RADIO DELAY PROJECT

Alexander M. Pratt
Department of Computer Science
Brown University
alexander_pratt@brown.edu

Tengfei Jiang
Department of Engineering
Brown University
tengfei_jiang@brown.edu

David Owoade
Department of Engineering
Brown University
david_owoade@brown.edu

ABSTRACT

From a walk-off win in the World Series to a penalty shootout that determines the winner of the World Cup, sports fanatics want to hear the play-by-play call from their favorite commentators. Certain fans prefer their local or regional radio station personalities over national broadcast announcers due to their specific analysis and unique characteristics used to portray a vivid vision of events happening in a match. A drawback for radio listeners is that radio broadcasts are often ahead of television broadcasts for a variety of reasons, both technical and capitalistic in nature. As a result of this, sports fans cannot simultaneously witness the match aurally and visually. The Radio Delay Project (*RDP*) is a means to solve the misalignment of timing between both the audio and visual experience of watching athletics. *RDP* uses any audio source capable of using a 3.5mm cable and outputs audio at an adjustable delay to allow users to listen to their favorite announcers while also watching a match on television.

1 TECHNICAL OVERVIEW

1.1 SPECIFICATIONS

There are three main subsystems of *RDP*: Audio, Power, and User Interface. This section of text describes the specifications and design choices for all three subsections. The calculations used to compute values presented in this section can be found in Appendix I.

1.1.1 AUDIO

Two categories encompass the totality of audio specifications: Audio Memory Specifications and Audio Quality Specifications. The former specification defines the memory required to maintain a sample buffer sufficient in size to enable the maximum defined delay for audio playback. These specifications are found in Table 1. The latter specification defines the audio characteristics for recording and playback. These specifications are found in Table 2.

AUDIO MEMORY SPECIFICATIONS

In order to properly design the hardware and software for *RDP*, it is important to understand the amount of memory required to store a maximum delay in samples. Table 1 contains the specifications of the required memory for the system assuming the specifications listed in Table 2.

Table 1 shows the memory requirement both with and without a compression algorithm. The compression algorithm used for this computation is Quite OK Audio (*QOA*) Compression. *QOA* has a five to one compression ratio, which greatly reduces the amount of memory required for a full delay sample buffer.

Table 1: Audio Memory Specifications

Specification Type	Value (Unit)
Memory Requirement (no compression)	3.84 (MB)
Memory Requirement (compression)	0.77 (MB)

Audio Quality Specifications

The input audio stream to *RDP* is a 3.5mm stereophonic headphone jack, as described in Table 4. Although stereo is used as the hardware input source to the system, in order to reduce the total number of samples required for a full delay buffer while still providing quality audio output to the end-user, the audio input into the microcontroller is monophonic. Similarly, a monophonic audio source is used to output from *RDP* to reduce the processing needed to send a signal to the user.

The audio input specifications are an input depth of 12 Bits at 16 kHz. The audio output specifications are an output of 12 Bits at 32 kHz. The maximum delay window for the samples input into the system is 120 seconds. The total expected Bluetooth transmission distance is 50 meters.

Table 2: Audio Quality Specifications

Specification Type	Value (Unit)
Audio Input Bit Depth	12 (Bits)
Audio Output Bit Depth	12 (Bits)
Maximum Delay Window	120 (Seconds)
Sampling Frequency	16 (kHz)
Output Frequency	32 (kHz)
Bluetooth Audio Transmission Distance	50 (m)

1.1.2 Power

To properly ascertain certain power requirements, there were a couple of underlying specifications that needed to be established. As a group, we determined that we wanted our product to last around five hours. With this life-span, a user should be able to use *RDP* for a full match of any major sport, even if the match goes to extra time. Since most of the MCUs that we were investigating have a 3.3 V power supply, we decided that it would be easiest to use a 3.7 V DC rechargeable battery.

Once we chose our underlying specifications and designed our hardware, we were able to analyze the current required for outputting audio at relatively maximum power. To output audio on our device, around 50 mA are required drive audio to headphones. Knowing this number, the minimum battery size needed for our system to meet its specifications was computed to be around 250 mAh. In theory, this number is sufficient for our specifications, however, we decided to add a cushion to our desired battery size to guarantee that the battery would work for five hours. Our specification became 500 mAh for the battery size.

Table 3: Power Specifications

Specification Type	Value (Unit)
Battery Voltage	3.7 (V DC)
Expected Battery Life	5 (Hours)
Battery Size	500 (mAh)
Audio Output Driving Current	50 (mA)

1.1.3 User Interface (UI)

The user interface connects the user to the internal hardware and software that drives *RDP*. All *UI* specifications can be seen in Table 4 below.

Through this section, there are design choices that were made to design a small-scale device that is a mixture of affordable and easy to implement in the time allotted. For example, our current *UI* specifications use an LED to give feedback to the user (see LED UI Specifications). This design decision was made for a few reasons: to maintain a smaller product size by removing a need for hardware related to various displays, to decrease the amount of work needed in order to finish a complete product by the semester deadline, and to decrease the total cost to produce and sell our finished product. While the current UI and LED implementation is sufficient, it is important to recognize that there could be improvements to the UI including LCD displays or other feedback components.

Table 4: User Interface Specifications

Specification Type	**Value (Unit)**
Audio Input Connection	3.5mm Stereo Connector
Audio Output Connection	3.5mm Stereo Connector, Bluetooth
Audio Output Mode Select	Right Push-Button Hold (5 Seconds)
Enable Bluetooth Pairing	Left Push-Button Hold (3 Seconds)
Increase Delay Count	Rotary Encoder (Clockwise)
Decrease Delay Count	Rotary Encoder (Counter-Clockwise)
Delay Change Amount	0.25 Seconds per Detent
Battery Charging Connection	5 V USB-C Connector
Power Control	Toggle On / Off Switch
Battery Health Check	Rotary Encoder Button Press
Delay Count Check	Rotary Encoder Button Press

Audio UI Specifications

A simple reductive description of *RDP* is a system where a user inputs audio, delays audio, and listens to the delayed output stream. In order to maintain the general, universal norm of audio transmission for consumer audio products, a 3.5mm stereophonic jack is used for both audio input and output. In addition to the 3.5mm stereo jack for audio output, Bluetooth output is supported using the BLE protocol. All audio input and output specifications are discussed in Section 1.1.1.

A critical aspect of any delay product is the interface used to control the delay amount. For *RDP*, a rotary encoder knob is used to controller the delay amount. When turned clockwise the delay time increases. Similarly, when turned counterclockwise, the delay time decreases. Each turn in either direction modifies the delay time by 0.25 seconds. The rotary encoder knob has a push-button in addition to the knob. This push-button is used as a diagnostic button. When pressed, *RDP* outputs an audio clip using text-to-speech that says the current delay time in seconds. Additionally, the current battery health displays to the on-board LED.

