Amica Aura:
Wireless Headphones with Audio Sharing

Ethan Bless-Wint, Winston Ching, Michaela Laurencin

Electrical and Computer Engineering Department, Carnegie Mellon University

***Abstract* — Amica Aura is a wireless headphone system that has active noise cancellation as well as audio sharing as additional capabilities. The system also features gesture controls and wireless charging. With this system, users can listen to audio, host a broadcasting session of their own audio, or join a nearby mesh station another user is broadcasting.**

***Index Terms* — Headphones, Bluetooth, Mesh, Active Noise Cancellation, Wireless**

I. Introduction

AMICA AURA is set of wireless headphones containing active noise cancellation and wireless audio sharing capabilities. Audio prosumer technology has experienced rapid development in the last few years. Specifically, the rise of headphones has dramatically increased the convenience of on-the-go listening. In this new age of auditory freedom, one area that has still not been fully developed is the ability to socialize. With our project, we would not only be able to have a device that can satiate this growing desire for wireless audio technology, but also fill the void of communal audio enjoyment that is in the current market. Competing products do exist, but they miss some aspect of this demand that our product covers, either the mobile nature, isolated sharing, or ability for standard headphone usage.

We have chosen to optimize four particular areas of our project to achieve a successful audio listening and sharing experience: sound quality, audio sharing, usability, basic level functionality. On the whole, we aim to achieve a pair of headphones capable of (a) in normal playback mode, playing 44.1kHz stereo audio at a bitrate of 320kbps via Bluetooth (b) detecting users' in-air hand gestures within 3cm of our headset with electrodes carrying 115kHz oscillating electric field and recognizing said gestures within 1s after their completions; (c) charging wirelessly at 5W or connectedly at 15W; (d) in multi-playback mode, broadcasting 320kbps stereo audio to at least two other headphones simultaneously with less than 30ms of latency (and less than 180ms for six other headphones in a mesh network) while maintaining robust enough of a connection such that any node that is not the sound source can fail with no longer than 10s of an impact on the streaming of audio.

II. Design Requirements

The main aforementioned design requirement, paired with the requirements for the four broken down aspects, will make up the total requirements for our system in order to be comparable to market standard.

Firstly, for the area of Sound Quality, our metrics related to the sub-features of low-jitter software, low-noise circuitry, active noise cancellation, and passive noise isolation. The networking software must reduce jitter to below 40ms without drastically increasing latency for a smooth listening experience for the user. For low-noise circuitry, we will need to keep the complexity low and compact such that it can fit within a 1800mm2. We will also need to keep the power consumption less than 125mA during either playback mode. Finally for the circuit, it must have a difference in signal-to-noise ratio of less than 6dB and total harmonic distortion plus noise of less than 1% as compared to the Bose QuietComfort 35 version 2's Related to active noise cancellation, the signal-to-noise ratio between our output and the input audio needs to be at least 12dB and the total harmonic distortion must be less than 3% in order to achieve successful cancellation. And lastly, for the passive noise isolation in the form of cushioning, it needs to be less than 120cm3 in volume and must have at least a 6dB sound pressure level reduction in order to provide both comfort and assist in the noise cancellation.

The audio sharing system will send audio from a broadcasting headset, or root node, to a series of receiver headsets, or sub-nodes. The latency from the root node to any given sub-node must not exceed 180ms. The failure of any given sub-node should not cause a disruption to system operation exceeding 10 seconds. In order to meet the bandwidth requirements of the chosen codec, SBC, a minimum effective link speed of 450kbps is required. In this way, users will not experience noticeable desynchronization or disturbance in their audio.

Regarding usability, we will need to have a number of metrics met for battery life for a successful experience. We have determined a playback time of over 40 hours, a multi-playback time of 6 hours and a standby time of over 450 hours should facilitate this. For charging we are looking to have a USB charging unit within less than 900mm2 and have a performance of 3A (5V), and a wireless charging coil within a dimension of 40mm by 60mm and a performance of 1A (5V).

Lastly for the basic functionality we are looking Bluetooth playback conform to the A2DP profile and the overall power consumption remain less than 125mA during playback.

III. Architecture and/or Principle of Operation

Our device consists of a digital audio playback system with a novel digitally configurable analog noise cancellation system. Each headphone half consists of either an ESP32-WROOM-32D module or an ESP32-WROVER-IB module, a battery, a speaker driver, op-amp circuitries, and either a DAC or power management ICs. The headphones halves are connected via a ribbon cable that transmits power and data. Some portions of the headphone are asymmetric. One half will contain a 3D gesture sensing subsystem, and the other will contain wireless charging circuitry.

The ESP32 module is in charge of the headphone and is responsible for configuring all the other hardware. The two ESP32 modules are arranged in a master-slave pair, with once accepting commands from the other. The modules are responsible for controlling the other hardware inside the headset, as well as wireless reception, transmission and local playback of audio. The amp and driver receive an I2S signal from one of the two ESP32 modules, and then decode and amplify the signal, sending it to the drivers. Each headphone halve also contains a 2500mAh battery for power, as well as a USB-C port for charging and UART based debugging and firmware updates. The 3D gesture sensing subsystem emits an electric field and detects perturbations to the field, which allow the subsystem to recognize in-air gestures of the users of our device. The ESP32 module can then react according to the recognized gestures per a predefined FSM.

A. UI and UX

The device exists in two main modes: Single Playback and Multi-Playback. Within the Single Playback mode, users can choose to either play their audio as they would with any normal pair of headphones or enter the branch Broadcast mode. Upon activation of this broadcast mode, the user is able to trigger the creation of a mesh system created over WiFi. This mesh is then joinable by other headphones, creating a tree topology for the relaying of audio packets with the broadcasting headphone as the root node. In the second main Multi-Playback mode, the user can join an already established mesh that exists immediately around it, adding on as a leaf to the tree that has already been established. The user can then switch into a "Pause" mode that simply silences the incoming audio for the user. The switching between these modes is governed by inputs to the MGC3130 gesture control module and the Bluetooth/WiFi communication for the audio transfer will be controlled by the ESP32 module.

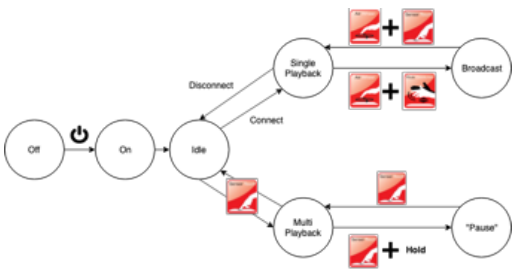


Fig. 1. UI FSM

B. Software Overview

The software itself consists of multiple of multiple tasks controlled by a FreeRTOS scheduler as well as our lightweight task dispatch system. Although we present the software as a single entity, the two ESP32 cores have different responsibilities, which are covered below. The slave ESP32 is responsible for accepting, routing, and dispatching network packets, controlling the multiplexer for I2S audio transmission, and responding to intra-headphone communication and gesture control. The master ESP32 is responsible for hardware control, drivers for Bluetooth audio, drivers for I2S audio transmission, and management of the state of the headphones via the control FSM.

C. Mesh Network Design

The mesh network allows for a root to disseminate music to multiples sub-nodes. It is a tree-based topology that is optimized for broadcasting from the root node to the sub-nodes. The mesh topology will be supported by multiple daisy-chained 54mbps WiFi networks. Each one-to-many collection of parents and nodes will comprise a separate WiFi network. Each node is capable of acting as both an AP and a Station. The parent nodes in the mesh will broadcast a special SSID that will indicate it is part of a mesh as well as what layer it occupies. Each root node will set a nonce in the SSID to differentiate separate mesh networks. All the parent nodes in the mesh topology subtree will have the same nonce bits in the SSID as the root node of the mesh. 15 bytes of the SSID will be reserved for the nonce. Since characters are restricted to the alphanumeric subset of ASCII characters, each byte can be one of 62 values. Thus, assuming a high-quality source of randomness, there is a 1 in 7.689097049E26 chance of collision for each additional overlapping mesh network.

Each node that wishes to join the mesh will first scan the SSIDs to discover parent nodes. Then it will contact parent nodes, going in ascending layer order. It will connect to the first parent node that indicates it has not reached the maximum node limit. A node will be considered lost if it does not receive data for 3 seconds. The node will then reset itself and try and join the mesh as if it was new.

The mesh will route data between nodes using a custom IPv6/ICMP stack. Not all of the packet fragmentation and broadcast features of IPv6 will be supported. In order to support routing, each node will maintain a list of its neighbors, as well as routes for all the IP addresses it is aware of in the mesh. In order to support smooth playback, the mesh network must support a minimum effective link speed of 450kbps to all nodes, as well as a reconnection speed of 10 seconds, and a packet jitter below 40ms.

