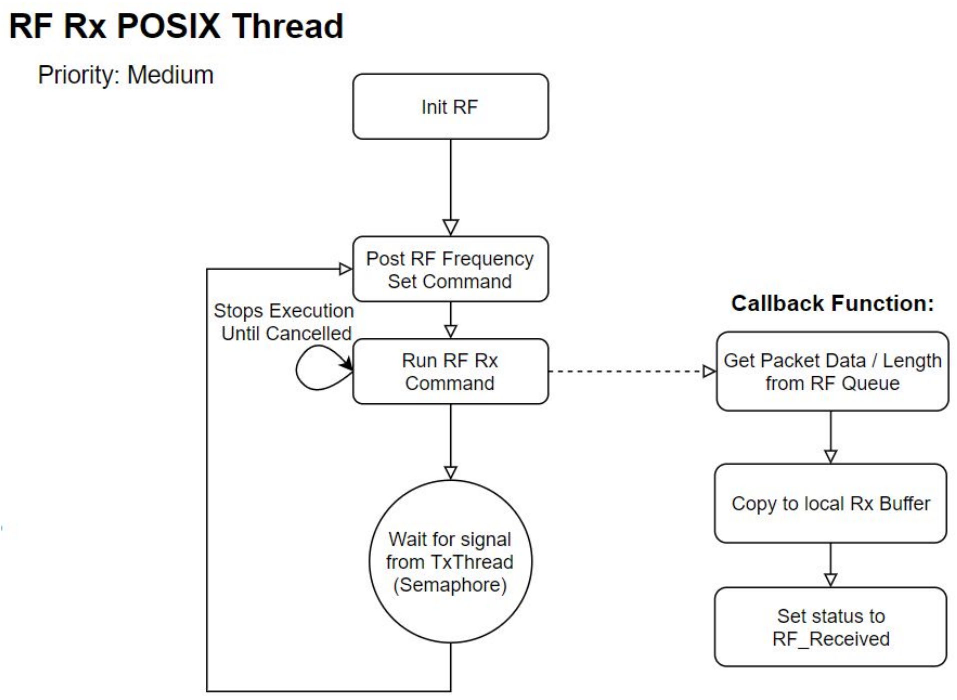

RF Rx POSIX Thread
Priority: Medium
Init RF
Post RF Frequency Set Command
Stops Execution Until Cancelled
Run RF Rx Command
Wait for signal from TxThread (Semaphore)
Callback Function:
Get Packet Data / Length from RF Queue
Copy to local Rx Buffer
Set status to RF_Received

Figure 16: RF Rx Pthread Flowchart

2.5.4 Remote IO Thread

The IO thread was developed for the remote emulated by a TI Evaluation Board (aka EVM). The purpose of this thread is to take in two button inputs which trigger RF Tx commands. This thread also interfaced with slightly modified Tx and Rx threads which allowed the user to see threads sent and had special RF decode functions.

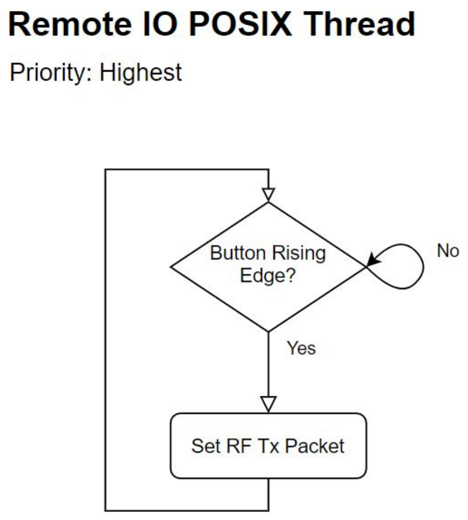


Figure 17: Remote IO Pthread Flowchart

2.6 Arduino Software

The Arduino software polls the shield using SPI (with Arduino as Master) calls which returns the status. Once the status returns that the RF is received, the Arduino retrieves the RF packet. The user is then given the flexibility on how the information is interpreted. For our demo, we implemented a primitive identification and message scheme which allowed the Arduino and remote to send and receive commands and strings. The source code can be found in Appendix B.

