ECE 445: Senior Design Laboratory

M.E.L.O.D.I.C Final Report

Macrae Wilson, Ryan Libiano, Colin Devenney

Team #37

Date Written: March 27, 2024

Contents

1 Introduction

1.1 Problem

A common problem associated with live performing is the 'rat's nest' of audio and control cables required to run front-of-house (FOH) equipment, digital effects, and instruments. However, in recent times UHF, VHF, and ISM systems have taken mainstay in the industry. For a large performance, having a $10,000+ rack dedicated to wireless audio systems makes sense. However, for the performing musician on a budget, such as a small house band or coffee shop artist, professional UHF, VHF, and ISM systems are not feasible to operate. Although low-cost or used legacy systems are popular amongst amateur musicians, they often suffer from problems such as data packet collisions from co-existing network protocols, interference from existing UHF and VHF television bands, and/or lack of scalability or configurability.

1.2 Solution

In order to combat this, we developed M.E.L.O.D.I.C., a low-cost, scalable, configurable, and high-fidelity wireless audio link compatible with commonly used audio equipment in the live audio industry. We used a commercial off-the-shelf Radio Frequency System-on-Chip (RF SOC), specifically the TI CC8530, commonly found in wireless headphones and karaoke systems. This chip is an attractive choice due to its operation in the ISM band, use of adaptive frequency hopping techniques for co-existence with other ISM devices, and configurable to either be an audio transmitter or receiver. Due to the configurability and low cost of the chip, our transmitter and receiver will have very similar circuit schematics, which will make it cheaper to manufacture multiple sets of transmitter and receivers.

1.3 High-level Requirements

- The system must transmit audio that meets or exceeds lossless CD audio standards, with a specific sampling rate of 44.1 kHz and a bit depth of 16 bits. This ensures high-fidelity sound reproduction suitable for professional live performance contexts.
- The system must co-exist with other 2.4 GHz wireless protocols (such as Wi-Fi and Bluetooth) without causing or suffering from interference that degrades performance. This will be quantitatively measured by maintaining a packet error rate (PER) below 1% in environments populated with at least three active 2.4 GHz sources.
- The device must feature a human-friendly user interface, equipped with an LCD that displays essential information including but not limited to battery status (with at least 10% granularity), network statistics (such as signal strength and PER), and unique device identification. This information must be easily readable under typical indoor lighting conditions from a distance of at least one meter.

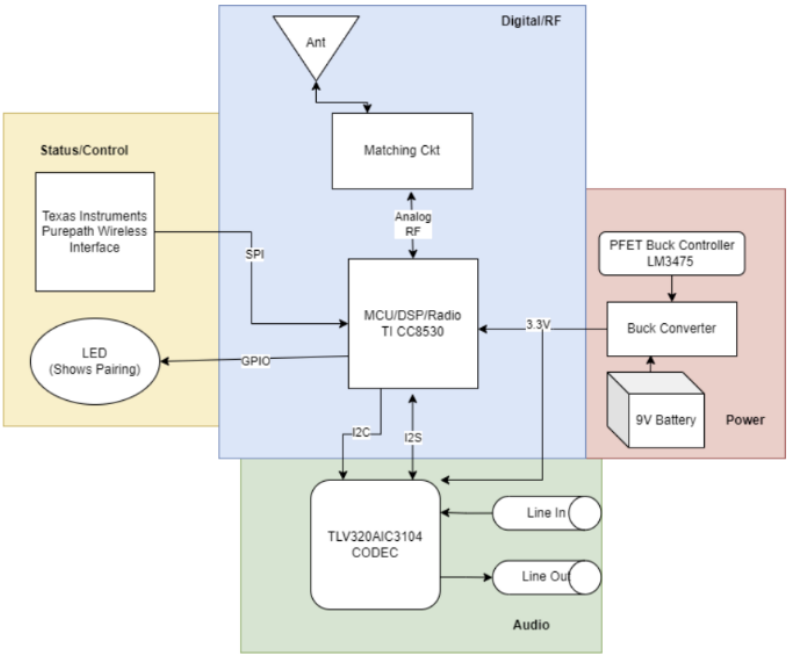


Figure 1: Block Diagram

2 Design

2.1 Design Procedure

2.1.1 Control

Our original desing plan was to utilize an STM32F103C8T6 Microcontroller to operate the CC8530 in host-controlled mode. Additionally, SPI and I2C commands would be used to program the CC8530 and the TLV320AIC3204 audio codec. Finally, a 16x2 LCD would display the following information about each device: network statistics, battery status, and device ID. Figure 2 shows the original GUI design flowchart. For the SPI programming, two main requests are utilized from the CC8530 instruction set: CMD_REQ and READBC. CMD_REQ is used for all main programming aspects of the RF chip. READBC reads the necessary data for the LCD and debugging. The code for both functions is shown in the appendix.

Ultimately, our control subsystem changed the most throughout the process. Instead of using the STM32, we used Texas Instruments' Purepath Wireless Configurator and Commander for flash programming and reading information required to meet the high-level requirement for the control system. An LED was attached to a GPIO pin on the CC8530 for pairing status. A slow blinking light means the device is alone (master unit control_enable activated). A fast blinking light means the device is in pairing mode. A solid light means the device is paired on the network. This was key to easily determining if the devices were connected without running SPI commands. The pairing process went as follows (references for the commands shown below are included in the appendix):

- One CC85xx is flash-programmed with Host-Controlled Master Firmware
- NWM_CONTROL_ENABLE is invoked by the master to begin pairing process
- NWM_DO_SCAN is invoked by slave, master device ID is returned
- NWM_DO_JOIN is invoked by slave with master ID as argument
- Use NWM_GET_STATUS on both master and slave to see information about current audio network
- Use PS_AUDIO_STATS and PS_RF_STATS to see RF and Audio statistics (used in verification)

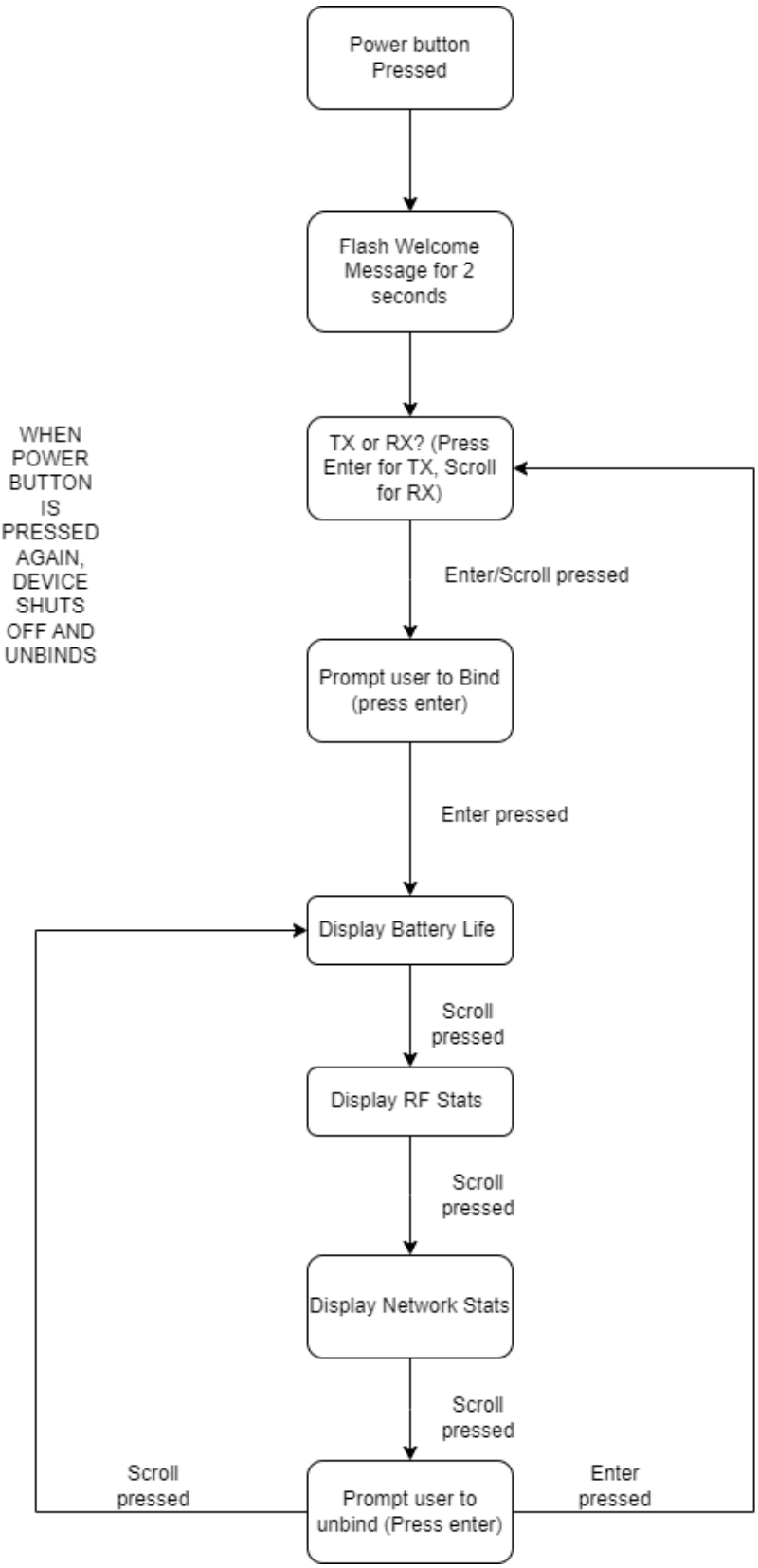


Figure 2: GUI Flowchart

I2C commands for the codec are handled by the CC8530's MCU. Pseudocode for these commands is also shown in the appendix. With these modifications, we were able to see network statistics and device ID for both the master and slave devices. However, we were not able to view battery level because Vbat (pin used for reading battery level of the CC8530) was tied to ground. Due to Texas Instruments' Proprietary SPI interface, programming the STM32 to handle the necessary SPI commands proved difficult. Figure 3 shows how the Texas Instruments SPI interface works. When we tried to read data from the MISO bus, it always showed high even though it should only be high shortly after CSn is pulled low. This made reading data from the CC830 impossible. Furthermore, the ST-LINK programming interface we used didn't include a UART bus for serial interfacing. This forced us to use the LCD for debugging which proved less than ideal.