Bluetooth UI Specifications

Since audio output can be via line-out or Bluetooth, there must be a way to control the audio output source. *RDP* defaults to using the line-out mode. To switch the output mode at any time during operation, there is a push button on the right edge of the device that the user must hold for five seconds. After five seconds, the mode will toggle from line-out to Bluetooth or Bluetooth to line-out. When in Bluetooth mode, the user manually establishes a paired connection. To do this, they must use the Bluetooth pairing button which is directly to the left of the mode select button. Holding for three seconds starts the pairing process.

Power UI Specifications

The rechargeable battery that comes with *RDP* can be charged using a standard 5 V USB-C cord. To power the system on or off, there is a power switch next to the USB-C Jack. As mentioned in the Audio UI Specification section, there is a health diagnostic button. This button is vital so the user knows if they have to charge their device while listening to a match. In addition to the battery health being displayed from the diagnostic button, it is also displayed when the device powers on.

Table 5: LED User Interface Specifications

Specification Type	**Specification**
Battery Check (0%, 25%]	Red LED (Blink, 1 Hz, 5 Seconds)
Battery Check (25%, 75%]	Yellow LED (Blink, 1/2 Hz, 5 Seconds)
Battery Check (75%, 100%]	Green LED (On, 5 Seconds)
Bluetooth Pairing Active	Blue LED (Fading, 1/2 Hz)
Bluetooth Pairing Success	Green LED (On, 3 Seconds)
Bluetooth Pairing Failed	Red LED (Blink, 1 Hz, 3 Seconds)
Audio Output Mode Switch	White LED (Blink, 2 Hz, 3 Seconds)
Maximum Delay Reached	Red LED (Blink, 4 Hz, 3 Seconds)

LED UI Specifications

LEDs are the main source of user feedback for *RDP*. The LEDs are primarily used in a few cases: Audio Delay Update, Battery Health Check, and Audio Output Mode Switching. All LED specifications are found in Table 5.

If the user attempts to increase the delay past the maximum value of 120 seconds or decrease the delay below the minimum value of 0 seconds, then the LED will blink red at four hertz for three seconds. This LED acts as a visual display to inform the user that their delay change is invalid.

Upon boot-up or diagnostic button press, the LED will display the current health of the battery. There are three potential outputs associated with this check. If the battery is between 0 and 25%, the LED will blink red for five seconds at one hertz. If the battery is between 25% and 75%, the LED will blink yellow for five seconds at ½ hertz. If the battery is above 75%, then the green LED will be turned on for five seconds.

When the audio output mode is switched, the LED will emit a white blink at two hertz for three seconds. As described in the previous section, when the user is attempting to pair to Bluetooth, the LED will emit a continually fading blue color at one half hertz until Bluetooth pairing succeeds or fails. If the Bluetooth pairing fails, then the LED will blink red at one hertz for three seconds. If the Bluetooth pairing succeeds, then the LED will display solid green for three seconds.

There is a possibility that a user may try to perform multiple operations at once, which can cause contention. A priority scheme was implemented where the LED only emits for the highest priority operation occurring. All other operations will not be output. The priority is in descending order: Bluetooth pairing, mode switch, battery check, audio delay update. This order was chosen to ensure the user knows what mode they are operating in, and if they are connected to Bluetooth. Interrupting the output for these functionalities is more disruptive to the user's experience compared to interrupting the current battery level or delay. Without the user knowing if they are properly outputting audio, no other functionality matters.

1.2 System Design

This section covers part selection, implementation choices, and algorithm design for *RDP*. The overall details and hardware design choices that are critical to *RDP* are discussed for reproducibility or for other engineers to quickly learn and work on *RDP*.

1.2.1 Part Selection

This section contains a comprehensive list of all of the critical, non-generic parts that were chosen to develop *RDP*. In addition, there is a brief discussion of select parts and reasoning for their use in implementation.

Table 6: Parts for *RDP*

Specification Type	**Specification**
Microcontroller	ESP32-S3-WROOM1-N8R8
Rotary Encoder	PEC12R-2217F-S0024
Operational Amplifier	LMV358MMX/NOPB
Digital Potentiometer	MCP4017T-104E/LT
Battery	ASR00035
USB Jack	USB4125-GF-A-0190
Push-Button	TS11-674-80-BK-160-RA-D
Battery Connector	S2B-PH-K-S
Power Switch	EG1206A
Battery Charging Chip	MCP73812T-420I/OT
LED	EASV3015RGBA0
P-Channel MOSFET	BSS84
Buck Switching Regulator	AP3445LW6-7
Stereo 3.5mm Connector	SJ1-3523N
Radial Capacitors	860020372003
Keyed Debug Connector	61200621621

Microcontroller

The microcontroller is the cornerstone for any embedded systems project. While choosing a microcontroller for this project, competing interests had to be considered including: cost, built-in peripherals and memory, clock frequency, available pins, product packaging, and, documentation and support.

Since ample memory is important for storing samples, it was decided that the ESP32-S3-WROOM1-N8R8 would be the best product for our initial design and implementation. This product came as a module with 8 MB of Flash Memory and RAM built-in. This is sufficient memory to store an entire program and any samples needed for our delay buffer. Additionally, this product was significantly cheaper compared to most of its competitors. It cost under four dollars and included all on-chip functionality that we needed including ADCs, PWM, I2C, and BLE.

After working with the ESP32 processor, there were some negative characteristics that were not apparent when purchasing the processor. These issues as well as potential path's forward are discussed in Section 2.4.

Rotary Encoder

Two methods of delay control were discussed by the group. Ultimately, a rotary encoder was chosen to be the delay control. While looking for rotary encoders it was important to keep in mind size, cost, and number of pulses per revolution.

Bourns, Inc. has a wide selection of small scale, relatively cheap rotary encoders that would be adequate for our product. The PEC12R-2217F-S0024 was chosen for *RDP* since it had the shortest knob and frame size, which reduces the amount of space that the rotary encoder held in the device. Additionally, this product has a push-button on the encoder, which reduces the total number of push buttons that are needed elsewhere on the product.

Ultimately, the rotary encoder proved to be the main reason of the thickness of *RDP*. The effects of this are discussed in Section 2.4.

Operational Amplifier

The operational amplifier is a gain adjusting device for the ADC for audio input, a filter for PWM output, and current driver for the final audio output to the 3.5mm headphone jack. It is important to find an operational amplifier that drives enough current, is rail-to-rail, meets the voltage supply constraints of an embedded device, is affordable, and is in a reasonably sized package.