D. Device Formfactor

One of the most prominent challenges in this project is accommodating not only the components required for supporting all the aforementioned features, but also the battery needed to power said components. To add to the challenge, both wireless charging and 3D gesture sensing require large flat non-conductive surfaces made available on our device, to house an

inductive coil and five electrodes respectively. Moreover, to be competitive with existing products in the market, our device must be in a reasonable formfactor such that it is not too bulky to fit on our users' head. As such, we jump-started our design process by scrutinizing an artist's rendition of a pair of headphones and then physically laying out various subsystems within the predetermined enclosure. Though this would not function as our final enclosure design, it gave us a rough idea on the constraints in terms of real estate that we must conform to, which cannot be obtained otherwise.

E. Hardware Considerations

Due to the aforementioned constraints in real estate, we have a stringent constraint on the size of our electronics. As a result, we must design our own PCB with SMD components in lieu of breadboards or perma-proto boards with breakout boards and through-hole components. Such a design decision incurs a large amount of engineering complexities that must be accounted for.

Moreover, within PCB design, there is a constant struggle among reducing the components' footprints, minimizing the number of different components, and avoiding extremely expensive parts. For example, a highly integrated IC might reduce the external passive components needed and thus reduce power consumption, number of components, and area occupied. However, such an IC could be prohibitively expensive due to budgetary constraints.

To aggravate the challenge, the optimizations as described above are non-linear, since trade-offs in one subsystem do not necessarily correspond linearly to trade-offs in another subsystem. For example, the integration of battery charging, monitoring, and protection and load-balancing circuitry into a smaller footprint allows for a more efficient design of power supply due to freed-up space, which in turn permits high power consumption elsewhere in the headphones.

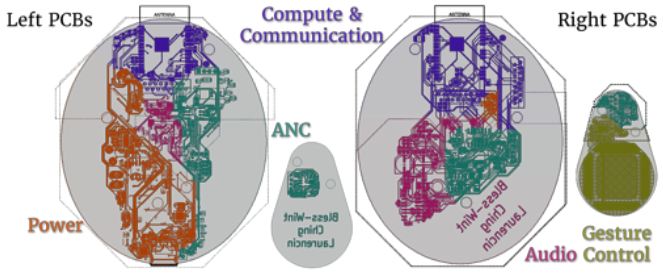


Fig. 2. Layout of PCBs labelled by functional areas

The PCBs are laid out in functional areas as indicated in the figure above. The full schematic is available in the Appendix section. Though the PCBs are eventually designed and manufactured per specification, they fail unexpectedly. It has been confirmed that, given functional components and correct PCBs, a populated PCB can become defunct over time, first slowly deteriorating and then failing precipitously. It is suspected that contaminated flux is causing such an issue, but in the interest of time, we decided to utilize the populated PCBs parasitically. Specifically, we only use the speaker drivers on them, paired with external DAC and ESP32 module; as such, the fault areas of the PCBs can be circumvented while salvaging some use from the remainder of them.

Moreover, to demonstrate the wireless charging and gesture control capabilities of our software design, we devised a deconstructed version of our headset with breakout boards and off-the-shelf components. In particular, a Flick HAT, a breakout board equipped with a MGC3130, interfaced directly with a Raspberry Pi, sending its reading with a ESP32 module via network; a universal Qi charging module, a breakout board equipped with BQ51013B (namely that wireless charging controller we were to use), supplies power to the battery charging/discharging circuitry whenever possible.

F. Software Design

The software has significant considerations with respect to the time of this project, as well as the compute power of the system. One of the most powerful mechanisms the design has is the intra-headphone communication path via I2S. This is primarily used for forwarding I2S packets between headphones, but it can also be used to enable complicated operations such as dual radio network operation and dynamic task scheduling. However, we've restricted its use, and the corresponding interactions between ESP32 modules for the sake of time and power consumption. The system has a 40hr battery life target, so the software cannot be wasteful, especially with wireless data transmission and complicated operations. To that end, the software has the ability to utilize the ultra-low-power coprocessor in the ESP32 when idle to avoid needless power consumption by entering "deep sleep." Also, the software has to be limited with respect to our time for implementation and debugging. It is difficult to debug embedded software, so we must go with more simple designs so that they can be constructed and debugged in a reasonable amount of time.

1) Network Stack Design

Our system uses a lightweight IPV6 stack for networking capabilities. It was designed to have minimal code size and functionality in order to work with the limited resources available on the ESP32. At the same time, it had to be robust to failure and changing network conditions, as it forms the foundation for our WiFi mesh. To that end, our networking stack does not allocate any dynamic memory, and features zero-copy operations whenever possible. The elimination of heap usage and excessive copying makes our network stack's behavior more predictable, stable and easy to debug. The networking stack implements UDP, ICMP6, and IP6 protocols. Only IP6 packets are supported, so from now in the context of the networking stack the word packet specifically refers to an IP6 packet unless explicitly stated otherwise.

When designing the networking stack, we had to choose between supporting IP6 and IP4 since we did not have enough time to code support for both. We ended up choosing IP6 because it Was better suited to the constraints of our system. IP6's SLAAC (Stateless Address Autoconfiguration) specification made it easy to give each node in our network non-colliding, self-assigned IP's, which was important because we did not want the overhead of adding a DHCP server into our network. Additionally, IP4 headers are variable length, while IP6 headers are fixed length with optional extensions. Because

the IP6 header was a fixed size, it made it easier to design a networking stack with fixed memory usage, as well as keep most network operations zero copy.

The design of our Networking Stack is split into two major components: core and drivers. The core of the networking stack contains components that implement the RFC specifications for UDP, ICMP6, and IP6 as well as general packet handling and routing. The core can be ported to any system with the C standard library and a compatible GCC compiler. The drivers contain system specific code and are not portable to other systems. The drivers and the core are separated by an abstract entity called the network interface which provides a reusable API to implement drivers without modifying the core. This separation enabled us to develop and test the core on a regular MacOS laptop, which significantly aided debugging and development productivity. Since we had separate drivers maintained for MacOS and the ESP32, we were able to freely move back and forth between the two devices when debugging or adding new features. Unfortunately, the additional discipline and time required to maintain the boundary between core and driver became too time-consuming to maintain in the last couple weeks of development, so after we became confident the network stack worked we stopped actively maintaining MacOS compatibility and focused solely on the ESP32. One big change we had to make to the design of the network core during development was we realized the ESP32's architecture necessitated processing packets in a separate thread then we received them in. However because the ESP32 hardware did not filter packets, we still had to perform packet filtering before we put the packets into a separate thread otherwise we would overload the threading system with all the incoming packets. The solution was to refactor the networking code to be more modular, and expose select chunks of functionality, such as packet filtering, forwarding and encapsulation to the device level drivers. This let us implement multithreading in the device level drivers while breaking encapsulation as little as possible.

2) Mesh Network Design

The effective link speed is the number of packets from the root node that can reach a given node in 1s. This is independent from the link speed of a single WiFi connection, as it takes into account the latency of the entire mesh infrastructure. A node that has a low effective link speed may still be able to play back audio smoothly, but it will miss the latency targets.

$$\text{Effective Link speed} = \frac{min(Linkspeed_i)}{N} - Nc$$

Fig. 3. N is the number of nodes between the target node and the root node. Link Speed is the speed of the connection between two nodes. C is a constant that represents the average processing time for a packet when it reaches a node.

Effective link speed is a critical metric and drives mesh design. A previous WiFi proposal, WiFi Proposal B was abandoned because it did not have a high enough effective link speed. WiFi proposal B was a novel connection-based design that has redundant links for data transmission. It was optimized for broadcasting from the root node and had no support for node-to-node or node-to-root communication. The network could sustain two node disconnection/failures per 10 seconds without disruption. However, that design (and thus its reliability) were abandoned in favor of the current design because it was too slow.