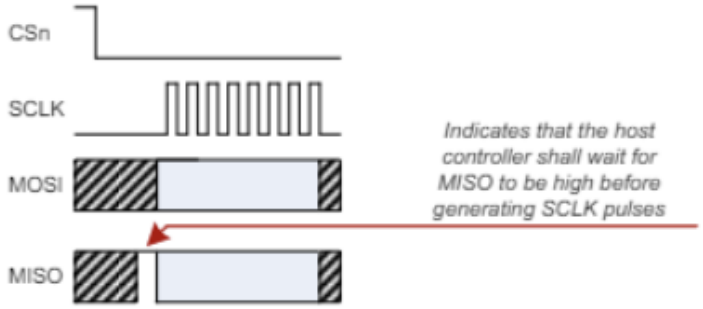


Figure 3: Texas Instruments SPI Interface

2.1.2 Power

The main power supply for our device is a 9V battery. This is brought down to the 3.3V that the CC8530 RF SoC needs using a hysteretic buck converter. The buck operates using TI's 10-V hysteretic PFET buck controller, LM3475, which controls the switching for a custom-designed PCB and buck circuit. This enables the system to operate consistently for an extended period, even as the 9V battery's charge diminishes until the battery has less than 3.3V. The LM3475 internally contains noise suppression circuitry, however, to ensure no noise propagates through the ground plane or interferes with the rest of the circuit we implemented external noise-dampening circuitry. To counteract noise spikes created by trace parasitic inductance several measures were taken. First, a small one-ohm resistor was added to the P-gate of the MOSFET. This slows the rise and fall times of the switch and can greatly reduce these voltage spikes [1]. In addition to this a small

RC snub circuit was implemented in parallel to the Schottky diode to further reduce the noise. This allows the inductor's current to discharge in the capacitor during switching which minimizes voltage spikes. The PCB layout of the circuit was also another critical aspect. Adequate placement was ensured to decrease EMI problems and excess switching noise.

2.1.3 Audio

In order for compatibility with standards of the music industry, M.E.L.O.D.I.C will act as a wireless audio cable with line-level inputs and outputs. We will be using Texas Instruments' TLV320AIC3204 Ultra Low Power Stereo Audio Codec to convert our analog audio into digital audio. This is done through the use of the codec's built-in DAC. The stereo audio DAC provided by the codec supports data rates ranging from 9khz to 198khz [2]. We choose to use PCM-16 at a sampling rate of 48 kHz to ensure that we reach the standards of CD-quality audio.

I2C The host, whether it is the CC8530 in autonomous mode or controlled by the MCU will configure the needed codec registers for operation. These registers control things such as input impedance, audio format, sampling frequency, and power consumption, to name a few. [2] In order to ensure it is configurable and usable with any type of device, we will have 2 different pin-outs for the digital transmission protocols.

- I2C: To configure the codec registers, an I2C protocol is used. (SCL, SDA)
- Digital Pins: To reset the codec. (RES_CODEC)

Power The codec features three different power supply pins. These pins are the digital blocks source and drain voltages, the analog block source and drain voltages, and the IO source and drain voltages. These pins require a 1.8V input, however we chose to use the 3.3V supply and step it down to 1.8v using the internal buck converter. In order to utilize the universal 3.3V supplied to the codec, the source pins for the digital and analog supplies are used as filtering outputs with each filtering output tied to ground through 10 uF and 1 uF capacitors in parallel. This allows the internal buck converters in the codec to step down the 3.3V to 1.8V without needing an external buck converter on the board.

Clocks In order for us to use the codec with the transceivers master clock of 10.2475 MHz, the codec's clock divider must be set to certain values to ensure 48 KHz sampling rate. The clock divider tree is shown in figure 4. The following clock divider register values are:

- NDAC/ADC = 1
- MDAC/ADC = 2
- DOSR/AOSR (Oversampling Rate) = 64

These values ensure that the ADC and DAC have a 48KHz sampling rate.

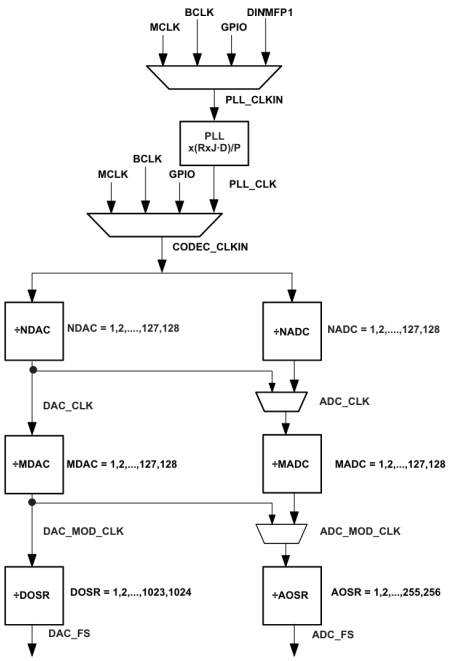


Figure 4: Clock Divider Tree

Digital Filter Another consideration we had to make was implementing a DC filter to cutoff the DC offset from the ADC input. The transfer function of the built in device filter is:

$$\frac{n_o + n_1 z^{-}1}{2^{23} - d_1 * z^{-}1} \tag{1}$$

Where $n_o = 32767 * 256$, $n_1 = -32767 * 256$, and $d_1 = 32768 * 256 * (1 - 2^{13})$. This gives a digital high-pass filter with cutoff at approximately 1 Hz. In order to utilize the filter, we chose to use the

PRB_P8 mode. The block diagram for the PRB_P8 mode is shown in figure 5. This mode uses the interpolation filter B whose frequency response is shown in figure 6.

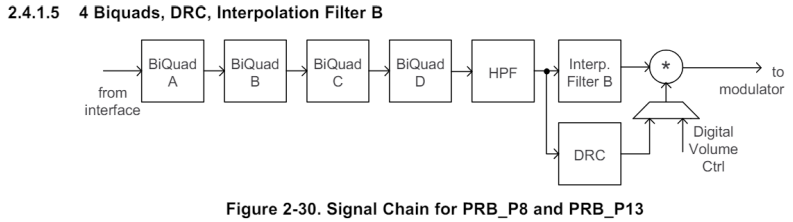


Figure 5: Digital Signal Processing Block Diagram for PRB_P8

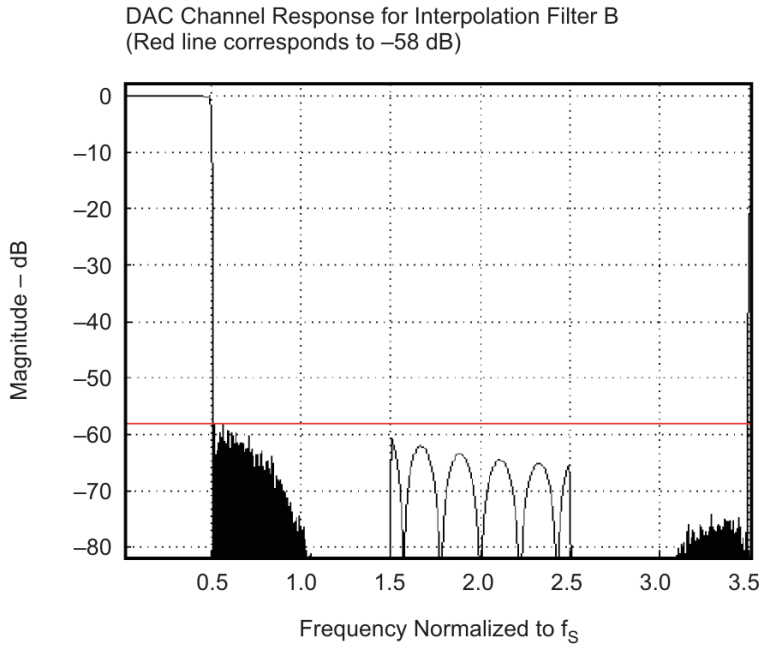


Figure 6: Interpolation Filter B Frequency Response

The I2C psuedo-code used to program the codec is shown in the appendix.

2.1.4 RF

The RF subsystem mainly comprises of the CC8530RHAT Pure-Path Wireless SOC and the Murata LFB182 Matching/Balun monolithic circuit. The RF subsystem is also be responsible for programming the codec through it's internal MCU and I2C interface.

Digital Design The digital back-end features pin-outs that are designed to mate with the other board sub-systems through jumper cables. This design choice was made so that the CC8530RHAT can work in autonomous mode, host-controlled mode and in production test mode [3] without needing significant board redesigns for prototyping, characterization and final production of the device. The chosen digital protocols that have pin-outs are:

- I2C: Used for control of the codec when in autonomous operation. The I2C lines will have the internal pull-up resistors required for I2C operation. (SCL, SDA)
- SPI: Used to flash the transceiver when in autonomous operation, configure and read registers when in host-controlled operation, and run included production code in production test. When the device operates in host controlled mode, and additional interrupt request pin is needed to interrupt SPI requests to and from the MCU [3]. (SCK,MOSI,MISO,CSn,IRQ)

In order to negate RF noise on the GPIO and I2C traces, 6.8 nH inductors were added in series. The I2C trace also have a 2.2 kΩ pull-up resistors in shunt with the traces as per I2C electrical requirements and 6.8 nH inductors to negate the RF noise. The I2S interface did not have pin-outs since the digital audio interface connections are the same for both host controlled and autonomous mode.

RF The CC8530 sees an optimum differential impedance of 70 + j30 Ω. The balun we chose, however, is a conjugate match to this optimum load impedance. This required the differential traces to have an impedance of $Z_{diff} = 70\Omega$ to ensure we have a reflection-less match. With the FR-4 board stack-up as shown in figure 7, the required board trace were found using an online calculator. The values for a trace with a 70 Ω differential impedance is shown in figure 8.