3.0 Implementation

Once the PCBs arrived, we started going to the lab to solder parts onto them. First, we soldered on the chips for the two buck converters and the load switches. Then, we soldered on the LEDs and the inductors for the two buck converters. We did not solder the third pin of the inductors since they were not connected to the circuit. With that done, we started soldering on the resistors, capacitors, and the rest of the inductors. We did run into some issues while soldering. We found that capacitor 21 was extremely close to the 5-volt buck converter's inductor. This made soldering on capacitor 21 extremely difficult and awkward, but we eventually managed to solder it on. Capacitor 21 and the 5-volt buck converter's inductor are shown in Figure 14. We also realized that we were missing parts for R6, R7, and R15. We quickly ordered parts for them because we needed them to arrive before the final demo.

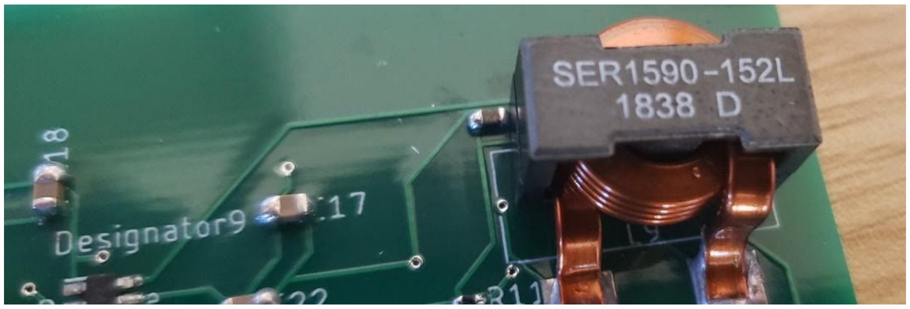


Figure 14: Capacitor 21 and the inductor. Capacitor 21 was almost touching the top left corner of the inductor.

Once the missing parts arrived, we went to the lab and started soldering on the remaining parts. We soldered on the missing parts and started soldering in the pins. We soldered on the pins for the V_IN, UART, DCDC, RXTX, and the pins that attach to the Arduino. However, we soon realized that the pins we were using did not actually fit onto the Arduino. We did not have time to order different pins, so we had to improvise. We unsoldered the pins and replaced them with wires. The Arduino was now connected to the Arduino Shield through wire instead of pins. The replacement wires are shown in Figure 15.

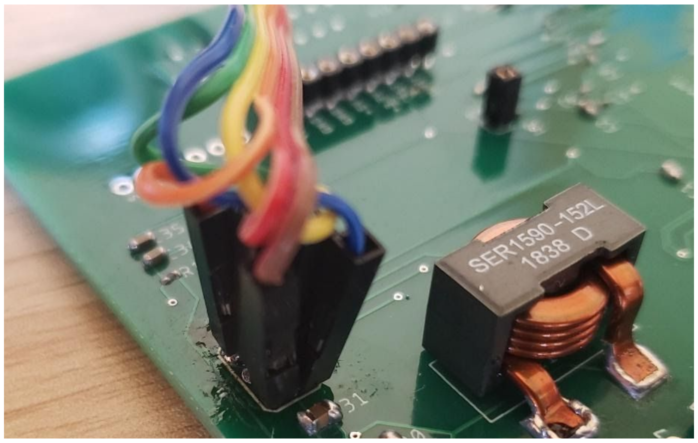


Figure 15: The replacement wires for the pins that did not fit onto the Arduino.

The final part that was soldered on was the CC1310F128RHBR chip for the MCU. Since all the soldering pads were underneath the chip, we could not use the soldering iron like all the other parts. We used a heat gun instead. First, we put a very small amount of solder on the PCB's soldering pads. Then, we put a drop of flux and the chip onto the soldering pads. Now it was time to use the heat gun. Initially, we set the heat gun to 300 degrees Celsius like the tutorial video that we watched [8] suggested. However, the MCU did not get soldered onto the board at this temperature. We ended up increasing the temperature of the heat gun all the way up to 450 degrees Celsius before the MCU would finally get soldered on. With the MCU soldered on, the PCB was ready to be tested. The completed PCB is shown in Figure 16.