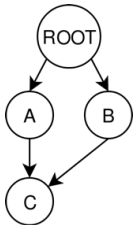


Fig. 4. "While audio does not require a particularly high data-rate, the paltry 1 MBPS link speed could prove troublesome. Consider the path from Root to C. The link speed is only 500kbps from ROOT→A and from ROOT→C if we consider a packet processing time of 10ms (c=0.03125) we get an effective link speed of 186kbps, which is already below our requirement. Node C will get eventually get all the audio packets, as each individual link has enough speed. However, it will have a latency of roughly 367ms. This latency shortcoming cannot be overcome by using multicast packets, as WiFi has significant issues with multicast reliability that could severely impact playback performance. These issues make it difficult to recommend implementing this mesh design."
- Two proposals for Mesh Networking on ESP32

G. Active Noise Cancellation

Due to the aforementioned constraints on signal-to-noise ratio and total harmonic distortion plus noise, two algorithms were considered in order to create the necessary filters to achieve this. The first of which was least mean square (LMS) algorithm. With this algorithm, one can create an adaptive cancelling algorithm, but the main issue is that it is a stochastic algorithm. In this case, there is some degree of randomness that exists within the model itself and it does look to other past data to create the current filter. In comparison, recursive least squares (RLS) filtering is an adaptive filter algorithm that is deterministic, in that the output is fully determined by the initial condition and the corresponding input parameters. It also is shown to converge faster than LMS. With these two main facts, as well as the following mathematical definition of RLS, for our purposes of a fast-responding algorithm that updates its filters with new incoming information, RLS was the best of the available options.

The approximation of FIR filters produced from RLS with an analog counterpart is a non-linear approximation, since the frequency response of a second-order filter is a non-linear function with respect to any one of the resistance values in said filter. Hence, a coordinate descent (CD) algorithm is used to find the best setting that can be used in the digital potentiometers such that their corresponding analog filter can achieve the desired frequency response.

Parameters: p = filter order
λ = forgetting factor
δ = value to initialize $\mathbf{P}(0)$
Initialization: $\mathbf{w}(n) = 0,$
$x(k) = 0, k = -p, \ldots, -1,$
$d(k) = 0, k = -p, \ldots, -1$
$\mathbf{P}(0) = \delta I$ where I is the identity matrix of rank $p + 1$
Computation: For $n = 1, 2, \ldots$

$$\mathbf{x}(n) = \begin{bmatrix} x(n) \\ x(n-1) \\ \vdots \\ x(n-p) \end{bmatrix}$$

$$\alpha(n) = d(n) - \mathbf{x}^T(n)\mathbf{w}(n-1)$$
$$\mathbf{g}(n) = \mathbf{P}(n-1)\mathbf{x}(n)\{\lambda + \mathbf{x}^T(n)\mathbf{P}(n-1)\mathbf{x}(n)\}^{-1}$$
$$\mathbf{P}(n) = \lambda^{-1}\mathbf{P}(n-1) - \mathbf{g}(n)\mathbf{x}^T(n)\lambda^{-1}\mathbf{P}(n-1)$$
$$\mathbf{w}(n) = \mathbf{w}(n-1) + \alpha(n)\mathbf{g}(n).$$

Fig. 5. Summary of the RLS algorithm.

After simulating this functionality in Python, we decided that for our final product we would no longer be including this form of active noise cancellation. This decision was made after noting it would require more integration time than previously expected, and with the resulting dysfunction of our PCBs, it would no longer be possible to integrate it at all.

IV. System Description

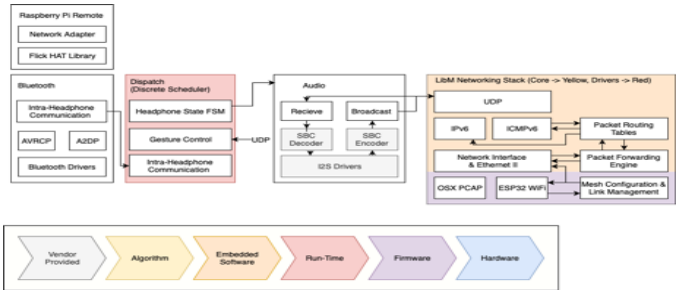


Fig. 6.

A. Software - Headphone State FSM

The control FSM maintains the state of the headphone. It receives FSM events, which cause it to update its state. The FSM state drives the state of the rest of components and turns other features (such as network and audio playback) on and off as required for the given state.

When the headphones are powered on, the subsystems (Intra-Headphone communication, Networking, Audio, and gesture control) are all initialized. The headphones will connect to the WiFi network, and then go to idle. At this point, the user can operate the headphones through TOUCH, HOLD, and WHEEL gestures as described in the *UI and UX* section.

B. Software - Dispatch

The ESP32 has limited resources and our code has a lot of real time tasks that must be executed. Dispatch is a lightweight discrete scheduler built on top of the Free RTOS scheduler. It is designed to simplify coordination of complex data across tasks and enable data driven event processing. With our system, functions only take up computational time if there is data for them to process so time is not lost polling for data. Dispatch also keeps track of which data is in use and allows data to be freed when it goes out of scope.

It functions by associating one or more functions are registered to an event tag. When something inside the program makes a call to `dispatchEvent(event_tag,data, callback)`, the event tag, data and callback are placed into a queue and returns immediately. The next time the dispatch system runs it pulls the first event added to the queue and looks up all the functions associated with the event tag, executing them sequentially. Once all the functions have been executed, it triggers the callback. This callback enables cleanup (e.g. freeing buffers) to automatically occur after the data has been processed.

C. Software - Gesture Control

An Interrupt Service Routine (ISR) is connected to a pin on the gesture sensing chip (MGC3130) in addition to an I2C communication line. When a TOUCH, HOLD, or WHEEL gesture is detected, an interrupt is raised and the ISR inserts a task into the lightweight dispatch queue to read the gesture sensor. Once the gesture is read, it will generate an FSM event to respond to the user's action.

D. Software - Intra-Headphone Communication and Synchronization

The intra-headphone communication is used to relay information between the headphones. The primary information communicated over this link is state information about the FSM. Each message consists of 64 bits and contains a command as well as data.

The FSM_UPDATE command is issued from the master to the slave in order to force the slave to the same state as the master. The state is derived from the FSM, so this mechanism enables clean synchronization between the master and the slave. The GET_UI_EVENT command is issued by the master to the slave to request the last UI gesture event that has occurred. In this way, the master can receive updates from the slave. A test is performed on power up to make sure the slave is present, and the right software revision is loaded.

E. Software - Audio System

The audio system is composed of a pair of send and receive buffers connected to SBC encoders and decoders. The SBC encoders and decoders were originally developed by Broadcom, and we ported them to the ESP 32.

When the user is in BROADCAST mode, they must first be connected to an A2DP compatible Bluetooth source via the Master ESP32. The Master ESP32 then outputs the Bluetooth audio over I2S. The slave receives the audio over I2S and compresses it into groups of SBC frames at 320kbit/s using the SBC. Each SBC frame takes up 115 bytes and contains 128 samples of audio (2.9ms). With default settings, two SBC frames (230 bytes) are encoded at a time and sent over the network to every node via multicast packets. Audio is also played back over the local speakers.

When the user is in MULTI PLAYBACK mode the audio system listens for incoming Audio Packets on UDP port 8000. As each audio frame is received, it is decoded into linear PCM

packets and placed into a playback buffer. When the playback buffer fills, it is written to the I2S output

The send and receive buffers are configurable and support up to 4096 samples (92ms). The default configuration will send 256 samples (5.8ms) of audio at a time, and buffer 512 samples (11.6 ms) at a time at the receiving end. If audio packets are received for more than 200ms it is assumed the connection has dropped and the audio system will automatically mute itself. It will continuously play silence until the signal is restored.

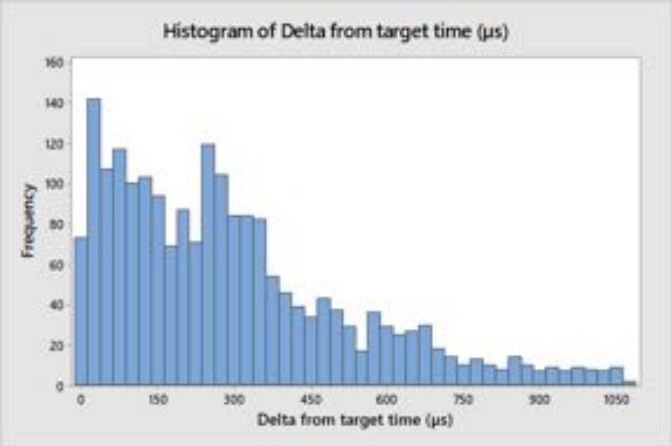


Fig. 7.

Our testing showed that our system is capable of smooth, jitter free playback. While roughly 10% of packets were late in our testing, if a packet was late it was only late by an average of 786us. This is well below the threshold of human perception. The graph above shows the histogram of late packets vs how late they arrived.