F.Cu	Copper			0.035 mm
Dielectric 1	PrePreg	FR4	...	0.1 mm
In1.Cu	Copper			0.035 mm
Dielectric 2	Core	FR4	...	1.24 mm
In2.Cu	Copper			0.035 mm
Dielectric 3	PrePreg	FR4	...	0.1 mm
B.Cu	Copper			0.035 mm

Figure 7: Four Layer Board Stack-up

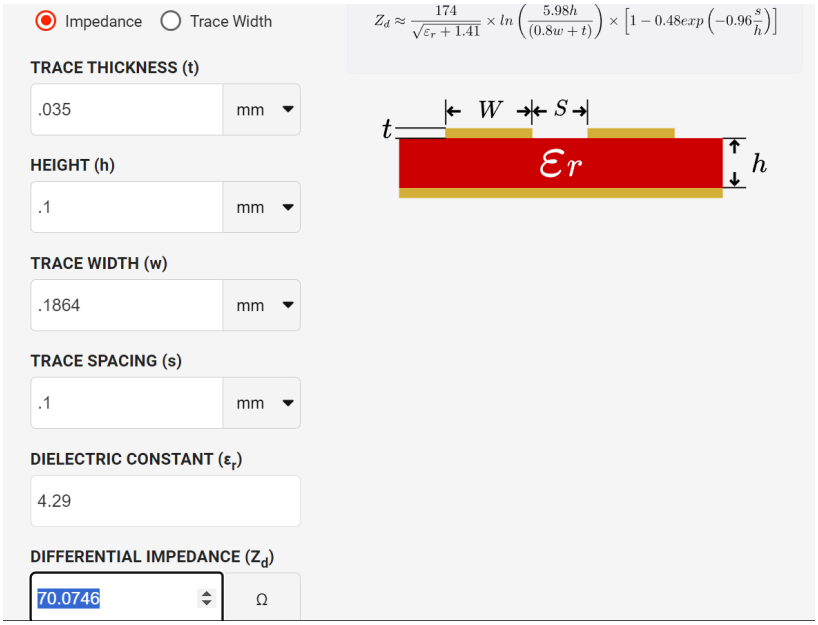


Figure 8: Differential Trace Geometry

Similarly, the balun has a 50Ω single-ended optimum load impedance. Since the antenna we are using has a 50Ω impedance as-well, the single ended line needed a characteristic impedance of 50 Ω. The geometry of this trace with the given board stack-up is shown in figure 9.

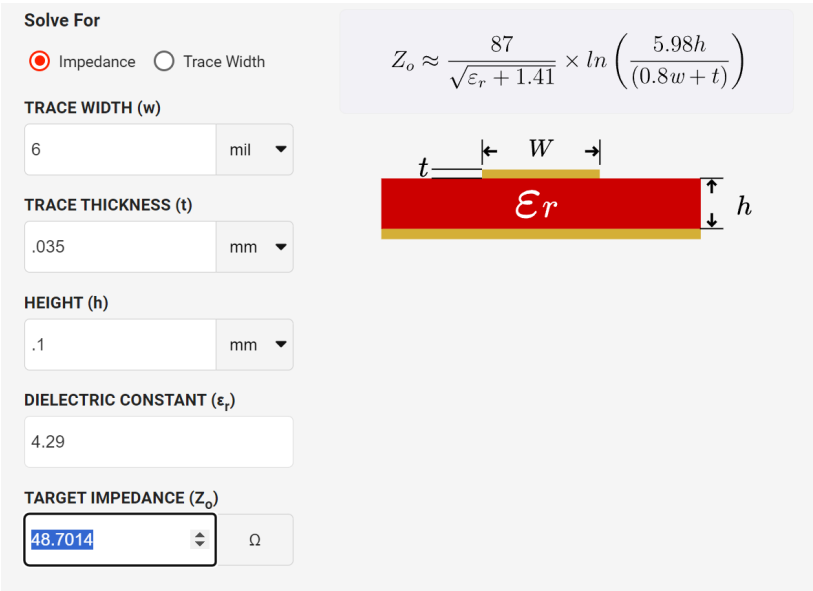


Figure 9: Single-Ended Trace Geometry

Another addition that was made was an exposed copper pour with stitching vias to the inner ground layer fencing the RF section from the other digital and analog traces on the board.

Power The supplies for both analog, digital and IO have decoupling capacitors. To negate high-frequency interference from the RF section, ferrite beads were placed in series with the analog and digital supplies. Since the codec and the transceiver are tied to the same IO supply, the ferrite bead was not added.

3 Results

3.1 Control

Our initial requirements and verifications are shown in the table below:

Feature	**Verification Actions**
Micro-controller successfully programs and controls the codec and digital radio SOC's.	1. Use static code analysis tools to ensure that the micro-controller code is error-free and adheres to coding standards. 2. Develop test cases to verify individual functions of the micro-controller related to codec and digital radio control. 3. Measure and verify the data transfer rate and integrity of communication between the micro-controller and the codec and digital radio SOC using a logic analyzer.
LCD shows all necessary info.	1. Conduct usability tests to ensure all necessary information is displayed clearly and is easily readable. 2. Verify the LCD refresh rate and response time to ensure real-time updates without lag.
Buttons handle GUI navigation in the user interface.	1. Perform end-to-end testing to ensure buttons navigate to the correct screens or perform the correct actions. 2. Test for button durability and responsiveness under various conditions.

Table 1: Control/Status Verification Table

While we were not able to implement buttons, our final status operations showed all necessary info through the Pure-Path Wireless Commander.

- Microcontroller wrote to the CC8530, but couldn't read. Verified by sending data to the CC8530 and seeing LED result.

- LCD turned on and displayed data, but couldn't display necessary data from the chip (no

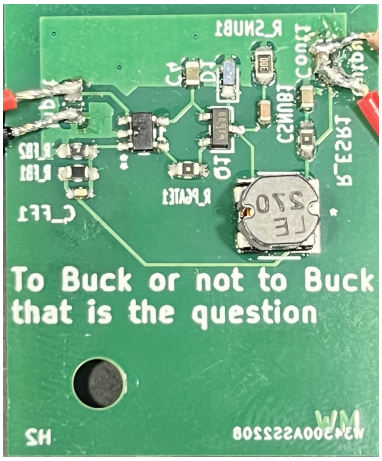


Figure 10: Physical Buck Converter

read capability). LEDs replaced the LCD for showing pairing status.

- Buttons were replaced with Purepath Wireless SPI commands, successful pairing confirmed by reading network statistics mentioned in Design section.

3.2 Power

Our initial goals for the power subsystem are listed below in *Table 2*.

Requirement	**Verification Method**
1. The buck converter must provide stable 3.3V from the 9V battery.	Tested by voltmeter to ensure voltage stability under various load conditions.
2. Ripple and noise on 3.3V line must be less than 50 mV p-p.	Use an oscilloscope to measure ripple and noise on the power line.
3. Power conversion efficiency must be at least 85% under full load.	Measure efficiency under various loads using a power analyzer.
4. The battery must last a minimum of 5 hours with adequate charge.	Tested by operating the device from a full charge under normal conditions until the battery is depleted, ensuring a minimum operational time of 5 hours.

Table 2: Power Management Verification Table

Because of delays in parts and PCB orders the buck converter was never integrated on the same PCB as the rest of the subsystems but left on its own as can be seen in *fig. 10*. Despite never receiving the final PCB design the standalone buck this subsystem behaved well and operated in the end without much error. The verification of each requirement is described below:

3.2.1 Requirement 1:

Three buck converters were created in total. Because of slightly differing resistances in the feedback network and other parasitic every device had a slightly different value but stayed constant $3.3V \pm 2\%$. Each of these was load-tested from 1 to 100mA. Their voltage remained constant throughout this entire range and beyond. The input voltage was also swept and the voltage remained constant from 11V-3.3V.

3.2.2 Requirement 2:

One of the problems with the initial PCB board design that we had to use in the end was that it only had one 0402 pad for the output capacitor. Two or more were needed to achieve the $100\mu F$ specified in the design. The maximum capacitance we could achieve with one pad was $47\mu F$. The switching frequency is dependent on the capacitance and this lowered capacitance increased the switching frequency which also increased noise and voltage spikes. Because of an additional $47\mu F$ was placed on the output voltage pad in between the 3.3V and ground wire. This led to a more stable voltage and reduced the mean noise level to 37mV.

3.2.3 Requirement 3:

Power efficiency testing was performed by using several different valued resistors, measuring the voltage across them, and reading the input voltage and current from a DC voltage generator. The input and output were then calculated and compared. Each converter operated distinctly. The first converter constructed was able to consistently achieve an efficiency of 82.5%. The second that was made achieved a maximum relatively constant efficiency of 63%. The thrid behaved much differently. It's efficiency varied from 20% to 85% depending on the input voltage with the worst efficiency at 6V.

3.2.4 Requirement 4:

Instead of the previously mentioned verification method of waiting for 5 hours with the device running, we simply measured the maximum and mean draw from the MELODIC device as well as the efficiencies for the converter described in the section above. We found the nominal current draw

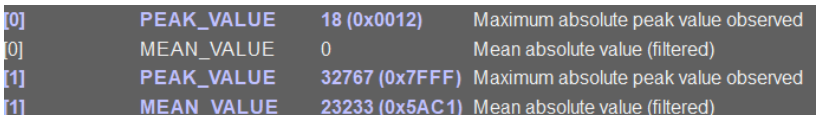


Figure 11: PS_AUDIO_STATS register

from the MELODIC is 30mA, which is significantly lower than the design current. This results in a 12mA draw from the battery at 100% efficiency or 18.3 at a more realistic 60%. This would give 30 hours of battery life from the standard 550mAh 9V battery.

3.3 Audio

Our initial goals for the power subsystem are listed below in *Table 3*.

Requirement	**Verification Method**
Codec must support 48 kHz sampling rate and 16-bit depth.	Use NWM_GET_STATUS_S to see the sampling rate and PS_AUDIO_STATS to see if max value reads a 16-bit word.
Input and output impedance must be 10kΩ and 600Ω, respectively.	Read the I2C register to see if register is set correctly
End-to-end latency must not exceed 20 milliseconds.	Use NWM_GET_STATUS_S on slave to get latency

Table 3: Audio Requirements Verification Table

To test the audio, we started a protocol link between slave and master, with the slave connected to a laptop through a debugger and the master connected to a laptop through a 3.5mm stereo jack. Using the PS_AUDIO_STATS register, we were able to check if the samples were being transmitted by watching the max and mean values of the samples change as the input samples volume changed. The test worked as expected, so the audio section did it's job of sampling and transmitting the signals to the transceiver. A sample of the max and mean values for an audio stream is shown in figure 11.