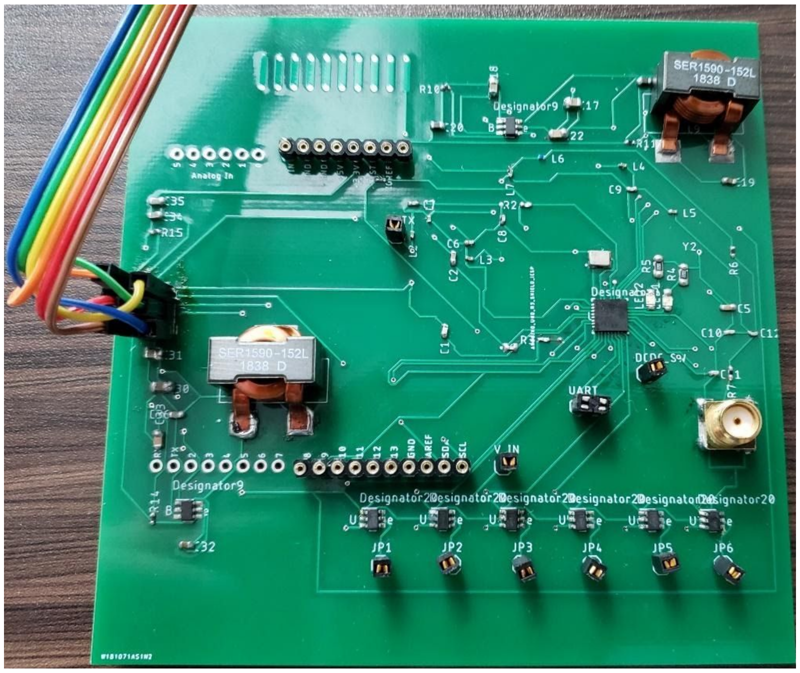
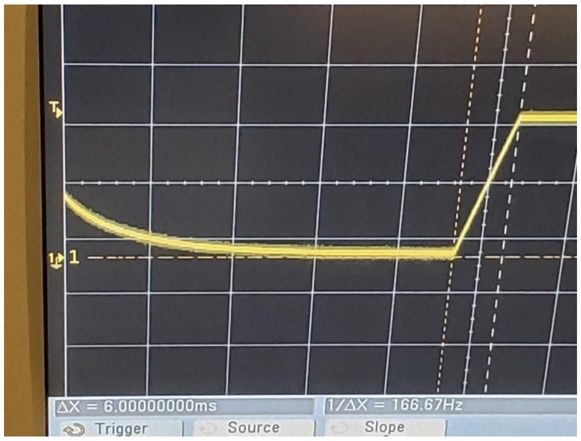


Figure 16: The PCB with all the parts soldered on.

4.0 Verification

First, we tested the two buck converters. The two buck converters must output a voltage within 5% of the target voltage, remain below the maximum operating temperature (125 degrees Celsius), and must be at least 80% efficient. The first thing we tested was the output voltage. We set the input voltage to 12 volts and measured the output voltage with a digital multimeter. We found that the output of both buck converters was around 5% higher than the target voltage. The output of the 5-volt buck converter was 5.27 volts and the output of the 3.3-volt buck converter was 3.47 volts. This should not cause any problems for our circuit and fulfills requirement one in the buck converter's R&V table. We then obtained the output current and calculated the efficiency. The efficiency of the two buck converters were both 88% efficient. This fulfills requirement three in the buck converter's R&V table. It should be noted that the measured efficiency was slightly less than Webench Power Designer's estimated efficiency. It estimated that the efficiency would be 90%. Then, we measured the temperature of the circuit with a thermometer. The temperature of the circuit was not anywhere near the maximum operating temperature. The temperature of the circuit never rose above 60 degrees Celsius. This fulfills requirement two in the buck converter's R&V table.

Then, we tested the other parts of the board. First, we tested the voltage output of the 3.3 volt pin of the Arduino to make sure it meets the input range of the CC1310. Using a multimeter, we found that the voltage output was 3.301 volts. Since this voltage fell within the input range of the CC1310, requirement one of the Arduino Shield R&V table was fulfilled. We tested the MCU, however, we found that we could not program it. The MCU needed more pins than our PCB had in order to be programmed. Since the MCU could not get programmed, the MCU would not work. We could not fix the problem because we did not have time to order new PCBs. Fortunately, we were still able to test our code by using EVMs (evaluation modules) instead of the MCU. Next, we tested the load switches. We found that they outputted the correct voltage (12 volts), had a rising voltage time of six milliseconds, and a falling voltage time of 25 milliseconds. The rising voltage time and the falling voltage time are shown in Figure 17 and Figure 18, respectively.