Texas Instruments creates a product line of rail-to-rail operational amplifiers that fit our needs of electrical characteristics, cost, and physical dimensions. The LMV358MMX/NOPB dual-channel operational amplifier can drive up to 160 mA of current, - which is well above our requirement - can be supplied by 3.3 Volts, is relatively cheap at under 50 cents per unit at 1000 part pricing, and is in a small scale 8VSSOP packaging. This operational amplifier proved to be a great selection for all stages of the design in which it was used.

Digital Potentiometer

It is important that our system performs some sort of gain control to ensure that audio input into the ADC is recorded in an appropriate range without clipping or distortion. Our solution to this problem is to use a digital potentiometer to auto-adjust the input gain ratio. The ideal digital potentiometer has range of resistance for attenuation and gain such that most audio sources would supply a signal that could properly be recorded and played back.

MCP4017T-104E/LT is a digital potentiometer that meets our needs for attenuation and gain. The product ranges from 0 to 100 kOhms split evenly between 128 steps. Our input signals to the operational amplifier circuit passed through a 10 kOhm resistor, meaning we had a dynamic gain range of 0x to 10x. In addition to the part having the electrical characteristics required, David had already worked with a similar part, which made its implementation straightforward.

After implementing the initial PCB board, it was discovered that the attenuation of signals was more likely than requiring an increase of gain of a signal. As a result, nearly 90% of the digital potentiometer is essentially useless. Alternate potentiometers in the MCP4017T family have smaller resistance ranges still split into 128 steps, which can provide more precision for attenuation. A new part was not ordered, but is discussed in Section 2.4.

1.2.2 Implementation Choices

This section overviews any implementation choices that were made during development of *RDP*.

Automated Input Gain Regulation

Different radios and audio sources can provide different voltage levels for line-out audio. This could be a massive problem since the ESP32 microcontroller utilizes ADC units that have a maximum tolerance of just above two volts. If signals greater than two volts are input into *RDP*, the audio could become distorted or unintelligible. As a result, a design decision for our input summing circuit was to use a digital potentiometer as the feedback resistor. By doing this, software could be created to dynamically adjust the input volume if the input was above a certain threshold. This ensures that audio signal inputs that are too hot can still be utilized by our system.

Monophonic Audio Signal

Initial designs and testing for *RDP* utilized stereophonic audio input and audio output. In theory, this creates the best listening experience for a user as possible. However, stereophonic audio input and output adds more work to both hardware and software. An additional op-amp channel is needed to handle separate audio input channels, two additional pins are required for PWM and ADC units, two times as much memory is required for storing the samples, and the software becomes more complex to handle the storage, access, and timing of audio samples being input and output. All of these facts contributed negatively to increased product size, product cost, and development time.

After realizing the complexity associated with stereo output, and after redefining our intended use case of *RDP* being for talk radio only, a monophonic signal input and output became sufficient for *RDP*. *RDP* receives audio from a 3.5mm stereophonic audio jack, which is combined into a monophonic input through an operational amplifier summing circuit. The output from this circuit is treated as our input into the microcontroller. For audio output, one single PWM source is sent through a low pass filter, and then the output signal is sent through two separate operational amplifier current driving circuits to send to the left and right channels of a 3.5mm stereophonic audio jack.

Hardware Filtering

As a rule of thumb, it is important to filter out any potential noise that is associated with PWM audio output generation. Due to hardware limitations, we could not output our signal at such a frequency that would naturally remove the PWM noise from the human audible range. A hardware low-pass filter is used to filter out any potential noise. To make the design compact and functional, we decided to use a second order low-pass filter.

1.2.3 Algorithm Design and Selection

This section overviews the algorithms that were designed or utilized by *RDP* development.

Audio Output Dithering

The audio specifications required the use of 12-bit audio input and output. Due to constraints on the ESP32 microcontroller's PWM modules, the highest bit rate that we could achieve with a 16 kHz output sample rate was 11-bits. This meant that we were unable to achieve our initial audio specifications chosen in the fall semester. In order for us to gain an effective extra bit of precision, we utilized dithering on our audio output. By applying dithering, we were able to slightly modify our desired PWM output duty cycle twice an output period, which gives an extra effective bit for our audio output.

Dithering was implemented by recording and outputting samples at 16 kHz. An additional hardware timer runs at 32 kHz in sync with the other timers. This secondary timer controls the PWM duty cycle for audio output. This timer accesses the same sample in consecutive cycles. On the first access to the sample, the raw sample is output via PWM. On the second access to the sample, the value is added with one, creating a slightly different duty cycle for audio output. It is here where the effective additional bit is created for our audio output.

Quite OK Audio (QOA) Compression

Stereophonic audio was initially the desired audio quality. For stereo sound, *RDP* would require nearly 8 MB of storage for audio samples. Even though the ESP32 microcontrollers can come with 8 MB of on-module PSRAM, modules would be cheaper with smaller PSRAM sizes. To reduce the amount of memory required by our system, QOA Compression can be effectively used to compress audio without losing quality of sound. With a compression ratio of 5:1, the amount of memory needed lowers in size substantially, reducing the overall cost. For more information, QOA has a website that contains source code and details about their compression algorithm: `https://qoaformat.org/`.

Rotary Encoder Software Debouncing

In terms of software, handling the rotary encoder electrical bouncing seemed to be the most difficult software to create. Unlike certain microcontroller manufacturers, Espressif does not have built in rotary encoder reading mode functionality. Rotary encoder signals must be read manually as two separate GPIO pins on the microcontroller. An algorithm is required to properly analyze if a rotation occurs, and if so, in which direction it occurs.

In products with minimal bouncing, straightforward algorithms and state machines can be found online to track whether or not a rotary encoder has been turned. Unfortunately, in our electrical design, there was a significant amount of bouncing when pulses for rotation were sent to the microcontroller. This resulted in additional rotations being logged per one actual rotation when using previously published methods of software debouncing. A state machine had to be created to properly ensure that readings of pulses that followed incorrect state patterns would not be triggered.

The state machine tracks the previous two states of both rotary encoder channels as a four-bit number. When a new pulse triggers the GPIO callback, both rotary encoder channels are read and are packaged into a two-bit number. This number is compared against either of the two previous states. If it is equal to one of the previous two states, then the input is interpreted as bouncing noise from the initial pulse edge, and the callback immediately exits. If the two-bit current state is not equal to either of the two previous edges, then we can assume that the interrupt being triggered is from the initial pulse edge, so the delay counter is modified using a predefined look-up table. To the best of our knowledge, this is a novel technique.