- We were able to successfully transmit 44.1khz/16-bit audio from the master device to the slave, fulfilling the requirement of CD-quality audio. The codec supported 48khz sampling rate, but this was unnecessary for our final design.
- I2C registers were read correctly because we were able to successfully program the TLV320AIC3204 codec through the CC8530's MCU.

- End-to-end latency measured at 20ms, confirmed from NWM_GET_STATUS

Aside from the specific requirements mentioned in the R&V table, unfortunately we were not able to actually hear the audio transmitted on the slave device. This is due to our PER being higher than expected. The codec automatically mutes the channel when the PER is too high, leading to the audio not playing. We believe this is due to our inexperience in RF design. With more time, some advice from experienced RF engineers, and another PCB revision we may have been able to fix this issue.

3.4 RF

Since the project seeks to fix issues with interference in the ISM band, the initial requirements for the RF section to ensure that the adaptive frequency hopping works are *Table 4*.

Feature	**Verification Actions**
AFH successfully negates the effects of frequency dependent interference from co-existing ISM protocols.	1. Setup a single link in an enviroment with other co-existing ISM protocols. 2. Using the PS_RF_STATS register, calculate the ratio between packets failed and packets attempted 3. If the ratio is greater than 25%, then AFH has failed.

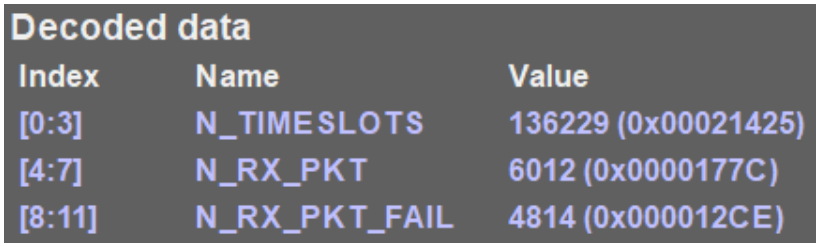
Table 4: RF Verification Table

Test The test environment, the ECEB mezzanine as shown in figure 13, was chosen since the need for ISM coexistence was assumed. This assumption was based on student wearing Bluetooth headphones which was continuously transmitting in the ISM band. The test setup is shown in figure 12. Both the devices were programmed with autonomous firmware for both protocol master and slave. The protocol master test board was connected to a laptop through a stereo 3.5mm headphone jack. A 1000 Hz test audio was played through the laptop, with the slave receiving these samples. In order to monitor network statistics, an SPI debugger specific to the CC85xx series of RF SOC's was connected to the slave board so results could be monitored with another laptop.

Figure 12: Test setup for two MELODIC RF boards

Figure 13: Test environment

As seen in figure 14, after 2-3 seconds of a wireless stream, we had a PER of $\tilde{8}2.5\%$. This was the root of failure for the project, as this PER meant that the samples would be muted due to the implemented hysteretic muting.



Decoded data

Index	Name	Value
[0:3]	N_TIMESLOTS	136229 (0x00021425)
[4:7]	N_RX_PKT	6012 (0x0000177C)
[8:11]	N_RX_PKT_FAIL	4814 (0x000012CE)

Figure 14: PS_AUDIO_STATS register

Issues In order to debug the issue, multiple tests were conducted on the board to isolate the issue. First, all the digital signals were tapped to see if the devices are transmitting and receiving the expected waveform for I2S and I2C. Since these digital protocols were functioning as expected, we decided to run another RF test and use the PS_RF_STATS register on the slave to see if the samples maximum values would change proportional to the laptop's volume. In our case this worked as-well, which further isolated our issue to the RF section on the board. Since we did not have a proper test setup to measure the RF section and were inexperienced in de-embedding cascaded S-parameters between differential and single ended lines, we couldn't properly measure the impedance over frequency of the RF traces. Solving this issue would require a complete board re-design, as well as proper simulations and test setups, which we did not have due to our lack of knowledge, working RF test equipment in the senior design lab, and time.

4 Ethics and Safety Considerations

4.1 FCC Regulations

One concern with designing a digital wireless communication system was aligning with FCC regulations and the FCC band plan. If we were to design our system around the same frequencies as existing systems, we would need to license our device to work in that band. Specifically, UHF and VHF equipment used in the live audio industry either have privately licensed bands with the FCC or are licensed around the same bands as terrestrial television [4]. Since we are using an RF SOC specifically made for wireless digital audio streaming, and it has already been tested and approved for use in the ISM band, we do not need to worry about FCC licensing.

4.2 Environmental Concerns

One concern with using a 9V battery is the potential environmental damage that it might cause when it is thrown away. We used Alkaline 9V batteries which are a safer alternative to lithium ion batteries, which are known to have a greater negative impact on the environment when thrown away.

4.3 Safety Concerns

We ensured that the device itself was safe to use before we demoed the project through thorough analysis of the device's power consumption. Mainly, all components (resistors, capacitors, SOCs, etc.) were within their allowed power consumption tolerance.

5 Conclusions

Throughout the project our team successfully tackled challenges related to wireless audio transmission interference from existing wireless protocols and integrating various electronic components. Future improvements could focus on enhancing the user interface and expanding the devices compatibility with a wider range of audio equipment. Despite difficulties such as integrating the STM32 microcontroller for SPI commands we were able to adapt using the Purepath Wireless software and deliver a successful project. Ultimately we believe that MELODIC has the potential to be a great solution to small artists looking for a wireless alternative to their current cable-based setup.

A Code

A.1 Arduino Code

```
void sendCMD_REQ(byte commandType, byte params[], int paramSize) {
  digitalWrite(PA15, LOW); // Start communication

  int SW;
  // Wait until MISO goes high
  if(digitalRead(PB4)==HIGH){

          // Read status word
  byte statusHigh = SPI_1.transfer(0x00);
  byte statusLow = SPI_1.transfer(0x00);

  int SW = (statusHigh <<8) | statusLow;

  // Send command type (assume '11' for the first two bits)
  SPI_1.transfer(0xC0 | commandType);

  // Send number of parameters
  SPI_1.transfer(paramSize);

  // Send parameters
  for (int i = 0; i < paramSize; i++) {
    SPI_1.transfer(params[i]);
  }



  }
```

```
  digitalWrite(PA15, HIGH); // End communication
  lcd.print(SW);
  delay(2000);
  lcd.clear();
}

int sendREADBC() {
  digitalWrite(PA15, LOW); // Start communication

  // Wait until MISO goes high
  while (digitalRead(PB4) == LOW);

  // Command to read byte count, assuming the command identifier is '0xA'
  SPI_1.transfer(0xA0);

  // Read status word
  byte statusHigh = SPI_1.transfer(0x00);
  byte statusLow = SPI_1.transfer(0x00);

  //SW = (statusHigh <<8) | statusLow;
  Serial.print("Status: ");
  Serial.println((statusHigh << 8) | statusLow, HEX);

  // Read number of bytes available
  byte numHigh = SPI_1.transfer(0x00);
  byte numLow = SPI_1.transfer(0x00);
  int numBytes = (numHigh << 8) | numLow;
  Serial.print("Number of bytes: ");
  Serial.println(numBytes);
```

```
  // Read the actual data bytes
  Serial.print("Data: ");
  for (int i = 0; i < numBytes; i++) {
    byte dataByte = SPI_1.transfer(0x00);
    Serial.print(dataByte, HEX);
    Serial.print(" ");
  }
  Serial.println();

  digitalWrite(PA15, HIGH); // End communication

  return numBytes;
}
```

B SPI Commands for Pairing/Reading Stats

Command	NWM_CONTROL_ENABLE		
Description	Used on protocol masters to enable/disable formation/maintenance of the audio network. Advertisement for pairing will be reset when disabling the network.		
SPI Operations	CMD_REQ(0x0C, 2, pars, sw)		
(pars)	**[Byte][bit] index**	**Field**	**Description**
	[0]	–	Reserved, write 0
	[1][7:1]	–	Reserved, write 0
	[1][0]	WM_ENABLE	Enable (1) or disable (0) formation/maintenance of network.
Related events	-		
Usage limitations	Only available on host-controlled protocol masters		
Notes			

Figure 15: SPI Command for Enabling Network on Master Device

Command	NWM_DO_SCAN		
	[15:12] [26+28*i*:27+28*i*] [11:0]	LATENCY[11:0]	Audio latency in audio network (in samples).
Related events			
Usage limitations	Only available in host-controlled protocol slaves		
Notes	Depending on the number of networks found, the number of returned bytes will be N * 28, where N is between 0 and SCAN_MAX. The parameters to this command were changed in FW 1.3.0 (two bytes were added).		

Figure 16: SPI Command for Scanning for Network on Slave Device

Command	NWM_DO_JOIN		
Description	Used by protocol slaves to join a specific PurePath Wireless network or the first found that matches the specified criteria. The command is also used to leave a network.		
SPI Operations	CMD_REQ(0x09, 18, pars, sw)		
(pars)	**[Byte][bit] index**	**Field**	**Description**
	[0:1][15]	–	Reserved, write 0
	[0:1][14:0]	JOIN_TO[11:0]	Timeout in increments of 10 ms for joining network.
	[2:5]	DEVICE_ID[31:0]	Network ID of network to join (device ID of master) A value of 0x00000000 means leave network if currently connected A value of 0xFFFFFFFF means join any network where master is currently attempting to pair with a new slave
	[6:9]	MANF_ID[31:0]	Manufacturer ID filtering criteria. A value of 0 means ignore manufacturer ID of master, any other value requires that the protocol master has a matching manufacturer ID.
	[10:13] [14:17]	PROD_ID_MASK[31:0] PROD_ID_REF[31:0]	Product/family ID mask and reference value to be used as filtering criteria for whether to join network. Only networks where PROD_ID&PROD_ID_MASK == PROD_ID_REF&PROD_ID_MASK for masters with PROD_ID will be joined. Thus a PROD_ID_MASK value of 0 will disable filtering based on product ID.
Related events	EVT_NWK_CHG – will be signaled when a network connection is lost – or if this DO_JOIN command fails		
Usage limitations	Only available on host-controlled protocol slaves		
Notes	The host controller should use NWM_GET_STATUS to determine the outcome of the command and to store the network ID for future pairing operations if required. The command will use ~500 ms to scan the entire RF channel space once. Using a JOIN_TO value below 50 (= 500 ms) is not recommended.		