However, in environments with adverse WiFi environments, such as Wiegand gym we found while we were not dropping packets, the variance on when packets arrived skyrocketed, we were seeing packets arrive in bursts at up to 100ms late. This was most likely caused by WiFi's collision detection system. Because the 2.4GHZ environment was so saturated, the WiFi hardware was waiting longer for conditions to clear, and then sending packets in batches. We were able to remedy this by increasing the amount of SBC frames buffered by the broadcasting node from 2 SBC frames to 6 SBC frames. This worked but came at the cost of increasing latency by 11.6 ms.

F. Networking Stack

Our network stack can support packets with a MTU (maximum transmission unit) of 1200 bytes. One network interface is assigned per physical network device on the chip. Although we only use 2, the system can support up to 255 different network interfaces on a single device.

For the ESP32, we assign one network interface to our WiFi Access Point, and one network interface to our WiFi Station. Each network interface has 4 IP addresses. The EUI64 link local address (unique address generated from the MAC address), a multicast version of the EUI64 address used to respond to neighbor solicitations, a manually assigned IP6 address, and a multicast version of the manually assigned IP6 address used to respond to neighbor solicitations requests. Although the manually assigned IPs are not required for the system to work, it makes it a lot easier for developers to keep track of which IP belongs to which device.

When the headphones are turned on and connect to the WiFi network the routing table is initially empty, so it does not know where any of the other nodes on the network are or how to reach them. However, it does not need routing tables to handle multicast packets, so playback or broadcast can begin immediately. The headphone can then gradually discover the other devices around it, sending out neighbor solicitations whenever it encounters an unknown IP address and building up the routing table as needed. Since the process is "lazy" in that it does not try and discover routes until it is needed, it is impossible to say how long it takes how long it takes to discover all possible routes. The time to respond to a neighbor solicitation request is negligible (t<50us), so the time it takes to discover a route to a given IP is largely driven by how long it takes for the message to physically travel across the network. The worst-case time for a response to a neighbor solicitation request is typically

$$2 \times \text{Number of Nodes in Network} \times \text{Average Latency}$$

For our demo setup, the average time it took to a node to discover the route to an unknown IP address was 39ms.

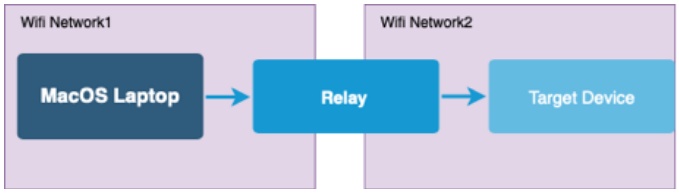


Fig. 8.

We tested the network stability and latency by flooding the system with ping packets and measuring the response. We setup our target device on our WiFi mesh network two hops away from a standard MacOS laptop. Except for the laptop, every device was running our software. We were able to push a maximum of 2800 kbps across the network and sustained 10% packet loss.

To measure per-hop latency, we performed another test with an identical setup, however instead of purposely trying to overload our network, we simulated packets with several SBC frames worth of audio (1024 bytes). We got an estimated per-hop latency of 7ms, which was well below our initial target of 30ms.

The next section will explain how packets are processed.

1)

Packets are received by the networking hardware, and then passed to the device level drivers and network interface. The device drivers then call a function in the core to tag the packet. Without copying, each packet's headers are analyzed. Then the packet is tagged with one of four actions before being returned to the drivers. DROP, FORWARD, RECEIVE, or FORWARD_RECEIVE. DROP means either the packet is not destined for the network interface it was received on or is

malformed. If a packet is marked as FORWARD, it is not destined for this network interface, but should be rerouted and sent towards the appropriate device. If a packet is marked as RECEIVE, the packet's final destination has been reached, and the packet should be decoded and further processed. If a packet is marked as FORWARD_RECEIVE, it should both be forwarded and received. This is used for multicast packets that should both be forwarded to other devices in the network, as well as received by the current device. FORWARD, RECEIVE, or FORWARD_RECEIVE it is put into a multithread safe queue with a maximum size of 20 packets.

2)

The core networking thread then receives packets from the queue. At this point, the packet has left the driver and is now being processed by the networking stack core. Benchmarking showed that our implementation is capable of processing a packet within 25 microseconds. (excluding time spent by networking hardware)

a)

If a packet was marked as FORWARD, the packet forwarding engine starts. The packets destination IP is looked up in the routing tables, and if a matching route is found, the packet is written to the network interface connected to the route. If no route is found, the packet will be dropped and an ICMP6 request will be generated to try and find a route to the destination IP. Originally packet switching was done at layer 2, unfortunately it was realized after the implementation was nearly complete that the ESP32 did not have support for WDS. WDS enables a WiFi packet to have different transmitter and source MAC addresses, which is needed for layer 2 switching over WiFi. When the ESP32 sends a WiFi packet, if the source MAC address does not match the transmitter MAC address, it drops the packet. So instead of supporting switching, the system was rewritten to forward packets at the IP layer (layer 3).

b)

When a packet is marked RECEIVE or FORWARD_RECEIVE the packet is sent to the UDP or ICMP6 subsystems.

The packet type value is used to figure out what kind of packet it is. As of now, only UDP and ICMP6 packets are supported. If the packet is ICMP6, is sent to the ICMP6 system. ICMP6 is responsible for responding to facilitating network management functionalities such as neighbor discovery via Neighbor Solicitations/Advertisements, duplicate address detection, and reachability detection. The echo/reply feature of ICMP6 was the basis of our latency and network reliability tests. If the incoming packet is UDP, the checksum of the packet is verified first to ensure the data Is correct. The UDP checksum is calculated by summing all the data inside the packet using 16-bit arithmetic, and then inverting the result and adding it to the packet's checksum. If the checksum is correct, the result should be 0. The checksum can be fooled by reordering groups of valid data on a 2-byte alignment but in our practical tests we never observed a corrupt packet pass the checksum.

3)

When the UDP packet passes the checksum, then it is routed to the application layer based on the port number. When the application is initialized, components outside of the networking core can be "bound" to UDP port numbers so they have a callback when data is received. Different UDP port numbers correspond to different features within the rest of the software. Port 8000 is for compressed audio packets, and port 9000 is for remote commands to the headphone state machine.

Since the process of sending packets involves many of the same components used in receiving, the process for sending a packet will be abbreviated. Either UDP or ICMP6 packets can be sent by the application. These functions accept data to send, as well as the IP address of the final destination. Any additional headers needed for either protocol is generated and placed into a separate buffer from the packet data. Then the IP6 subsystem will combine all the buffers into a single packet. This requires copying data, so this occurs at the last step, and the resulting packet is submitted directly to the network hardware. At the same time the packet is being assembled, the routing tables are used to lookup the appropriate path to the destination IP address. If a route is not found, the packet is rejected with an error and the ICMP6 system sends out a solicitation request to attempt to discover the route to the destination IP address. If the device is found and a response to the solicitation is received, the IMCP6 system will update the packet routing tables with the route to the destination IP.

G. WiFi Mesh and Demo Configuration

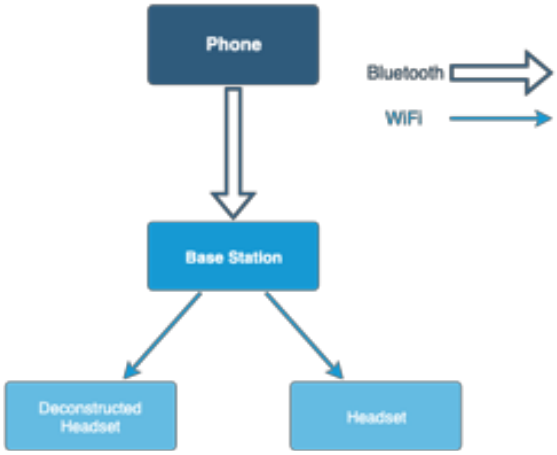


Fig. 9.

As part of discussions to narrow the scope of our project, we removed the automatic WiFi network selection process mentioned earlier in our report and replaced it with a static configuration. This configuration only specifies in what topology the WiFi network connections are made. All the underlying network routing is still done dynamically.

In our demo, the phone connects to a base station. The base station is a pair of ESP32's loaded with a modified version of our headphone software. They automatically go into broadcast mode when they turn on. Similarly, we also have deconstructed

version of our headset that has loudspeakers instead of small headphone drivers. Since we only have one pair of headphones, we had to develop these extra devices in order to demonstrate the mesh functionality. A common smartphone connects to the base station, which then simultaneously broadcasts the audio to both the deconstructed headset and our actual headset.

H. Battery Life

Battery life is calculated by dividing the theoretical capacity of our batteries (5000mAh in total) by the measured average power consumption over two minutes. In single playback mode, a battery life over 100 hours is achieved, while that in multi-playback mode is about 5.8 hours. The standby battery life is believed to be over 250 hours but cannot be ascertained due to inherent noise of the provided multimeter in our lab.