Scaling Audio Samples

To effectively use the entire range of the PWM period, input samples recorded by the ADC have to be converted to cover the entire duty cycle range. Using an 80 MHz base clock and 32 kHz timer to gather samples, the calculated duty cycle range is from 0 to 2499. To fully center and maximize audio output, we have to divide the current sample by the size of the 12-bit range (4096), and multiply by the duty cycle range for our PWM timer (2500). This scaling effectively centers and maximizes the audio output signal.

Simultaneous Recording and Playback

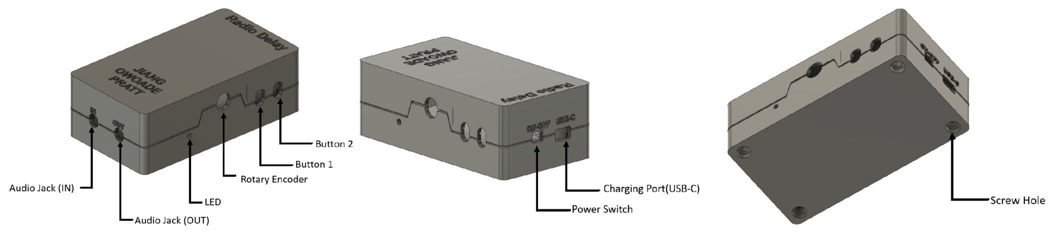
An often-used approach to record samples for playback is a double buffer approach. In this scheme, one half of the buffer is being written as the other half of the buffer is being played back. For our real-time playback, we needed a different approach. Our approach uses a read and a write pointer on a sample buffer. The write buffer pointer will always be equivalent to the current read pointer minus any delay. To properly index the buffer, all calculations of read and write pointers are done with modulus of the size of the buffer to ensure that wrap-around of indices happens cleanly.

Our playback scheme works fairly well. However, there are instances where clicks or glitches occur on *RDP*. These problems occur when the playback pointer points to the exact location where bytes are currently being written on start-up or when the delay time goes from non-zero to zero. To account for this, a minuscule delay of around 80 samples (5ms) is applied. The addition of this delay removes any clicks or glitch sounds that result from start-up and resetting the delay to zero.

1.3 Product Packaging

Developing the product packaging proved to be a relatively difficult endeavor, since none of the members of the Radio Delay Project group have a deep background in mechanical modeling software and design.

A goal of the packaging design is to create a device that is portable and usable in multiple environments. Users may want to listen to their broadcasts while mobile or stationary, so it is important that they have a product that is easily ported. We created a product package that fits our needs of portability. It should be noted that our product packaging is more thick than we would prefer. Future work to the packaging and hardware design of *RDP* include attempts to continue to minimize the overall size of the product. Figure 1 contains a few images of the final product casing.



(a) Audio and System UI (b) Power UI (c) Bottom View

Figure 1: Product Packaging. (a) contains the view of *RDP* that contains the audio and system user interface. (b) contains the view of the power user interface (c) contains a bottom view showing how the packaging is sealed.

2 Product Development

2.1 Final Product Status

The final product status at the end of the Spring 2024 semester is a functioning device with both hardware and software implemented.

During the semester, we were able to create and order two rounds of hardware prototypes. The first round of boards had two hardware defects that required wire workarounds. Aside from those workarounds, the entire system had the required system functionality that was desired by the team. The two workarounds were: connecting the analog battery health signal to a pin that was incompatible with an ADC module and not grounding the rotary encoder. With about two weeks left in the semester, we ordered ten new boards with enough parts to populate five of those boards. We implemented the fixes from the first round of boards that were ordered, and tested to confirm that there are no other problems with these new boards.

While we do have a functioning hardware and software device, there are specifications that we did not have implemented or hardware selections that can be improved upon. We did not get around to programming the BLE connectivity as desired when first starting to work on the project. It would be nice to make this feature work. Due to time, we also did not implement QOA Compression and automatic dynamic control. Additionally, some of our hardware could be replaced with better alternatives, such as an external ADC to reduce noise, use of a different delay control mechanism to reduce size, and a different microcontroller to correct certain deficiencies in the ESP32. All of these potential changes are covered in Section 2.4.

2.2 Product Development Timeline

Our group had high hopes to finish our product, or at least the main bulk of the work, by the end of the Fall 2023 Semester. As seen in Table 7, our initial phases of design met the timelines that we set for ourselves. However, we were unable to maintain our timeline due to various factors. In this section, each design phase is analyzed with respect to the work that was completed in the phase, as well as any delays that may have occurred.

Table 7: Product Development Timeline

Design Phase	Planned Completion Date	Actual Completion Date
Planning / Specifications	October 12th, 2023	October 12th, 2023
Initial Testing	October 26th, 2023	October 26th, 2023
Circuit Design / Prototyping	November 13th, 2023	February 29th, 2024
Development / Assembly	November 30th, 2023	May 10th, 2024

2.2.1 Planning and Specifications

The initial phase of development is Planning and Specifications. During this phase of development, the entire team was responsible for working together to plan what kind of product we wanted to develop. Responsibilities for this phase of work include deciding the product that was going to be developed, choosing the initial design specifications for our product, completing an initial high-level block diagram of our design, and performing preliminary hardware testing for functionality. This phase was straightforward, so we were able to complete the necessary work on schedule.

2.2.2 Initial Testing

The second phase of development is Initial Testing. In this phase, each member was responsible for testing various aspects of our design ranging from hardware to software. Example testing that was completed during this phase was Audio Input Amplitude Testing, Audio Output Testing, User Input and Output Testing, and Power Testing. During this phase, our conceptual designs started to become actualized into hardware and software. Additionally, during this phase of development the team started to become more familiar with the microcontroller and hardware that we would be using. The results of these tests helped inform the team on what the final schematic would require. With respect to timeline, we had to wait for parts to arrive to perform some of this testing. Once we received the parts, we were able to get our work done on schedule.

2.2.3 Circuit Design

The third phase of development is Circuit Design. This was the longest phase of our development process. During this phase, the group finalized all circuitry and electrical layouts that were required for our embedded system. After completing various testing in the Initial Testing phase, this section should have been a relatively simple culmination of all of the work that we had done leading up to this point. However, this phase proved to be the most time consuming.

In general, our schematics were finalized around a week behind what we had intended due to valuable presentatino feedback from Professor Patterson, Professor Silverman and Ken Silverman. While our overall schematics were acceptable, the transition of schematics to PCB layout proved to add a lot of unforeseen complexity. Due to Winter Break, we were unable to receive feedback on our designs until the end of January. After resolving all of the PCB problems, we were able to order our first PCB boards at the end of February. This added around three months of development time to our initial estimates.