Figure 17: SPI Command for Joining Network on Slave Device

Command	NWM_GET_STATUS		
			network
	[12][0]	–	Reserved, ignore value
	[13:14]	ACH_SUPPORT[15:0]	Bitmask indicating which audio channels the protocol master supports. The bit indexes correspond to the logical audio channel indexes defined for NWM_ACH_SET_USAGE.
	[15][7]	ACH_ACTIVE[0]	Flag that indicates whether audio channel at spatial location 0 is currently in use in audio network
	[15][6:4]	ACH_FORMAT[0][2:0]	Audio type used by audio channel at spatial location 0 0: unused audio channel 1: PCM16 2: PCME24 4: SLAC 5: PCMLF
	[15][3]	ACH_ACTIVE[1]	Flag that indicates whether audio channel at spatial location 1 is currently in use in audio network
	[15][2:0]	ACH_FORMAT[1][2:0]	Audio type used by audio channel at spatial location 1
			...
	[22][3]	ACH_ACTIVE[15]	Flag that indicates whether audio channel at spatial location 15 is currently in use in audio network
	[22][2:0]	ACH_FORMAT[15][2:0]	Audio type used by audio channel at spatial location 15
	[23]	RSSI[7:0]	Averaged RSSI of recent packets received from master (signed 2s complement, 0 → ~0 dBm)
	[24:25][15:12]	–	Reserved, ignore value
	[24:25][11:0]	SAMPLE_RATE[11:0]	Sample rate (in increments of 25 Hz) as reported by master
	[26:27][15:12]	NWK_STATUS	Current network status of protocol slave (this node): 0: No network connection 8/13: Connected to network other: Searching/tracking/joining network
	[26:27][11:0]	LATENCY[11:0]	Audio latency in audio network (in samples).
	[28:29]	ACH_USED[15:0]	Bitmask providing information about which audio channels are currently used by slave. The bit indexes correspond to the logical audio channel indexes defined for NWM_ACH_SET_USAGE.
	On protocol master: READBC(sw, nBytes, data)		
(data)	**[Byte][bit] index**	**Field**	**Description**
	[0][7:4]	–	Reserved, ignore value
	[0][3:0]	NWK_STATUS	Current network status of protocol master: 0: No network maintained other: Maintaining network
	[1:2][15:12]	–	Reserved, ignore value
	[1:2][11:0]	SAMPLE_RATE[11:0]	Current sample rate (in increments of 25 Hz)
	[3:4]	ACH_USED[15:0]	Bitmask indicating which audio channels is consumed/produced by any protocol slaves.
	For each connected slave, *i* =[0, (nBytes-5)/16-1]. nBytes=5 means no protocol slaves connected.		
	[5+16*i*:8+16*i*]	DEVICE_ID[31:0]	Device ID of slave
	[9+16*i*:12+16*i*]	MANF_ID[31:0]	Manufacturer ID reported by protocol slave
	[13+16*i*:16+16*i*]	PROD_ID[31:0]	Product/family ID reported by protocol slave
	[17+16*i*:18+16*i*]	ACH_USED[15:0]	Bitmask indicating which audio channels the protocol slave is consuming/producing. The bit indexes correspond to the logical audio channel indexes defined for NWM_ACH_SET_USAGE.
	[19+16*i*]	–	Reserved, ignore value
	[20+16*i*][7:4]	–	Reserved, ignore value
	[20+16*i*][3:1]	SLAVE_SHORT_ID	Short ID for referencing a protocol slave (1-7)
	[20+16*i*][0]	WPS_DSC_EN	If set, indicates that protocol slave supports a data side-channel.
Related events	–		
Usage limitations	Available on both host-controlled protocol masters and slaves but returned data must be interpreted differently		
Notes			

Figure 18: SPI Command for Reading Network Stats

Command	PS_AUDIO_STATS		
Description	Requests and returns audio statistics gathered since the last PS_AUDIO_STATS command or chip reset. The counter values for processed/concealed/muted samples and mute events are only valid for locally consumed audio channels. The counters are reset to zero after running the command. If not polled regularly, the counters will saturate (at 0xFFFFFFFF for 32-bit values and 0xFFFF for 16-bit values). The peak and mean values are valid for both produced and consumed audio channels.		

SPI Operations	`CMD_REQ(0x11, 0, sw)`
	
	`READBC(sw, nBytes, data)`		

(data)	**[Byte][bit] index**	**Field**	**Description**
	[0:3]	SMPL_PROCESS_COUNT16	Number of processed samples/16 contributing to the counter values below.
	[4:7]	SMPL_CONCEAL_COUNT16	Number of samples/16 where error concealment was active (due to missing odd audio slices for PCM16/PCME24).
	[8:11]	SMPL_MUTED_COUNT16	Number of muted samples/16 where muting was active (due to missing critical audio slices)
	[12:13]	MUTE_EVENT_COUNT	Number of mute events, i.e. number of transitions from unmuted to muted audio outputs.
	[14]	CH_COUNT	Number of channels in the peak and mean value statistics output
	[15]	-	Reserved, ignore value
		For each channel, *i* =[0, ((nBytes-16)/4)-1].	
	[16+4i:17+4i]	CH_PEAK_VALUE	Peak absolute value
	[18+4i:19+4i]	CH_MEAN_VALUE	Mean absolute value (filtered)
Related events	-		
Usage limitations	Available in host-controlled and autonomous operation		
Notes	For the period between chip reset and initial audio streaming, there are muted samples, but no mute event. For the MICROPHONE_0 to MICROPHONE_3 logical audio channels, mute events are counted separately for each.		

Figure 19: SPI Command for Reading Audio Stats

Command	PS_RF_STATS		
Description	Requests and returns RF statistics gathered since the last PS_RF_STATS command or chip reset. All counters are reset to zero after the command.		

SPI Operations	`CMD_REQ(0x10, 0, sw)`
	
	`READBC(sw, nBytes, data)`		

(data)	**[Byte][bit] index**	**Field**	**Description**
	[0:3]	TIMESLOT_COUNT	Number of timeslots contributing to the statistics output
	[4:7]	RX_PKT_COUNT	Number of packet receptions attempted
	[8:11]	RX_PKT_FAIL_COUNT	Number of packet receptions that failed
	[12:15]	RX_SLICE_COUNT	Slave: Number of slices received Master: Number of slices transmitted
	[16:19]	RX_SLICE_ERR_COUNT	Slave: Number of slices received with errors Master: Reserved, ignore value
	[20]	NWK_JOIN_COUNT	Slave: Number of successful network join Master: Number of successful slave join
	[21]	NWK_DROP_COUNT	Slave: Number of network drops Master: Number of slave drops
	[22:23]	AFH_SWAP_COUNT	Slave: Reserved, ignore value Master: Number of times the adaptive frequency hopping algorithm swapped out an active RF channel.
		For each RF channel *i* = [0,19]	
	[24+2i:25+2i]	AFH_CH_USAGE_COUNT	Number of times RF channel *i* is used
Related events	-		
Usage limitations	Available on host-controlled and autonomous devices		
Notes			

Figure 20: SPI Command for Reading RF Stats

C I2C Psuedo-code

C.1 Master Psuedo-Code

```
    # PIN RESET

p Reset 1     # Release the reset pin
```

```
# RESET
w 30 00 00    # Select register page 0
w 30 01 01    # I2C reset
# CLOCK SETTINGS
w 30 12 81    # Power up the NADC divider with value 1
w 30 13 82    # Power up the MADC divider with value 2
w 30 14 80    # Program OSR for ADC to 128
# DIGITAL INTERFACE
w 30 1B 00    # I2S, 16-bit, BCLK and WCLK are inputs
# PROCESSING BLOCK USAGE
w 30 3D 01    # Select ADC processing block PRB_R1
# ANALOG POWER SUPPLY
w 30 00 01    # Select register page 1
w 30 01 08    # Disable internal crude AVDD before powering up the internal AVDD LDO
w 30 02 01    # Enable internal analog LDO, analog blocks powered
w 30 0A 40    # Common mode set to 0.75V
# MICPGA DELAY, REFERENCE CHARGING AND HEADPHONE DE-POP
w 30 47 31    # MICPGA startup delay is 3 ms
w 30 7B 01    # Reference charging time is 40 ms
# AUDIO ROUTING
w 30 34 40    # IN1L is routed to Left MICPGA with 10K resistance
w 30 36 40    # CM1L is routed to Left MICPGA via CM1L with 10K resistance
w 30 37 40    # IN1R is routed to Right MICPGA with 10K resistance
w 30 39 40    # CM1R is routed to Right MICPGA via CM1R with 10K resistance
# DC FILTER LEFT CHANNEL
w 30 00 08    # Select register page 8
w 30 18 7F    #                n0 + n1 * z^-1
w 30 19 FF    #     H(z) = ----------------------
w 30 1A 00    #              2^23 - d1 * z^-1
w 30 1C 80    #
```

```
w 30 1D 01    # The constants are defined as
w 30 1E 00    #     n0 = 32767 * 256
w 30 20 7F    #     n1 = -32767 * 256
w 30 21 FC    #     d1 = 32768 * 256 * (1- 2^13)
w 30 22 00    # This gives a filter with cutoff at approx. 1 Hz
# DC FILTER RIGHT CHANNEL
w 30 00 09    # Select register page 9
w 30 20 7F    #                n0 + n1 * z^-1
w 30 21 FF    #     H(z) = ----------------------
w 30 22 00    #               2^23 - d1 * z^-1
w 30 24 80    #
w 30 25 01    # The constants are defined as
w 30 26 00    #     n0 = 32767 * 256
w 30 28 7F    #     n1 = -32767 * 256
w 30 29 FC    #     d1 = 32768 * 256 * (1- 2^13)
w 30 2A 00    # This gives a filter with cutoff at approx. 1 Hz
w 30 00 00    # Select register page 0
w 30 51 C0    # Power up ADC channels
w 30 52 00    # unmute ADC channels
```