V. Project Management

A. Schedule

The diagram for our schedule can be viewed on Appendix III of our report, but we will provide an overview and the relation between tasks. In the below diagram, items that were finished on time are listed in the base color of each individual: blue for Ethan, purple for Michaela, red for Winston. The darker hues of each color indicate items that were completed later than the initial timeline, while lighter colors indicate items that were removed from the timeline eventually. The design and implementation of the mesh was fairly separate from the other portions of the project. Due to constant improvement and tuning of the system however, more time was devoted than initially intended. During the time Ethan was working on this, Michaela was be working on dimensioning out the physical enclosure design so that Winston could finish his layout. The layout took longer than expected as well, due to the fact that Winston was testing the system in simulation, and per the request of our advisor, updated some of the schematics to avoid potential failure points. Before Winston had finished this, the next physical design iteration occurred, was completed during the last few weeks once the PCBs had arrived and alterations could be made to properly fit and accommodate them. After mid-April, Michaela and Ethan began integrating mesh networking into the high-level software of the system. This took a bit longer than initially expected however, due to previously unaccounted for aspects, such as necessary data management solved by the dispatch. During this time as well, the final CAD design was developed, including both the cup and headband designs. Unfortunately, there were some drawbacks in the assembly of the PCB, as the system would regularly overheat and catch fire. With that, towards the end of our work period, we worked together to salvage what hardware we had and support it with other components, in addition to modifying the software, such that we had a viable demonstration.

B. Team Member Responsibilities

As previously touched upon in the, this project was broken into 3 main categories: hardware, software, and signal processing, with signal processing later being removed for the previously stated reasons. Ethan's main focus was on the software. This includes designing and implementing the mesh, as well as creating the related firmware to support it. Winston's main focus was on the hardware. This included the designing, simulating, and testing the schematic, layout, intra-headphone communication, and/or power supply and the implementation of the filters derived for active noise cancelling. Finally, Michaela's main focus was on the signal processing. This included, the development and testing of the RLS algorithm used to derive the necessary filters for the active noise cancelling system. Since this was relatively small related to the other areas, she was also tasked with designing the physical enclosure for the project that will then be printed. Later on when the RLS was removed, she worked on the high level software that supported the entire system along with Ethan. This included such items as the UI, dispatch, and intra-headphone communication, along with Ethan.

C. Budget

The chart for our budget is viewable in the Appendix of our report.

D. Risk Management

For our project risks related to design, we did our best to do our research on similar products beforehand so as to avoid pitfalls in designs. In the case of the mesh, this included researching different forms of wireless communication to determine the optimal form for our use case. For the hardware, we attempted to stick to chips that are popular and feature more pre-established schematics. For those features that we had to use chips other than the ones previously described, we had to meticulously read the data sheets in order to develop an effective design.

For our project risks related to resources such as components, we did our best to research the components that both fit our design requirements and the requirements of the routines and algorithms we are using in order to mitigate the chance of choosing ill-fitting components.

Finally, for our project risks related to schedule, we built in multiple iterations of our design to allow for time to change any aspects that did not work the way we had planned. Unfortunately, we were not able to utilize this additional amount of time on extra iterations, instead we spent the time on refining the first iterations.

E. Related Work

As one of our main objectives was to create a pair of headphones comparable to those on the market today, there are many products similar to ours. The key thing that sets our product apart from these however is the wireless audio transfer system that is integrated in. However, there are portions of the overall system that exist in a similar way across devices.

Most notably, for the Bluetooth and wireless mesh system for audio sharing, there are two systems that utilize these technologies to a similar end. The first of which are the Sonos Multi-room speaker sets which utilize a Bluetooth/WiFi combo chip. The speakers in these sets can be paired together in order to have synchronized audio throughout multiple rooms. This technology, however, has not been replicated in any of their

portable audio devices. Another example of this is an application called AmpMe which synchronizes audio between mobile devices in order to compete with portable speaker systems. They also employ AI and machine-learning in order to account for things like the few fractions of a second delay that exists when transferring data between packets in order to achieve perfectly synchronized audio. For our use case however, it does not make sense to optimize in this way as users will still have isolated audio during sharing, versus their use case of creating a collective speaker system.

VI. Summary

In the end, we were able to meet some of our design specifications and some we were unfortunately not able to meet. This was mostly due to the fact that our main hardware was eventually not usable. This resulted in some of our key metric areas, related to both the hardware and the signal processing to be lost. Fortunately though, we were able to meet our major specifications when it came to the software.

If we had more time, we could definitely improve both our system performance and functionality if we could complete a working version of our PCB. This would mostly likely involve a redesign that is one-sided and mainly focuses on the active noise cancellation circuitry. The rest of the hardware we could actually keep fairly similar to what we ended up with.

A. Future work

After completing our project, we realize there are a number of areas we wish to further explore past this semester. For improved signal for our mesh design, we would like to look into MIMO antennae in order to increase the complexity of our mesh connections. Similarly, we would also like to generally look into other aspects of antennae, utilizing the diverse capabilities of them. Also related to the mesh, we would like to look into Bluetooth 5 and explore the capabilities of that protocol over what we utilized for the mesh. We would also like to reintegrate active noise cancellation into our project. As mentioned in the summary, we would like to consolidate our PCB design to one-side and focus on achieve the appropriate circuitry for this feature alone on it. Finally, we would like to look into making some non-critical adjustments to our system, such as identifying a better gesture control system for our needs, adding debouncing to the gesture control system for better movement recognition, and having our battery packs in series rather than in parallel.

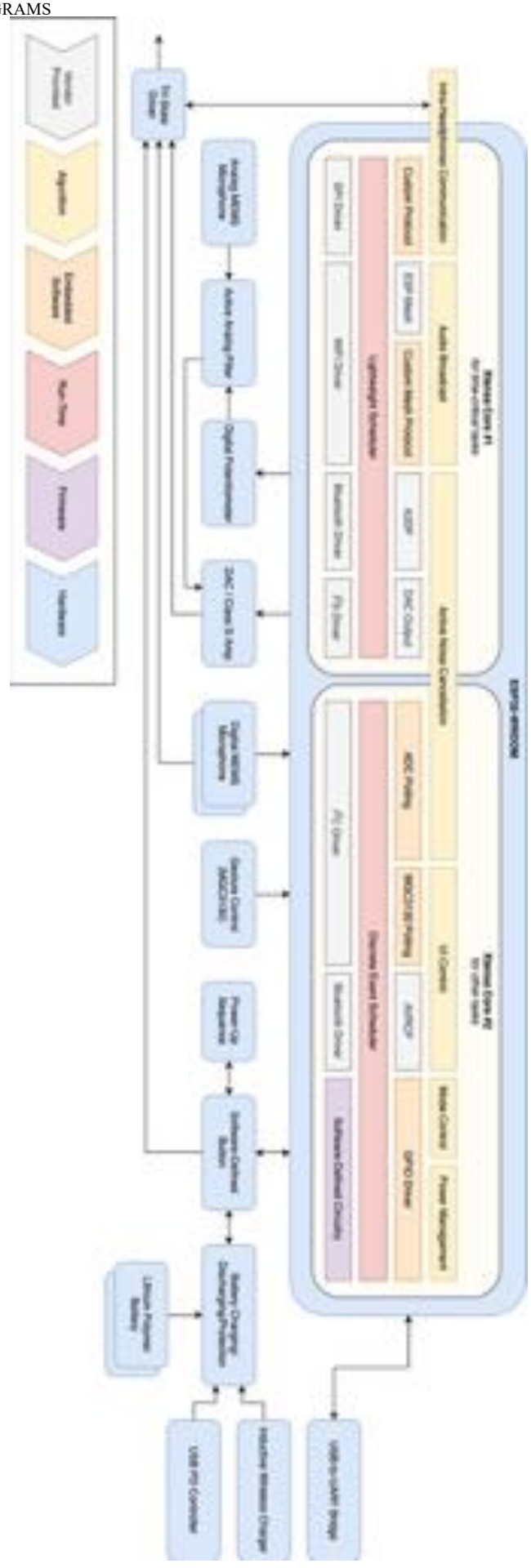
B. Lessons Learned

With this project, we have learned a number of lessons related to team dynamic and project design. Firstly, for team dynamic, we have learned that you need to make sure at least two people are sufficiently knowledgeable about any critical components. One of our critical missteps was that only one person was knowledgeable regarding PCB design, thus the other two were not able to help when things went wrong. With a project with such a short time span such as this, having at least two people be knowledgeable is key to sticking with the schedule. Regarding project design, there are a couple of lessons. First, always have a backup plan. We did have a backup plan for most of the aspects of our design, for example the PCBs, so we were still able to construct a final product, but without it would have been disastrous. The next lesson is to not prematurely optimize and focus on technical purity over results. Since this project is on such a short timeline, once something works well, it is better to move on and make sure the MVP is achieved. And finally, with PCB design on such a short timeline, try to keep the system as simple as possible. As mentioned before, we would like to have another design iteration where we simply make it one sided and only contain noise cancellation circuitry. This way, when debugging we would have less chances for error.