2.2.4 Development and Assembly

The final phase of our development process is Development and Assembly. This phase consisted of receiving our ordered PCB boards and electrical components, assembling our hardware, developing any required software, and making board revisions as necessary. During this phase, the group was able to test all hardware and software on board. As mentioned in Section 2.1, we did have to re-order a board to fix two problems that were found on the initial boards ordered from the Circuit Design phase.

This phase required more time than previously anticipated due to the amount of software and hardware triage required. What was planned to be a three-week phase turned into around a ten-week phase of development. The main cause for these delays was an underestimation of the amount of time required to develop the software for the product. Another cause for delay was the ordering of incorrect packages for certain electrical components leading to additional orders being required.

2.3 Individual Contributions

Each individual contributed to the development of *RDP*. This section contains a brief discussion of each team members contribution.

Tengfei Jiang

Tengfei spent most of the Fall working on creating the schematics and PCB Layout for *RDP*. The finalized hardware design can be attributed to the work that Tengfei did in the Fall and early Spring semesters. This work was time consuming, as almost weekly we would receive feedback that required changes to both the schematic and PCB layout. Once PCB boards were ordered and arrived, Tengfei played a critical role in helping debug and populate the initial boards that were created. Seeing that there were some issues with the initial board, Tengfei made some changes to the initial PCB design, which have since been re-ordered and proven to work.

David Owoade

At the early stages of the project, David was charged with designing the audio input and output modules, as well as selecting the parts that comprised those sections. David was additionally in charge of the product packaging and design.

Parts selection was done on Digikey, where David searched for components whose features matched the initial design specifications. As a result of the team skipping some required testing, this endeavor turned out to be more time-consuming than anticipated. David led testing for audio output by using tools like Multisim. These tests provided useful information about the electrical requirements for the audio input and output modules.

After the initial PCB had been delivered and populated, David began work on creating the packaging using Fusion 360 and the Bambu 3D printer in Prof. Silverman's Laboratory. David continued this work through the end of the semester.

Alexander Pratt

Alec's work was primarily based in clerical work, system design, and software engineering of *RDP*.

At the beginning of the Fall 2023 semester, Alec unofficially assumed the role of team leader. Throughout the entire academic year, Alec led the group by delegating work and ensuring that deadlines were being met. Alec created all of the presentations for the in-class status updates. Alec tracked all of the progress of the group's work through Gantt charts. Additionally, Alec primarily wrote the entire final report for the project.

In terms of system design, Alec was mainly responsible for the overall block diagram and high-level design that was created during the Planning and Specifications Phase. Alec was not a large contributor in the hardware design and testing for the system. The group decided that the best way to work as a team was to focus everyone on their core strengths. By leveraging our experiences and skill sets, we thought we could speed up our design process. As a result of this, Alec primarily focused on the software development for the project.

While David and Tengfei were working on hardware designs and schematics, Alec spent most of his time in the lab learning the ESP32 API and software libraries. Alec spent the Fall 2023 semester creating breadboard representations of the exact circuitry that the group would try to use on our PCB. By creating these simulated circuits, Alec was able to test fully functioning software for aspects of *RDP* such as handling the rotary encoder, recording audio samples, storing audio samples, and outputting audio samples. Creating this software in the fall semester in parallel with hardware design enabled a seamless transition from using the ESP32 development kit to using the PCB hardware. In the Spring 2024 semester, Alec had to modify the code that was being used on the development kit to use the appropriate pin configuration as described in the *RDP* schematics. Additionally, Alec spent a lot of time getting the rest of the software functionality to work including LEDs, a full delay buffer of recording and playback, battery health monitoring, and user interface schemes.

2.4 Future Work

As mentioned in Section 2.1, by the end of the Spring 2024 semester, the team was able to produce a functioning version of *RDP*. This section highlights potential opportunities for improvement or expansion upon our current design.

2.4.1 Audio Input Recording

Analog-to-Digital Conversion Hardware

The analog-to-digital conversion (ADC) units built into ESP32 microcontrollers are noisy, which was discovered after testing *RDP*. Traditionally, oversampling and a digital filter can be used to reduce noise. However, the ADC units on ESP32 cannot record above around 60 kHz. For our sampling rate, we could barely get a four times oversample. Ideally, we want around an eight times oversample to minimize noise. Ultimately, this means our hardware creates a noisy signal that cannot efficiently be filtered.

A potential solution to this is to use an external ADC unit that connects to the microcontroller via I2C or SPI protocol. External ADC units can record well above our required eight times oversample rate, and I2C on ESP32 microcontrollers can transfer data at the required rate as well. This adds additional cost and requires additional space on the PCB, but using an external ADC would improve our audio recording quality and allows for oversampling with a digital IIR filter on audio input.

Digital Potentiometer Range

Currently, the digital potentiometer that is used for automatic gain adjustment is sized to 100 kOhms, split evenly across 128 steps. This gives a step granularity of around 780 Ohms. Microchip Technologies offers an identical digital potentiometer to the current product being used that is sized to 50 kOhms. This gives a step granularity of around 390 Ohms. The next smallest resistance value of these products is 10 kOhms, which makes the 50 kOhm digital potentiometer the smallest sized digital potentiometer that provides *RDP* with its required electrical characteristics.

The primary purpose of the potentiometer is to reduce input gain, not increase gain. In a worst-case scenario, we would not expect voltages below 1 volt, which would require at most a two times gain. The 50 kOhm digital potentiometer exceeds our requirements for increasing and decreasing audio input gain with a 10 kOhm input resistor. Utilizing this digital potentiometer provides twice the precision of adjustment and in theory, a better listening experience.

2.4.2 Microcontroller Selection

The reasoning for selecting the ESP32 line of microcontrollers is discussed in Section 1.2.1. The microcontroller was successful in doing everything that we required of it. It was able to record analog signals, play audio in an audible range, store memory, and more. However, a lot of the on-chip peripherals and software libraries were not useful.

Espressif does not have a normal PWM library. Instead, they have an LED PWM Mode and a Motor Control PWM Mode, neither of them offers a generic PWM option similar to those offered by competitors like STMicroelectronics. To get the LED PWM Mode to act like a normal PWM output, a software timer had to be used in parallel to an LED PWM module to properly update the duty cycle for audio output at a regular interval. This essentially doubled the amount of code that needed to be written in order to create a functioning PWM output.

The analog-to-digital conversion unit on-chip is extremely noisy, which leads to a significant amount of noise being added to recorded samples at low amplitude. As mentioned in the previous section, a fix to this may require an external ADC electrical component to get a high quality audio recording.