C.2 Slave Psuedo-Code

```
    # PIN RESET
p Reset 1     # Release the reset pin
# RESET
w 30 00 00    # Select register page 0
w 30 01 01    # I2C reset
# CLOCK SETTINGS
w 30 0B 81    # Power up the NDAC divider with value 1
w 30 0C 82    # Power up the MDAC divider with value 2
w 30 0D 00    # Program OSR for DAC to 128 (MSB)
```

```
w 30 0E 80    # Program OSR for DAC to 128 (LSB)
# DIGITAL INTERFACE
w 30 1B 00    # I2S, 16-bit, BCLK and WCLK are inputs
# PROCESSING BLOCK USAGE
w 30 3C 08    # Select DAC processing block PRB_P8
# ANALOG POWER SUPPLY
w 30 00 01    # Select register page 1
w 30 01 08    # Disable internal crude AVDD before powering up the internal AVDD LDO
w 30 02 01    # Enable internal analog LDO, analog blocks powered
w 30 0A 40    # Common mode set to 0.75V
# MICPGA DELAY, REFERENCE CHARGING AND HEADPHONE DE-POP
w 30 7B 01    # Reference charging time is 40 ms
w 30 14 65    # HP driver power-up: 50 ms soft routing step time, 5.0 time constants, 6k resist
# AUDIO ROUTING
w 30 0E 08    # LOL routing: Left channel's DAC reconstruction filter output
w 30 0F 08    # LOR routing: Right channel's DAC reconstruction filter output
w 30 00 01    # Select register page 1

w 30 12 00    # LOL driver: Unmute, 0 dB gain
w 30 13 00    # LOR driver: Unmute, 0 dB gain
w 30 09 3C    # All output drivers powered up
w 30 00 00    # Select register page 0
w 30 3F D6    # Power up the DAC channels, normal channel routing, soft-stepping disabled
w 30 40 00    # unmute the DAC digital volume control
```

D Costs

D.1 Labor Costs

Based on the average salary of an ECE graduate from the University of Illinois, we are assuming a wage of $45/hr. We worked for 15 hours per week per person. The project design phase lasted about 10 weeks. Calculating the total labor cost: $45 * 15 * 10 * 3 = \$20250$.

D.2 Bill Of Materials

Table 5: Bill of Materials

Component	Part Number	Unit Price ($)	Quantity	Package	Total Cost ($)
		RF and Audio Board			
Capacitors 10u	0402ZD106MAT2A	0.49	10	0402	4.99
Capacitors .47u	C0402C474K8RACTU	.39	5	0402	1.95
Capacitors 1.0u	04026C105KAT2A	0.49	20	0402	9.8
Capacitors 47n	C0402C473J3RACTU	0.49	5	0402	2.45
Capacitor 2.2u	GRM155R60J225ME01J	0.49	10	0402	4.90
Audio Jacks	SJ-63053A	Free	5	THT	0
Resistors 100	RC0402FR-7W100RL	0.80	10	0402	8.0
Audio Codec	TLV320AIC3204IRHBT	6.29	5	VQFN-32	31.45
Capacitors 12p	GRM1555C1H120JA01D	0.49	5	0402	2.45
Capacitor 220p	GRM1555C1H221JA01D	0.49	5	0402	2.45
Capacitors 100n	GRM155R71A104KA01D	0.49	20	0402	11.03
Ferrite Beads	28C0236-0JW-10	0.94	10	THT	9.4
SMA Connector	0733910070	Free	5	SMD	Free
Inductors 6.8n	LQG15HS6N8J02D	0.68	25	0402	17.0

Continued on next page

Table 5 – *Continued from previous page*

Component	Part Number	Unit Price ($)	Quantity	Package	Total Cost ($)
Resistor 56k	RK73H1ETTP5602F	0.80	5	0402	4.0
Resistors 2.2k	RC0402FR-7W100RL	0.80	10	0402	8.00
Transceiver/μC	CC8530RHAR	8.24	5	VQFN-40	41.2
Crystal 48MHz	FA-128 48.0000MF20X-K0	0.32	5	FA-128	1.6
Monolithic Matching Circuit and BALUN	LFB182G45BG2D280	.396	SMT	3.96	
	Power Board				
Schottky Diode D1	DO-214AC	0.296	10	SMD/SMT	2.96
PFET	SI2343CDS-T1-GE3	0.365	10	SMD/SMT	3.65
Capacitor 100uF	GRM153R61A105ME95D	0.075	10	0402	0.75
Capacitor 22uF	C0402C220J8HACTU	0.1	10	0402	1.0
Capacitor 1uF	80-C0402C105K9PAC	0.016	10	0402	0.16
Capacitor 1nF	CL05B102KB5NFNC	0.013	10	0402	0.13
Resistor 1 Ohm	CRM1206-FW-1R00ELF	0.077	10	SMD/SMT	0.77
Resistor 30 Ohm	ESR10EZPJ300	0.077	10	SMD/SMT	0.77
Resistor 5k Ohm	RT0603BRE075KL	0.094	10	SMD/SMT	0.94
Resistor 1.6k Ohm	ERJ-UP3F1601V	0.082	10	SMD/SMT	0.82
Inductor 27uH	CR54NP-270MC	0.674	10	SMD/SMT	6.74

Continued on next page

Table 5 – *Continued from previous page*

Component	Part Number	Unit Price ($)	Quantity	Package	Total Cost ($)
Buck Controller	LM3475	0.61	10	SMD/SMT	6.1
Battery Clip	546-BS61	2.31	5	THT	11.55
		Control Board			
Buttons	474-COM-00097	0.35	15	THT	5.25
Switches	R13112ABB	2.56	5	THT	12.80
μC	STM32F103 Blue Pill Dev Boards	6.99	1	THT	6.99
LCD	HY1602E	5.33	2	THT	10.66
LCD I2C Driver	PCF8574	.74	2	SMT	1.48
				Total Cost ($)	**234.19**