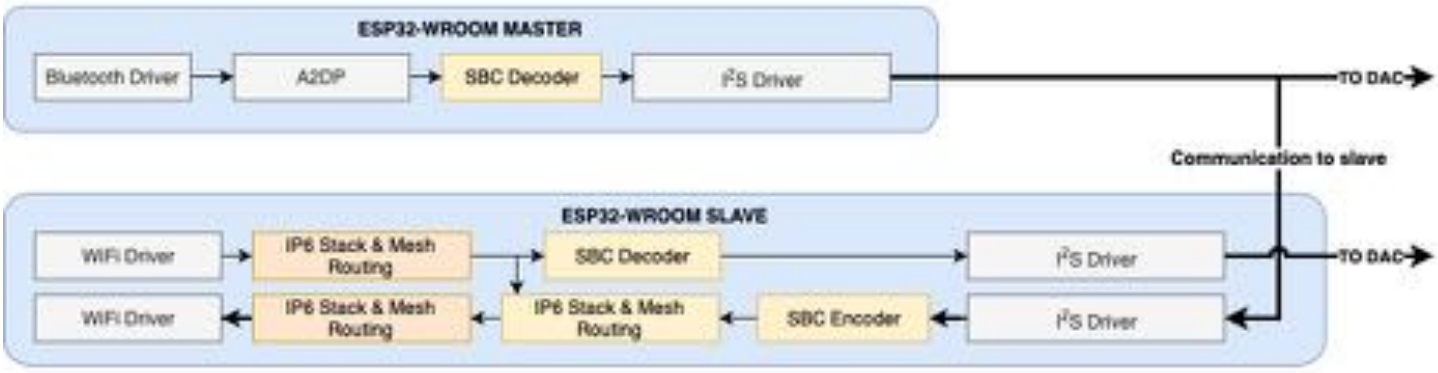
VII. References

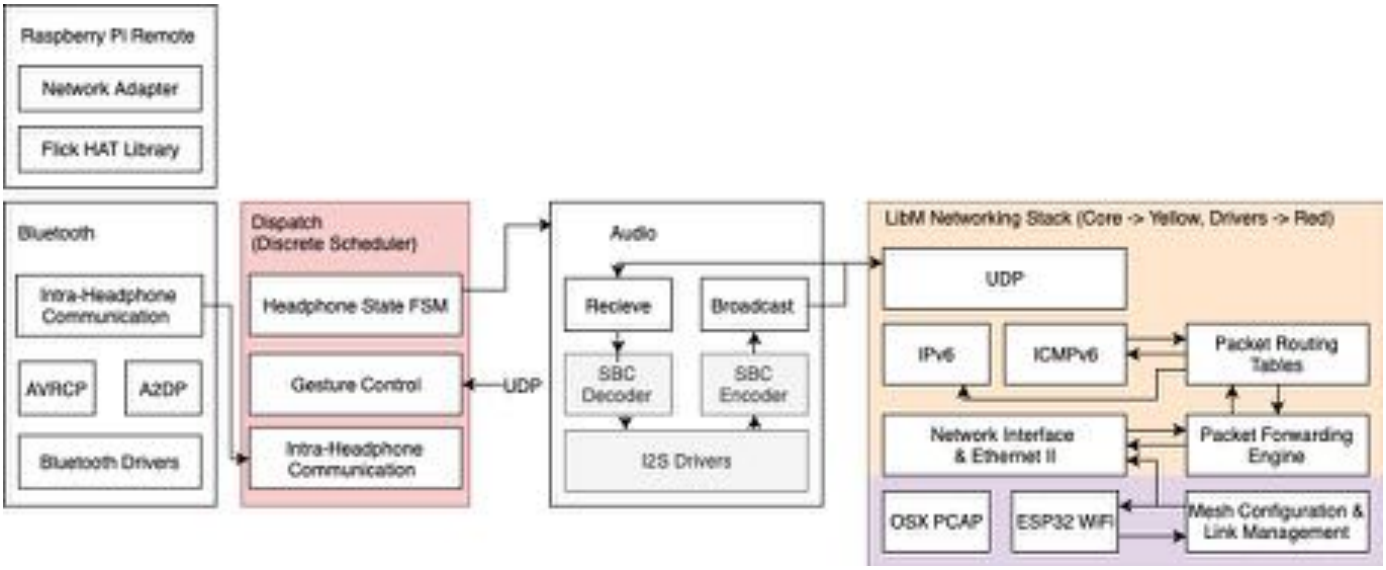
[1] https://www.ampme.com/about?locale=en_US
[2] https://venturebeat.com/2018/07/03/ampme-plans-to-kill-bluetooth-speakers-by-syncing-music-between-smartphones/
[3] https://www.mathworks.com/help/dsp/examples/adaptive-noise-cancellation-using-rls-adaptive-filtering.html
[4] https://www.mathworks.com/help/dsp/examples/acoustic-noise-cancellation-lms.html

APPENDIX I: SYSTEM BLOCK DIAGRAMS



Audio Packet Routing





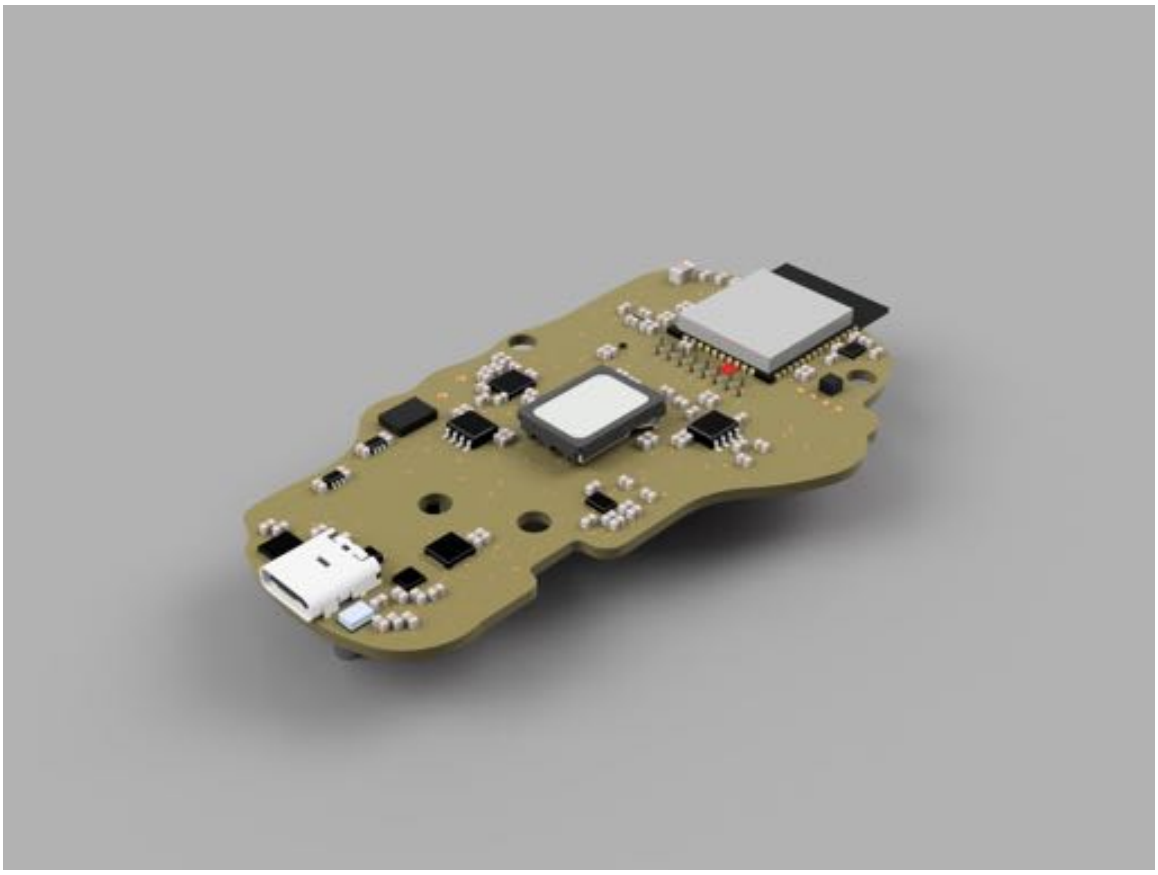
APPENDIX II: BUDGET AND BILL OF MATERIALS