Although the ESP32 microcontrollers have a lot of built-in functionality for a low cost, it may be worthwhile to look into other microcontroller options. Competitors, such as STMicroelectronics, come with better software libraries, better community support, and better on-chip hardware (ADC, PWM, etc.). There are drawbacks to switching to a new microcontroller, like requiring off-chip SRAM/PSRAM and external Bluetooth hardware. However, based on our current specifications and implementation, we are not using Bluetooth and we are requiring less than a megabyte of data to be recorded, which should not be expensive to purchase as an external module.

2.4.3 PCB Layout Re-Design

Currently, *RDP* uses a PCB layout where all components - excluding three capacitors and a battery connector - are on the top surface of the product. Because of this, space is not used optimally due to relatively tall parts being on both the top and bottom of the system. Moving forward, it may be better to optimize the PCB layout for component height to continue to reduce the packaging height. Examples of this optimization include placing the battery on the same side of the tall components instead of on the bottom of the board.

2.4.4 User Interface Improvements

Effective Use of Push-Buttons

At the moment, there are three push buttons on *RDP*. It seems like we have too many buttons which could make the user interface confusing. In the future, it may be worthwhile to think through required and effective uses of push buttons to reduce the amount of interactive hardware accessible to the user.

LCD SCREEN

An LCD screen would be a valuable addition to *RDP*. Although it increases the cost, an LCD screen would provide clear visual feedback for important information relating to audio delay, Bluetooth network connection options, battery health, etc. This was initially thought of as an option for user feedback when designing *RDP*, but the group decided not to pursue this design option to reduce product cost and reduce complexity of design and implementation.

3 ECONOMIC ANALYSIS

This section highlights the product cost including parts, personnel costs, and overhead. The subsections explore how each cost type impacts the overall final cost of production.

3.1 DEVICE COST

This section contains a table of the cost per part for a single *RDP* device. The prices listed are at a per thousand rate unless specified otherwise.

Table 8: Cost of Parts for *RDP*

Part	**Quantity**	**Cost per Unit (USD)**
ESP32-S3-WROOM1-N8R8	1	3.62
PEC12R-2217F-S0024	1	0.92834
LMV358MMX/NOPB	2	0.43935
MCP4017T-104E/LT	1	0.53
ASR00035	1	6.95
USB4125-GF-A-0190	1	0.34663
TS11-674-80-BK-160-RA-D	2	0.08908
EG1206A	1	0.46162
MCP73812T-420I/OT	1	0.55
EASV3015RGBA0	1	0.16107
BSS84	1	0.14079
AP3445LW6-7	1	0.13817
2N3904BU	1	0.06774
SJ1-3523N	2	0.50774
LBR2012T2R2M	1	0.04545
860020372003	2	0.05800
61200621621	1	0.38880
Resistors	33	0.000980
Capacitors	20	0.01761
PCB	1	0.5859

Using the values from the table above, the cost of the parts for a single *RDP* device is around 17.77845 USD.

3.2 DEVELOPMENT COST

There are costs associated with paying for engineers, work spaces, and technologies that are used to develop the product. This section includes the valuation of overhead to develop a product. Most of these costs have been ignored while creating the project since we did not have to pay for engineers or lab space.

3.2.1 INITIAL NON-RECURRING ENGINEERING (NRE)

In theory, three engineers worked on this product. This initial NRE calculation is used to determine how much it costs to pay three entry level engineers to work on a similar project in industry. The numbers calculated in this section use the rates provided by Prof. Patterson from 2010. In order to pay for three engineers in entirety for the eight-month project, including the benefits entitled to engineers, it is estimated the total cost is approximately 137,324 USD. This is around 45,775 USD per engineer. Although it is recommended in the report document to double our NRE, we believe that we have a finished product at the end of the semester, so we will not double this value to compensate for additional development time.

3.2.2 Manufacturing Transfer

According to the project guidelines, we should assume that the costs to turn a small startup into a manufactured product should double our initial NRE due to stiff costs for industrial designs. With this assumption, we assume that our NRE is doubled from 137,324 USD to 274,649 USD.

3.2.3 Assembly

For assembly cost, we have calculated that it will cost around 1.1343 USD per unit. This is assuming it costs one cent for every surface mount component, two cents for every through hole component, and one cent per square inch of product packaging. This increases the overall cost of creating a device to become 18.91275 USD.

3.3 Analysis of Product Cost

3.3.1 Final Sale Price

Based on the calculations above, the final sale price of the device should be around three times as large as the per unit cost. This is to build in profitability from assembly costs, NRE, marketing, and distribution. This brings our product sale price to be around 53.33 USD. In order to account for markup through wholesale and retail sales, we should assume a multiplier of around 1.5 times, resulting in a final sale price of 80 USD.

3.3.2 Is *RDP* Sellable?

The final sale price of 80 USD is on the lower end of cost for radio delay products currently on the market. Table 9 compares *RDP* to two other products. It is possible for *RDP* to be sold for a smaller sum of money, if certain features are changed or different assumptions of profitability, marketing, and distribution costs were changed. We think that the models used for computing the profitability of the product for this assignment are on the higher end of what actually may be required to create the product. We believe that based on functionality and comparison to current market products, *RDP* can be a budget competitor within the market.

Table 9: Competitor Comparison

Category	**RTS-200C Radio TVSync**	**SR-404 SportSync Radio**	**RDP**
Cost (USD)	199.95	69	80
Max Delay (S)	86	60	120
Other Features	Separate remote control for setting delay, changing modes, and muting audio	Settings and control are done on device Built in radio	Settings and control are done on device No built in radio

In comparison to it's competitors, it appears that *RDP* can survive on the market. However, it is important to determine how many units it would take to make back the money invested in NRE. Assuming costs of NRE to be 274,649 USD, and a sale price at 53.33 USD, it would take around 5150 units to be sold to break even on our initial investment, assuming all proceeds from the 53.33 USD went back to recovering our initial investment and not toward profit. There is a real chance that we cannot sell 5150 units of this product since our competitors are no longer able to be found on platforms such as Amazon. Based on this fact, the current price valuation may make our product infeasible for long term production and sale.

Appendix I: Calculations for Specifications

This section contains all of the relevant calculations that were used for specification decision for *RDP*.