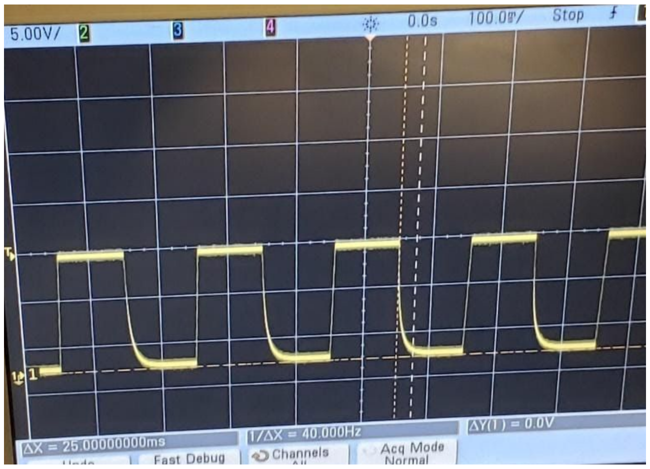


Figure 17 (Left): The rising voltage time of the load switches is six milliseconds.

Figure 18(Right): The falling voltage time of the load switches is 25 milliseconds.

5.0 Cost and Schedule

Week	**Alex Beck**	**Christopher Baldwin**
10/5	Finish schematic for buck converters	Figure out SPI for Arduino
10/12	Finish schematic for MCU	Figure out SPI for CC1310
10/19	Finish designing PCB	Order components
10/26	Order PCBs	Figure out RF communication for CC1310
11/2	Obtain case for the Arduino Shield	Figure out RF communication for CC1310
11/9	Start Soldering PCB	Figure out RF communication for CC1310
11/16	Finish Soldering PCB	Finish Software
11/23	Thanksgiving Break	Thanksgiving Break
11/30	Work on Presentation	Work on Presentation
12/7	Work on Final Report	Work on Final Report

Table 1: The weekly schedule for our project.

Part	Amount ordered	Cost ($)
PCB	10	53 (Including Shipping)
CC1310F128RHBR chip	1	4.379
TPS563249	2	0.50
TPS22810	6	4.81
TSX-3225 24.0000MF20G-AC3	1	0.33
Resistors	15	0.78
Capacitors	36	2.70
Inductors	11	6.85
Pins	19	22.61

Table 2: The parts we purchased and the price of those parts.

Total Cost of Parts = $95.96

Labor Costs = $30 per hour * 30 hours * 2 people = $1800

Total Cost = $1800 + $95.96 = $1895.96

In an industrial setting, the labor cost should be much lower because the producer can make the product in much less time. In addition, they do not have to research and design the project like our team did. This will reduce the time needed by a large amount.

6.0 Conclusion

Our project was very ambitious to begin with and with only two group members, things only got harder. We used 23 unique components on the board with many of them being incredibly small SMD components.

6.1 Accomplishments

The buck converters gave us acceptable outputs and reached our efficiency goals. The load switches functioned and allowed us to output at much higher voltages than the Arduino or CC1310. The code we developed worked on the EVM boards and proved the concept of the shield to be a success.

6.2 Uncertainties

While we know the software worked we were not able to successfully flash the program onto the MCU. This means we did not get to test the filter and antenna of the shield which we had developed.

6.3 Future Work / Alternatives

In the future, we may try this concept again. The project needs to have the board redesigned in order to function as a stackable shield. Another offshoot of the project is to create a separate board which attaches by a cable. This would allow for other devices such as raspberry pis to also utilize the same framework. This could mean that the user also is able to run an internal web server and connect their devices to IoT managers like Alexa.

6.4 Ethical Considerations

We believe our device upholds the ethical considerations of the IEEE Code of Ethics [9]. Our project seeks to enable people to create more projects of their own and ultimately seeks to empower those especially with limited resources all the while keeping their devices secure and private.

7.0 References

[1] TPS563249, Texas Instruments, Available at:
https://www.ti.com/product/TPS563249

[2] Webench Power Designer, Texas Instruments, Available at:
https://www.ti.com/design-resources/design-tools-simulation/webench-power-designer.html

[3] Ser1590 Shielded Power Inductors, Mousre.com, Available at:
https://www.mouser.com/datasheet/2/597/ser1590-270632.pdf

[4] TIDA-00484 Schematic and Block Diagram (Rev. C), Texas Instruments, 2018. Available at:
https://www.ti.com/lit/df/tidrey5c/tidrey5c.pdf?ts=1604001692547&ref_url=https%253A%252F%252Fwww.google.com%252F

[5] CC1310 SimpleLink^TM Ultra-Low-Power Sub-1 GHz Wireless MCU datasheet (Rev.D), Texas Instruments, July 2018. Available at:
https://www.ti.com/lit/ds/symlink/cc1310.pdf?ts=1604010202434&ref_url=https%253A%252F%252Fwww.ti.com%252Fproduct%252FCC1310