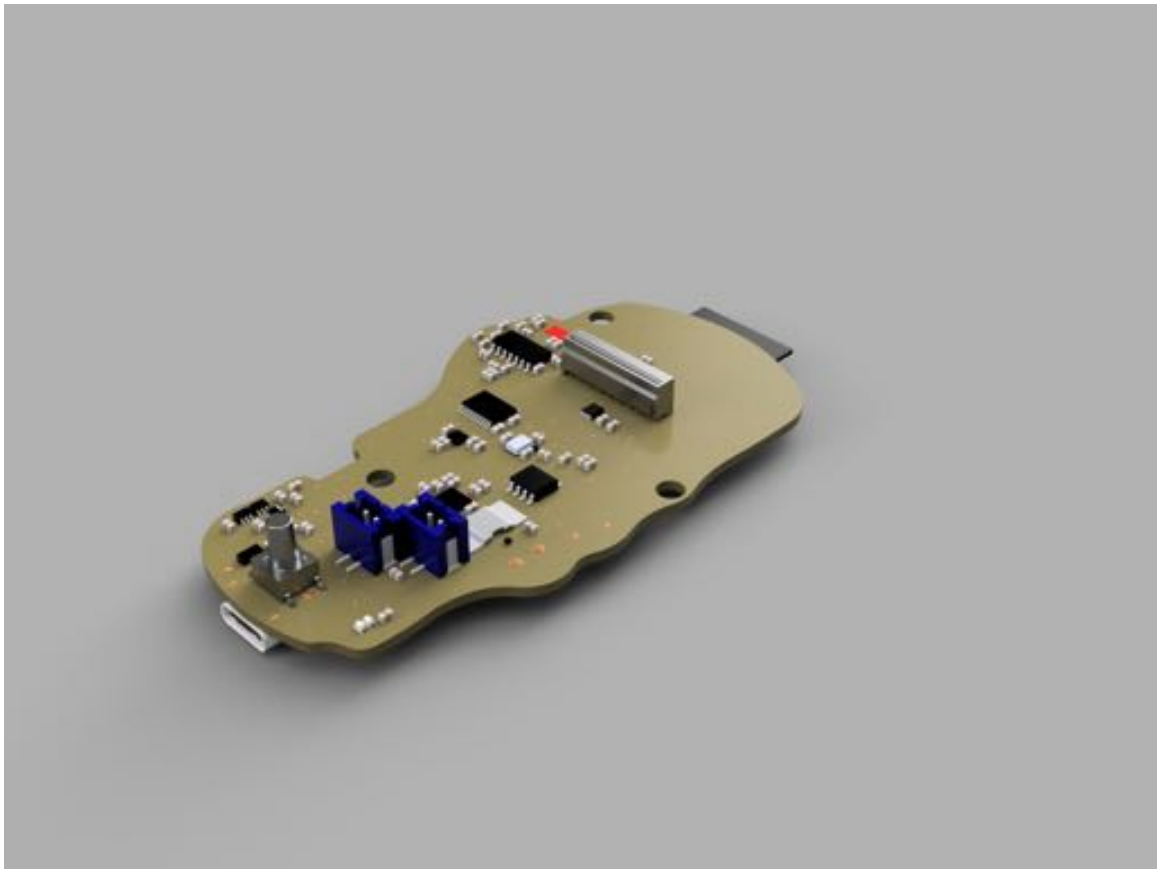
Category	Vendor	Item #	Order #	Cost ($)	Qty	Amt ($)
Hardware	Digikey	BAV74LT1GOSCT-ND	S19-679	0.12	2	0.24
		296-38885-1-ND	S19-680	3.49	2	6.98
		478-1215-1-ND	S19-681	0.1	2	0.20
		1276-1036-1-ND	S19-682	0.038	20	0.76
		1276-1045-1-ND	S19-683	0.0245	100	2.45
		1276-1009-1-ND	S19-684	0.039	12	0.47
		587-1256-1-ND	S19-685	0.124	24	2.98
		1276-1984-1-ND	S19-686	0.1	4	0.40
		399-1280-1-ND	S19-687	0.1	4	0.40
		1292-1425-1-ND	S19-688	0.1	4	0.40
		1276-1000-1-ND	S19-689	0.0123	100	1.23
		490-9966-1-ND	S19-690	0.7	4	2.80
		1276-2222-1-ND	S19-691	0.0282	100	2.82
		1276-1168-1-ND	S19-692	0.1	4	0.40
		PCF1132CT-ND	S19-693	0.531	10	5.31
		PCF1196CT-ND	S19-694	0.46	4	1.84
		PCE3753CT-ND	S19-695	0.13	4	0.52
		102-5642-ND	S19-696	2.07	4	8.28
		455-1734-1-ND	S19-697	0.55	4	2.20
		490-16641-1-ND	S19-698	0.13	2	0.26
		P10549CT-ND	S19-699	0.1	2	0.20
		CP2105-F01-GMRCT-ND	S19-700	1.56	2	3.12
		1904-1025-1-ND	S19-701	4.5	2	9.00
		1904-1017-1-ND	S19-702	9.32	2	18.64
		587-1923-1-ND	S19-703	0.052	25	1.30
		FUSB302B11MPXOSCT-ND	S19-704	1.75	2	3.50
		609-1831-ND	S19-705	0.2	8	1.60
		535-13615-ND	S19-706	4.91	2	9.82
		445-6388-1-ND	S19-707	0.14	4	0.56
		296-28104-1-ND	S19-708	1.21	3	3.63
		732-4980-6-ND	S19-709	1.31667	6	7.90
		DMP3085LSD-13DICT-ND	S19-710	0.45	4	1.80
		296-51365-1-ND	S19-711	0.5	4	2.00
		296-47593-1-ND	S19-712	0.46	4	1.84
		MAX9700DETB+-ND	S19-713	1.08	4	4.32
		MBT3904DW1T1GOSCT-ND	S19-714	0.21	4	0.84
		MC74VHC126DR2GOSCT-ND	S19-715	0.43	2	0.86
		MCP4351-103E/ST-ND	S19-716	1.39	0	5.56
		MCP23008T-E/SOCT-ND	S19-717	1.05	2	2.10
		MCP23008T-E/MLCT-ND	Winston	0.00	2	0.00
		MCP73871-2CCI/ML-ND	S19-718	1.84	2	3.68
		MGC3130-I/MQ-ND	S19-719	4.49	2	8.98
		1910-1059-ND	Winston	0.00	2	0.00
		MM3Z3V3T1GOSCT-ND	S19-720	0.15	2	0.30
		MMDT5451-FDICT-ND	S19-721	0.42	4	1.68
		NC7NZ17K8XCT-ND	S19-722	0.4	2	0.80
		NCV8570BMN300R2GOSCT-ND	S19-723	1.14	4	4.56
		296-36692-1-ND	S19-724	2.01	2	4.02
		296-35039-1-ND	S19-725	2.94	2	5.88
		568-10953-1-ND	S19-726	1.05	8	8.40
		CKN10822CT-ND	S19-727	0.31	3	0.93
		F3160CT-ND	S19-728	1.11	2	2.22
		423-1405-1-ND	S19-729	2.97	6	17.82
		423-1139-1-ND	S19-730	0.73	4	2.92
		TCK107GLFCT-ND	S19-731	0.412	10	4.12
		296-52032-1-ND	S19-732	1.67	0	10.02
		568-11912-1-ND	S19-733	1.48	2	2.96
		102-4484-1-ND	S19-734	2.03	3	6.09
		541-4378-1-ND	S19-735	2.29	2	4.58
		RG16P1.0KBCT-ND	S19-736	0.299	10	2.99
		RG16P3.4KBCT-ND	S19-737	0.318	10	3.18
		RG16P6.8KBCT-ND	S19-738	0.299	10	2.99
		RG16P10.0KBCT-ND	S19-739	0.299	10	2.99
		RG16P33.0KBCT-ND	S19-740	0.299	10	2.99
		RG16N68WCT-ND	S19-741	0.8	10	8.00
		RG16P100BCT-ND	S19-742	0.299	10	2.99
		RG16P220KBCT-ND	S19-743	0.299	10	2.99
		RHM1.00KADCT-ND	S19-744	0.12	20	2.40
		541-2803-1-ND	S19-745	0.134	10	1.34
		P20339CT-ND	S19-746	0.149	10	1.49
		P20359CT-ND	S19-747	0.149	10	1.49
		RHM10KADCT-ND	S19-748	0.0462	100	4.62
		511-1704-1-ND	S19-749	0.15	3	0.45
		A130434CT-ND	S19-750	0.132	10	1.32
		RHM51.0ADCT-ND	S19-751	0.0462	100	4.62
		2019-RK73H1JTTD66R5FCT-ND	S19-752	0.023	13	0.30
		P20511CT-ND	S19-753	0.1084	30	3.25
		P130BYCT-ND	S19-754	0.16	5	0.80
		311-2455-1-ND	S19-755	0.12	3	0.36
		511-1705-1-ND	S19-756	0.132	20	2.64
		P20214CT-ND	S19-757	0.149	10	1.49
		541-294KHCT-ND	S19-758	0.04	10	0.40
		P300KBYCT-ND	S19-759	0.16	5	0.80
		A130446CT-ND	S19-760	0.16	5	0.80
		RHM470ADCT-ND	S19-761	0.12	18	2.16
		A130448CT-ND	S19-762	0.16	5	0.80
		P20264CT-ND	S19-763	0.18	3	0.54
		AE11363-ND	S19-764	1.18	2	2.36