D.3 Layouts and Schematics

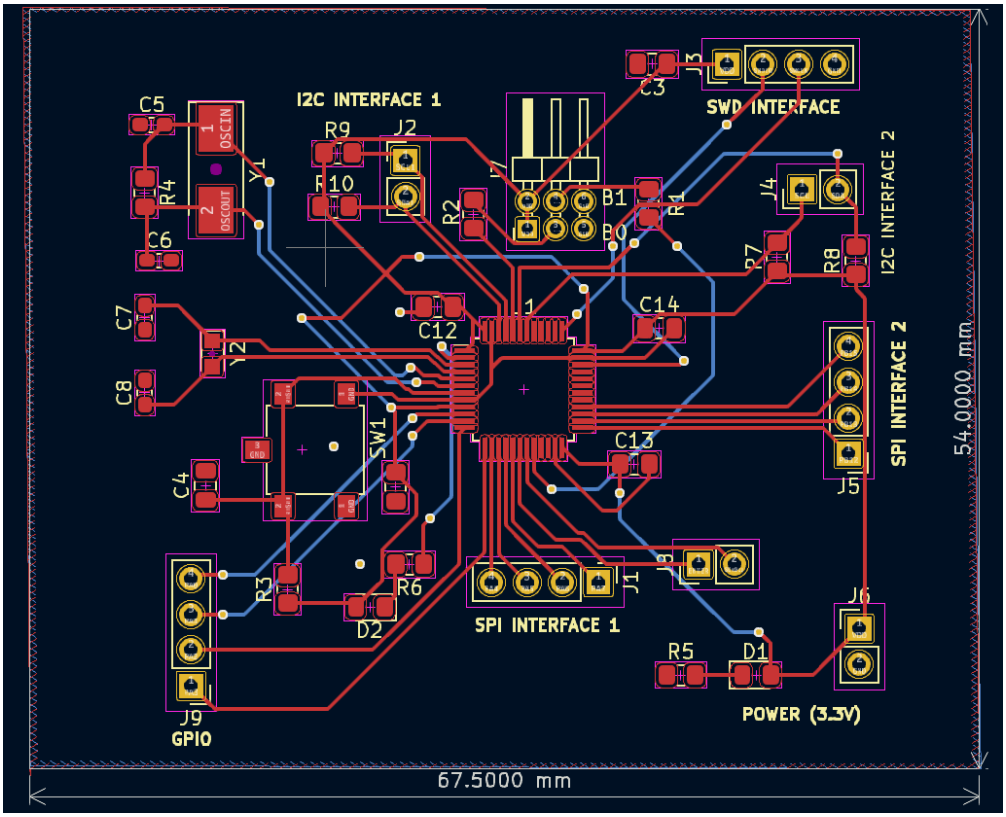


Figure 21: Control Subsytem Layout

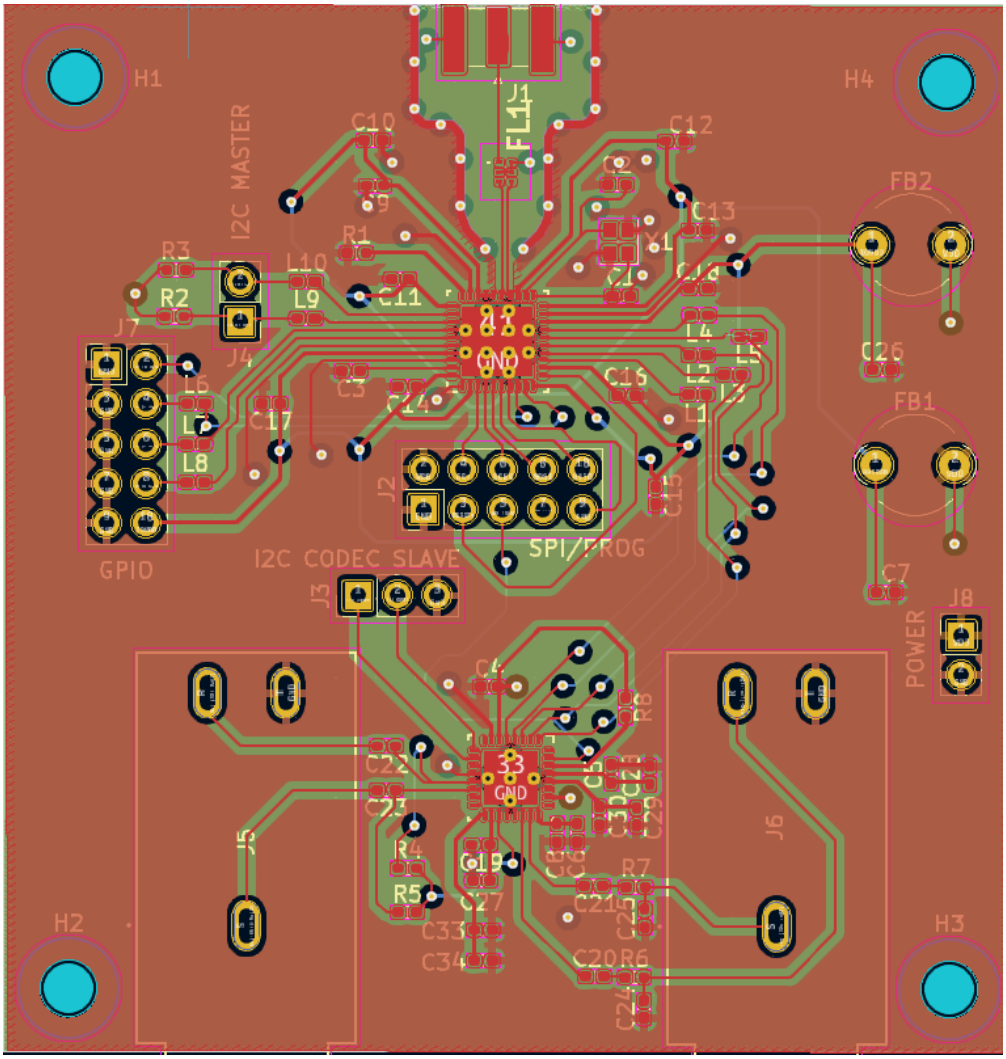


Figure 22: RF Subsytem Layout

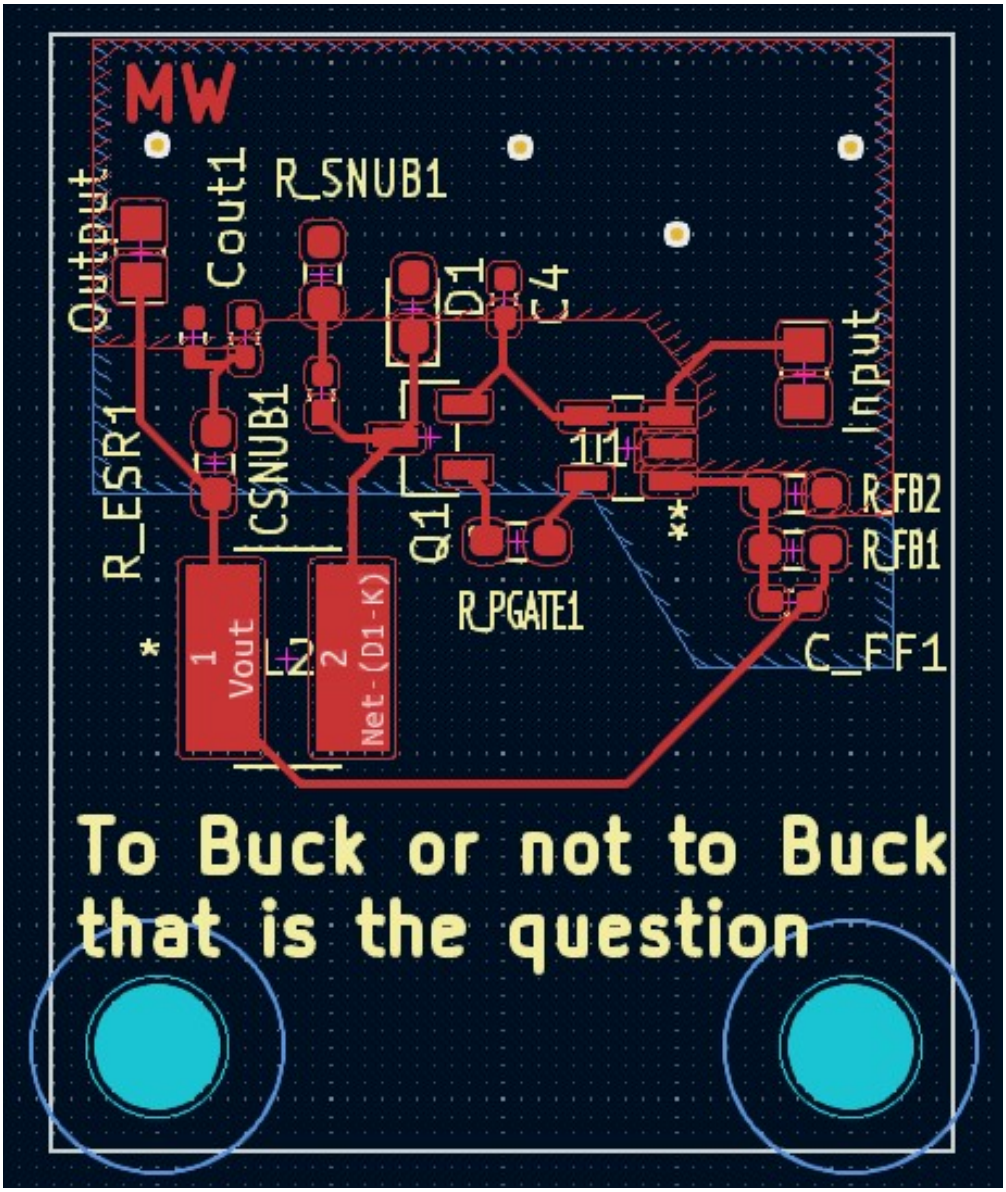
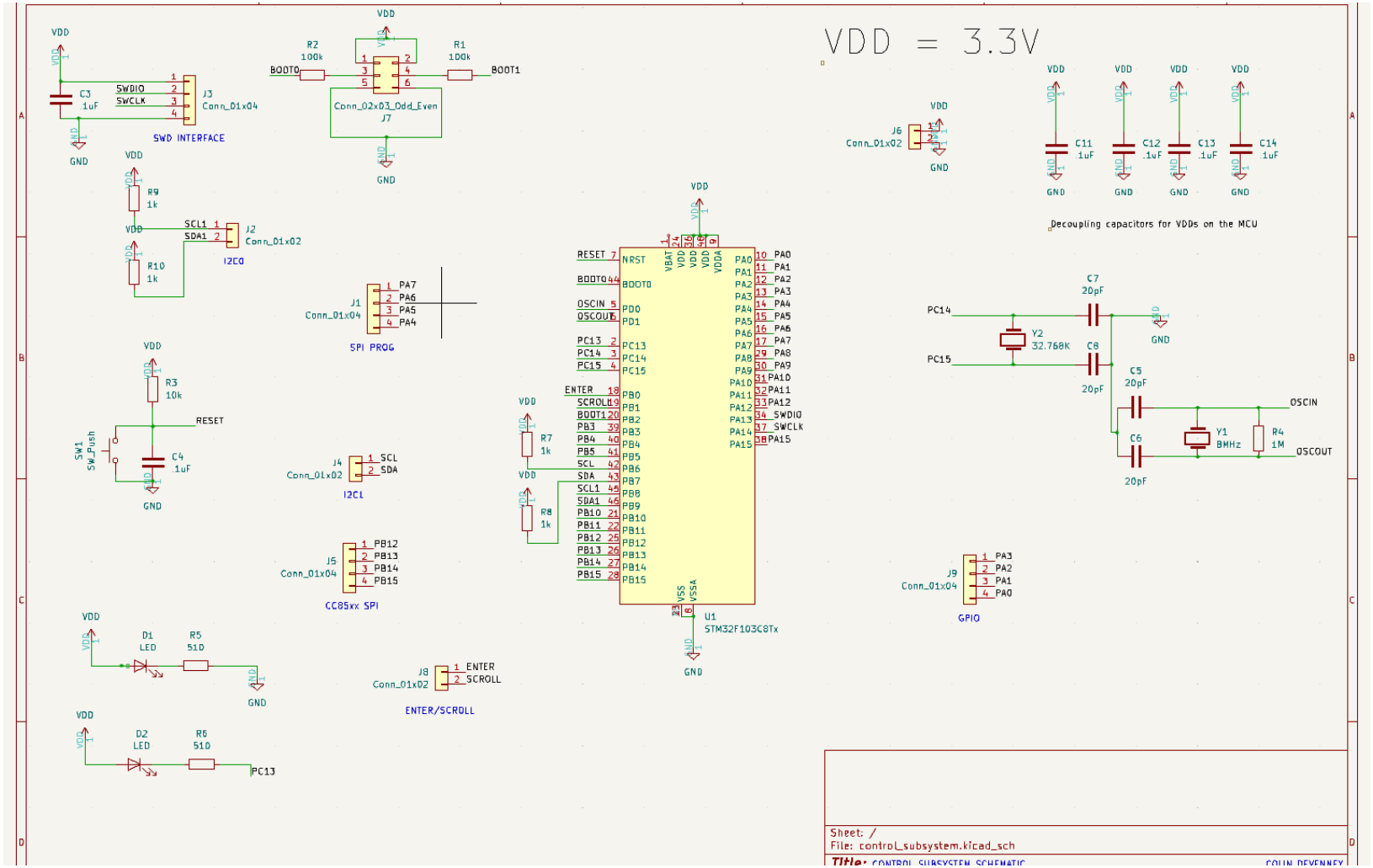
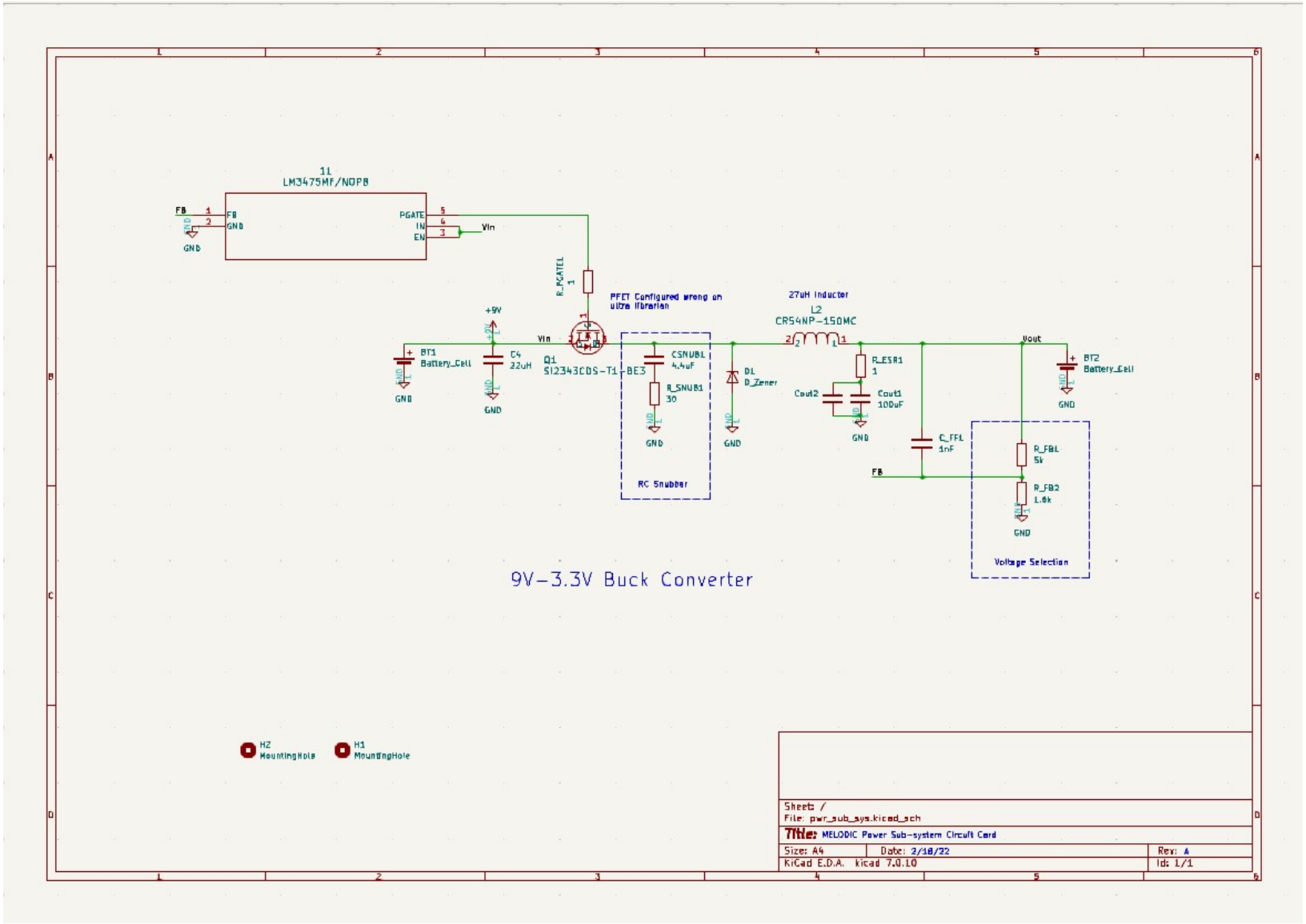
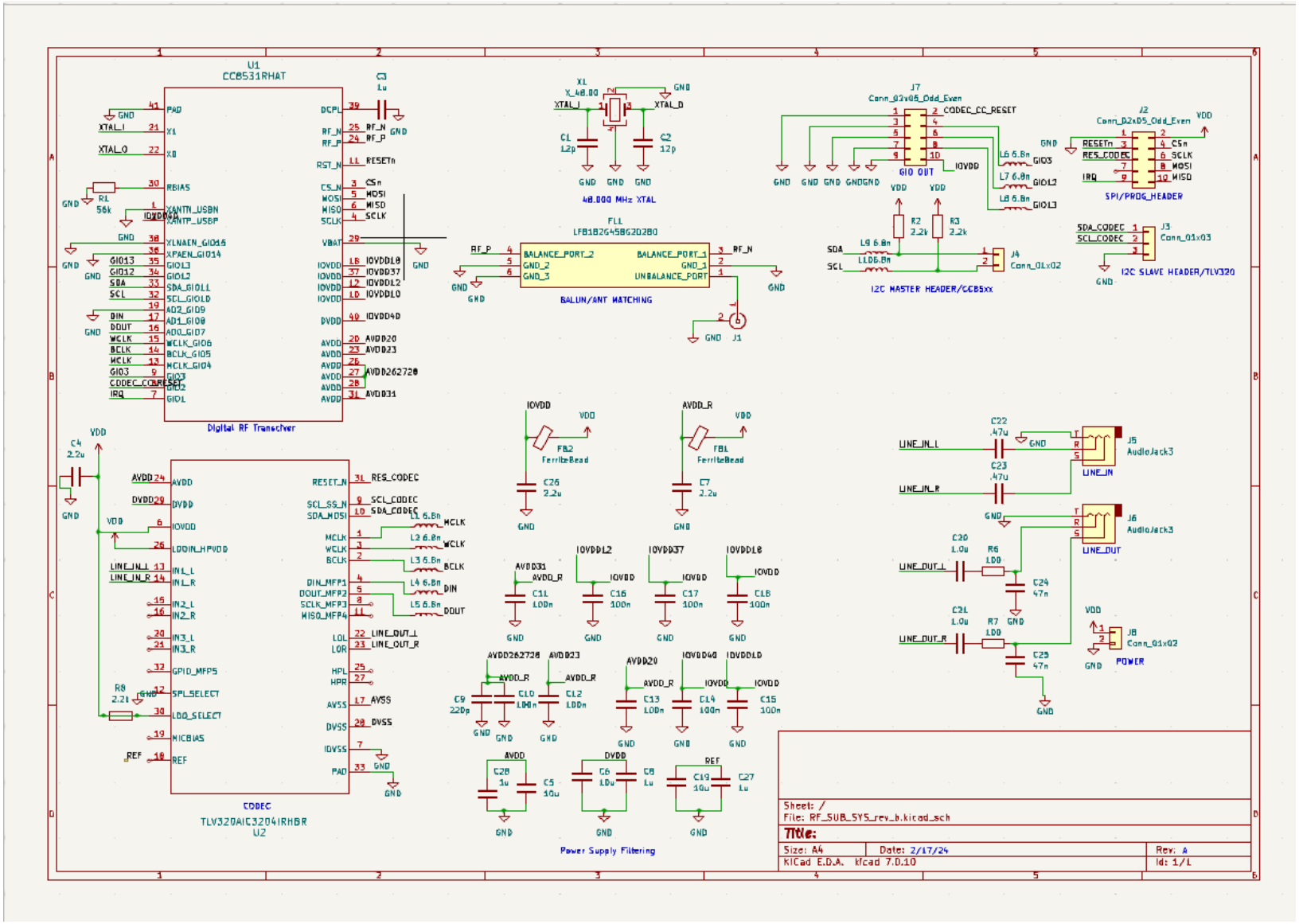