Memory Usage Computation (Uncompressed)

$$Memory_{Uncompressed} = \frac{\#\ Bits}{Sample} * \frac{\#\ Samples}{Second} * \#\ seconds * \frac{1Byte}{8Bits}$$
$$Memory_{Uncompressed} = 16\ Bits * 16\ kHz * 120\ s * \frac{1Byte}{8Bits}$$
$$Memory_{Uncompressed} = 3.84\ MB$$

Memory Usage Computation (Compressed)

$$Memory_{Compressed} = Memory_{Uncompressed} * Compression\ Ratio$$
$$Memory_{Compressed} = 3.84\ MB * \frac{1}{5}$$
$$Memory_{Compressed} = 0.768\ MB$$

Battery Capacity Computation

$$Battery\ Size = I_{Audio\ Input} * BatteryLife * Battery\ Health\ Buffer$$
$$Battery\ Size = 50\ mA * 5\ Hours * 2$$
$$Battery\ Size = 500\ mA$$

Appendix II: Calculations for Economic Analysis

PCB Cost

$$Cost_{PCB} = Cost_{Square\ Inch} * PCB_{Length} * PCB_{Width}$$
$$Cost_{PCB} = 0.07USD * 2.3622\ in * 3.54331\ in$$
$$Cost_{PCB} = 0.5859\ USD$$

Employment Cost

$$Cost_{Employment} = \#\ Engineers * Engineer\ Salary_{2010} * Time_{Years} * Benefits\ Multiplier$$
$$Cost_{Employment} = 3\ Engineers * 62,420\ USD * \frac{2}{3}\ years * 1.1$$
$$Cost_{Employment} = 137,324\ USD$$

Packaging Size

$$Size_{Packaging} = 2 * L * W + 2 * W * H + 2 * L * H$$
$$Size_{Packaging} = 2 * 3.5433\ in * 2.3622\ in + 2 * 2.3622\ in * 0.9055\ in + 2 * 3.5433\ in * 0.9055\ in$$
$$Size_{Packaging} = 27.4321\ in^2$$

Assembly Cost

$$Cost = \#\ Parts_{Sur.Mnt.} * Cost_{Sur.Mnt} + \#\ Parts_{Thr.Hole} * Cost_{Thr.Hole} + Size_{Pckg.} * Cost_{Sq.In.}$$
$$Cost = 62 * 0.01USD + 12 * 0.02USD + 27.4321in^2 * 0.01$$
$$Cost = 1.1343\ USD\ per\ Unit$$

APPENDIX III: ELECTRICAL SCHEMATICS AND PCB LAYOUT

SCHEMATICS

This section contains the final schematics that are used for the Radio Delay Project. There are three separate schematics: Audio, ESP Interface, and Power.