[6] TIDA-00484 BOM (Rev. C), Texas Instruments, 9/11/2018. Available at:
https://www.ti.com/lit/df/tidrey6c/tidrey6c.pdf?ts=1604080634089&ref_url=https%253A%252F%252Fwww.google.com%252F

[7] TPS22810, 2.7-18-V, 79-m Ω On-Resistance Load Switch With Thermal Protection, Texas Instruments, January 2018, Available at:
https://www.ti.com/lit/ds/symlink/tps22810.pdf?HQS=TI-null-null-mousermode-df-pf-null-wwe&ts=1606858018556&ref_url=https%253A%252F%252Fwww.mouser.com%252F

[8] How To Solder QFN MLF chips Using Hot Air without Solder Paste and Stencils, CuriousInventor, May 2, 2010, Available at: https://www.youtube.com/watch?v=c_Qt5CtUlqY

[9] IEEE Code of Ethics, IEEE, 2020. Available at: https://courses.engr.illinois.edu/ece445/documents/ECE_445_Final_Report_Guidelines.pdf

8.0 Appendix A: Source Code

The source code can be found on github at the following link: https://github.com/chrisbaldwin2/Sub-G-Shield/

9.0 Appendix B: Requirement and Verification Table

9.1 Buck Converter

Requirements	Verification	Testing Result
1. Provide 5 V ±5% from a 7-17V DC source in the range of 0-600 mA	1A. Measure the output voltage using a multimeter, ensuring the output voltage stays within 5% of 5V with a current source load sweeping from 0 to 600 mA.	1A. Measured at 5.27420V for 5V buck and 3.47749 for 3.3V buck under no load. We may tweak the feedback resistor in the future but it is acceptable. (PASS).
2. The system should remain below the maximum operating temperature of all the devices at room temperature (20 °C - 30 °C).	2A. During verification for Requirement 1 an IR thermometer or K-type thermocouple will be used to check all devices and ensure they remain within their thermal operating temperature.	2A. The buck did not reach over 60 °C. The acceptable range is (-40 °C, 125 °C) (PASS).
3. Efficiency of the DC-DC conversion from 7-17 V to 5 V must be 80% or higher at 600 mA.	3A. We will measure the current and voltage at both the input and output when the output current is at 600 mA.We will use this to calculate the total power and efficiency of the buck/PMIC.	3A. Worst case efficiency measured was 88.15% (90% predicted) (PASS).

9.2 Sub-1 Gigahertz Arduino Shield

Requirements	Verification	Testing Result
1. CC1310 controller will be powered from the 3.3 V output of Arduino	1A. Measure 3.3 V output of Arduino with a multimeter to ensure that it will meet the input range of the CC1310 with input and load transients.	1A. The Arduino measured 3.301V (Acceptable 1.8V, 3.8V) (PASS).
2. The shield will be able to send and receive instructions and data with the Arduino over the I2C and/or SPI bus protocols.	2A. Given a command from the Arduino over I2C and/or SPI, we ensure that the CC1310 is able to trigger a response**.	2A. The shield is able to trigger GPIO, send data over serial com, and send RF data (PASS).
	2B. The shield will also be able to send data to the Arduino either by having the Arduino poll a register on the CC1310 or by changing the voltage of a digital pin on the Arduino.	2B. The shield was able to send data by having the Arduino poll the status register of the shield (PASS).
3. The shield will trigger RF/GPIO signals from commands received over I2C and/or SPI bus protocols. The shield will also buffer commands sent by the remote and/or other shields.	3A. We will measure the GPIO output signal with a multimeter to ensure that the output reaches logic high* and returns to logic low* whenever specified by the command/message.	3A. We were able to measure and see that the LED reached logic high due to an input over SPI (PASS).

* Logic high refers to the range between VDD and VDD/2. Logic low refers to the range between VCC and VDD/2. VDD refers to the max rated current of that IO value and VCC refers to the effective ground/neutral/zero voltage of the subsystem.

** A triggered response from the CC1310 may be any internal/external state change which is caused by an external stimulus. A triggered response may be but is not limited to an internal register value change, a change in GPIO output, an RF command/message output, a print statement over UART, etc.. External stimulus may be but is not limited to an RF command sent by another Sub-Gigahertz chip either by shield, remote, or EVM; a change in GPIO input; a command/message sent over I2C and/or SPI; etc..