Figure 23: Power Subsytem Layout


VDD = 3.3V
SWD INTERFACE
SWDIO
SWCLK
I2C0
SPI PROG
I2C1
CC85xx SPI
ENTER/SCROLL
GPIO
Decoupling capacitors for VDDs on the MCU
STM32F103C8Tx
Sheet: /
File: control_subsystem.kicad_sch


11
LM3475MF/NOPB
FB
GND
PGATE
IN
EN
Vin
R_PGATE1
PFET Configured wrong on ultra librarian
27uH Inductor
L2
CR54NP-150MC
+9V
BT1
Battery_Cell
C4
22uH
Q1
SI2343CDS-T1-BE3
CSNUBL
4.4uF
R_SNUB1
30
DL
D_Zener
R_ESR1
1
Cout2
Cout1
100uF
Vout
BT2
Battery_Cell
C_FFL
1nF
R_FBL
5k
R_FB2
1.6k
RC Snubber
Voltage Selection
9V-3.3V Buck Converter
H2
MountingHole
H1
MountingHole
Sheet: /
File: pwr_sub_sys.kicad_sch
Title: MELODIC Power Sub-system Circuit Card
Size: A4
Date: 2/18/22
Rev: A
KiCad E.D.A. kicad 7.0.10
Id: 1/1


U1
CC8531RHAT
Digital RF Transciver
XL
X_48.00
48.000 MHz XTAL
FLL
LFB182G45BG2D280
BALUN/ANT MATCHING
J7
Conn_02x05_Odd_Even
GIO OUT
J2
Conn_02x05_Odd_Even
SPI/PROG_HEADER
I2C MASTER HEADER/CC85xx
J3
Conn_01x03
I2C SLAVE HEADER/TLV320
CODEC
TLV320AIC3204IRHBR
U2
Power Supply Filtering
J5
AudioJack3
LINE_IN
J6
AudioJack3
LINE_OUT
J8
Conn_01x02
POWER
Sheet: /
File: RF_SUB_SYS_rev_b.kicad_sch
Title:
Size: A4
Date: 2/17/24
Rev: A
KiCad E.D.A. kicad 7.0.10
Id: 1/1

References

[1] *LM3475 Hysteretic PFET Buck Controller*, Texas Instruments, Oct. 2015, sNVS239C â October 2004 â Revised October 2015. [Online]. Available: https://www.ti.com/lit/ds/symlink/lm3475.pdf

[2] *TLV320AIC3204 Ultra Low Power Stereo Audio Codec*, Texas Instruments, 2019, rev. SEPTEMBER 2019. [Online]. Available: https://www.ti.com/lit/ds/symlink/tlv320aic3204.pdf

[3] *CC85xx Family User's Guide: RF SoC for Wireless Audio Streaming*, Texas Instruments, 2013, sWRU250M â June 2013, Revised. [Online]. Available: https://www.ti.com/lit/ug/swru250m/swru250m.pdf

[4] Federal Communications Commission, "Title 47 CFR Part 15.247 - Operation within the bands 902-928 MHz, 2400-2483.5 MHz, and 5725-5850 MHz," Online, 2013. [Online]. Available: https://www.govinfo.gov/content/pkg/CFR-2013-title47-vol1/pdf/CFR-2013-title47-vol1-sec15-247.pdf