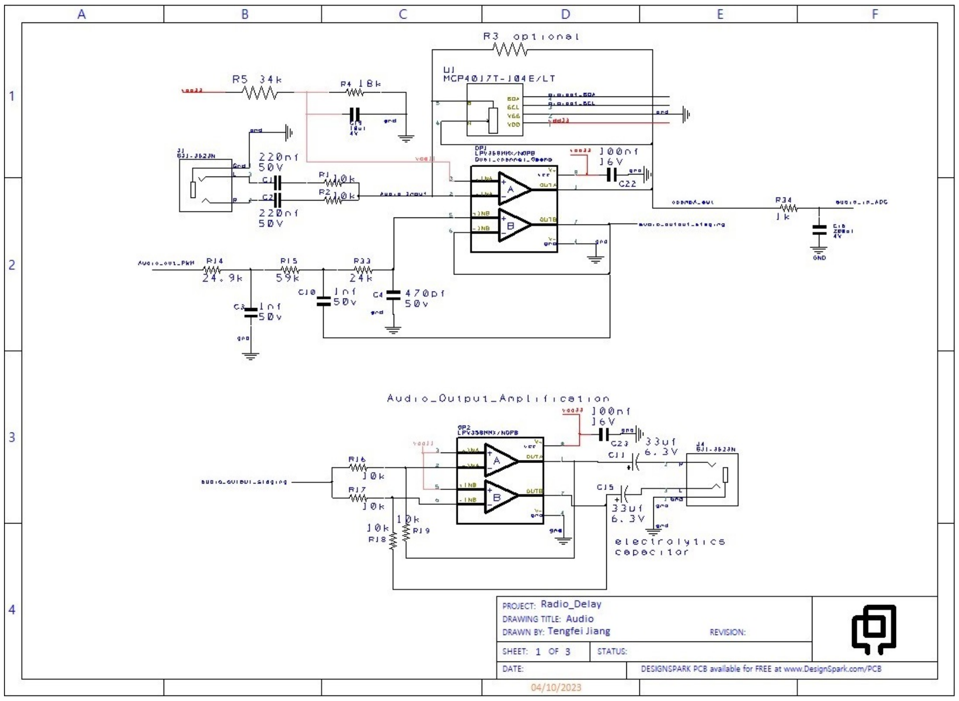


Figure 2: Audio Hardware Schematic

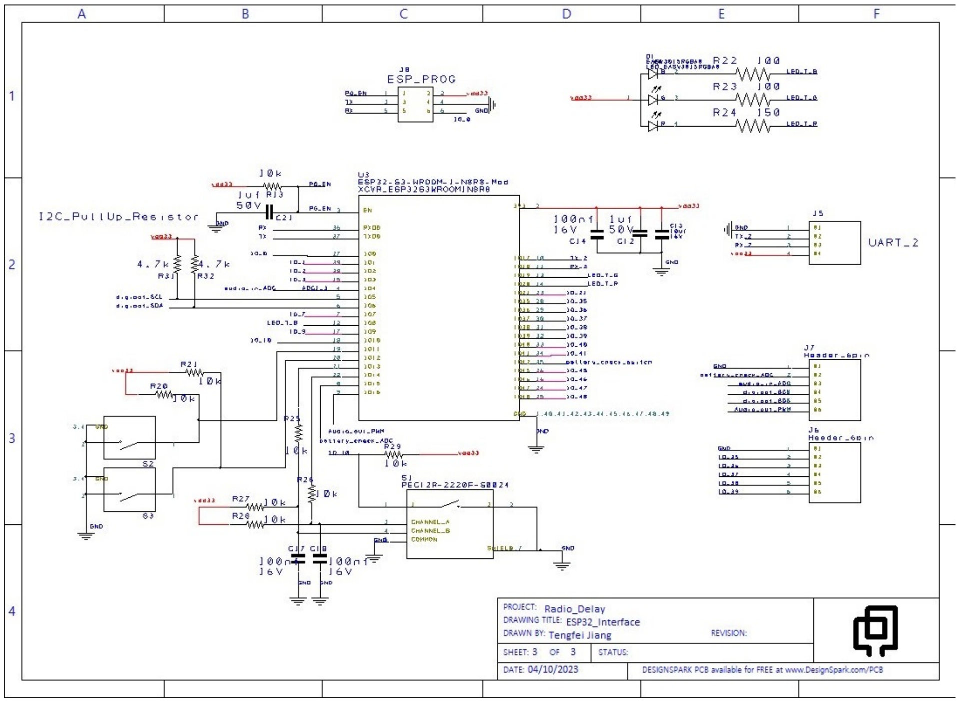


Figure 3: ESP Interface Hardware Schematic

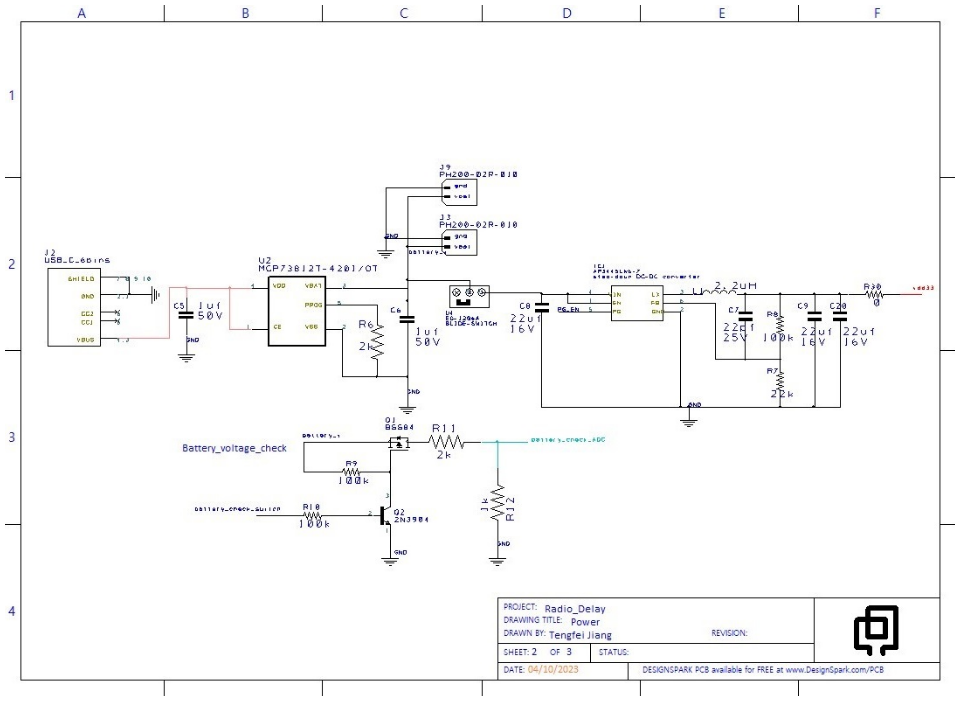


Figure 4: Power Hardware Schematic

PCB Layout

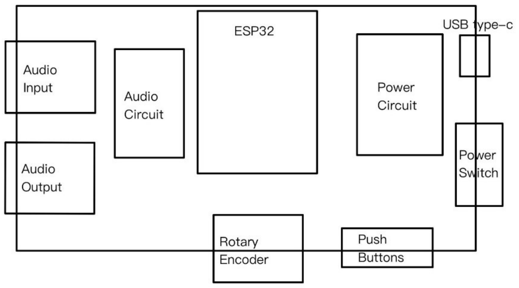
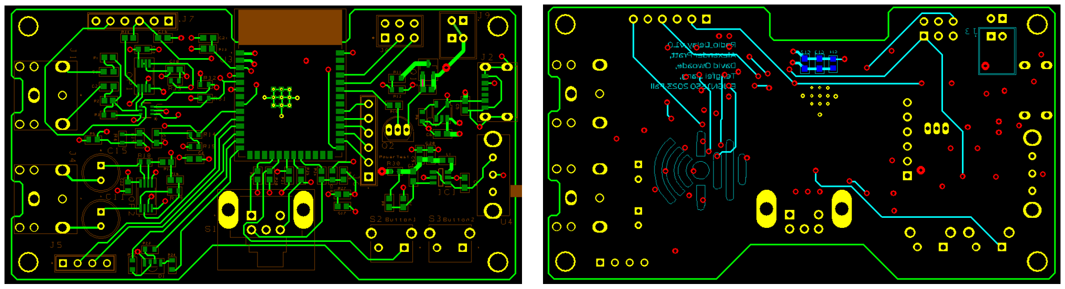
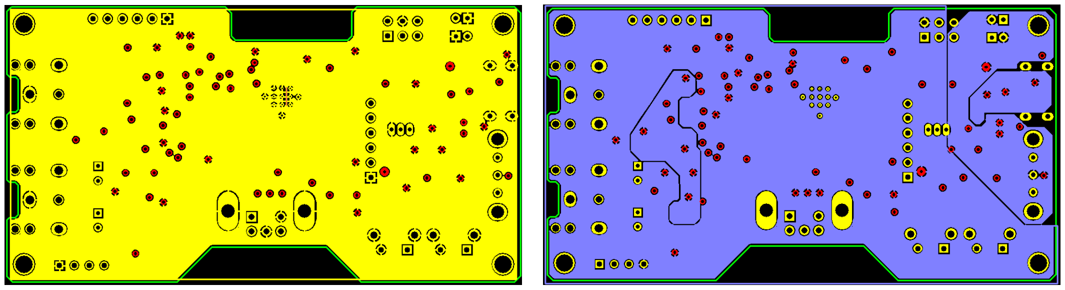


Figure 5: General Layout of PCB for *RDP*



(a) Top Copper Layer of PCB for *RDP* (b) Bottom Copper Layer of PCB for *RDP*

Figure 6: Copper layers of final PCB



(a) Ground Inner Layer of PCB for *RDP* (b) Power Inner Layer of PCB for *RDP*

Figure 7: Inner layers of final PCB. Note that for (b), in general all of the planes are 3.3 V. However, the left island is 1.1 V for the audio input and output op-amps, and the right islands are 5 V for USB input and battery charging.

Appendix IV: Link to Repository

Repository Link: `https://github.com/alecpratt/ENGN1650`

NOTE: If you are unable to access the repository, request access via email to alexander_pratt@brown.edu or alexandermarcuspratt@gmail.com.

Project Background

The Radio Delay Project is the year-long embedded microprocessor design project intended to fulfill the requirements of ENGN 1650: Embedded Microprocessor Design at Brown University for the 2023-2024 Academic Year. This project comprises a full scale design that includes hardware, software, and mechanical components. The team members are Tengfei Jiang (ScM Electrical and Computer Engineering), David Owoade (ScM Electrical and Computer Engineering), and Alexander Pratt (ScM Computer Science).

Acknowledgment

Special thanks to Prof. Harvey Silverman, Prof. William Patterson, and Ken Silverman, for providing us with technical and financial support throughout the product development process.

We would also like to thank the Brown University Engineering Department and Brown University for their support and allowing us to use the facilities and resources throughout the product development process.