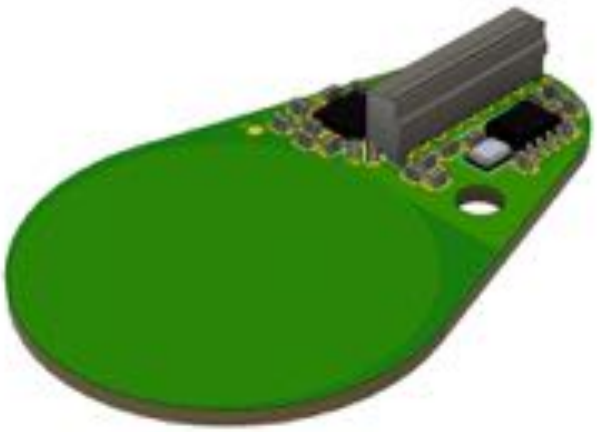
		HF16U-18-ND	S19-765	7.43	2	14.86
		P18853CT-ND	S19-766	1.507	24	36.17
		1528-1840-ND	S19-767	14.95	2	29.90
		1528-1841-ND	S19-768	7.95	2	15.90
	Adafruit	0328	Winston	0.00	2	0.00
		0353	Winston	0.00	1	0.00
		1699	Winston	0.00	1	0.00
		1901	Winston	0.00	1	0.00
		3006	Winston	0.00	2	0.00
		3351	Winston	0.00	2	0.00
		3619	Winston	0.00	2	0.00
	Amazon	B07DBNHJW2	Winston	0.00	1	0.00
		B07C1XD61W	Winston	0.00	1	0.00
		B01N0SB08Q	Winston	0.00	1	0.00
		B07546GQ9W	Winston	0.00	1	0.00
		B0718T232Z	Winston	0.00	2	0.00
		B01C6Q2GSY	Ethan	0.00	1	0.00
	Jlc Pcb	8534009000 (Y9)	S19-846	3.00	1	3.00
		8534009000 (Y10)	S19-843	4.00	1	4.00
		8534009000 (Y11)	S19-849	5.00	1	5.00
		8534009000 (Y12)	S19-850	4.00	1	4.00
		SMT STENCIL	-	6.03	1	6.03
		SMT STENCIL	-	6.03	1	6.03
		SHIPPING	-	25.69	1	25.69
		Subtotal				397.93
Enclosure	Maker Space	FORM II PRINTER	-	130.18	1	130.18
	Amazon	B07GQXK7C7	Winston	0.00	1	0.00
	Home Depot	051141395234	Winston	0.00	1	0.00
		051141929583	Winston	0.00	1	0.00
		051141333922	Winston	0.00	1	0.00
		051141405506	Winston	0.00	1	0.00
		051141405513	Winston	0.00	1	0.00
		051141405520	Winston	0.00	1	0.00
		021200092923	Winston	0.00	1	0.00
		021200092923	Winston	0.00	1	0.00
		Subtotal				130.18
Others	Lab Supply	DELL MONITOR	-	0.00	1	0.00
	Scrap	LASER-CUTTING WOOD	Michaela	0.00	1	0.00
	Amazon	B07DBX67NC	Winston	0.00	1	0.00
		B07GK1QR56	Winston	0.00	1	0.00
		B01L8LLP2G	Winston	0.00	2	0.00
		B014EV0G3G	Winston	0.00	1	0.00
		B06ZYN4MC7	Winston	0.00	1	0.00
		Subtotal				0.00
		Total				528.11

APPENDIX III: PHOTOS OF DEMO SETUP

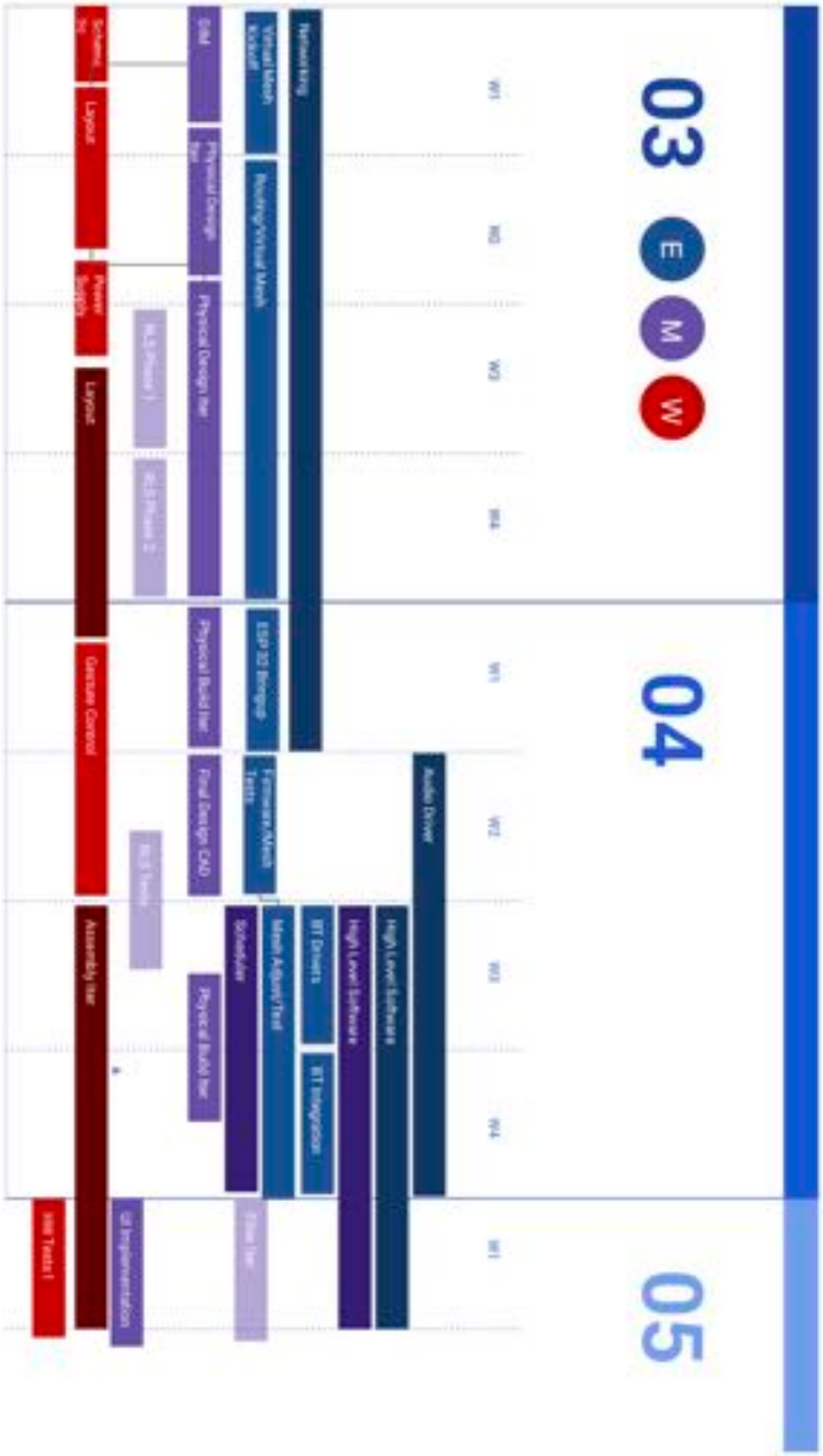
APPENDIX IV: PCB DESIGN



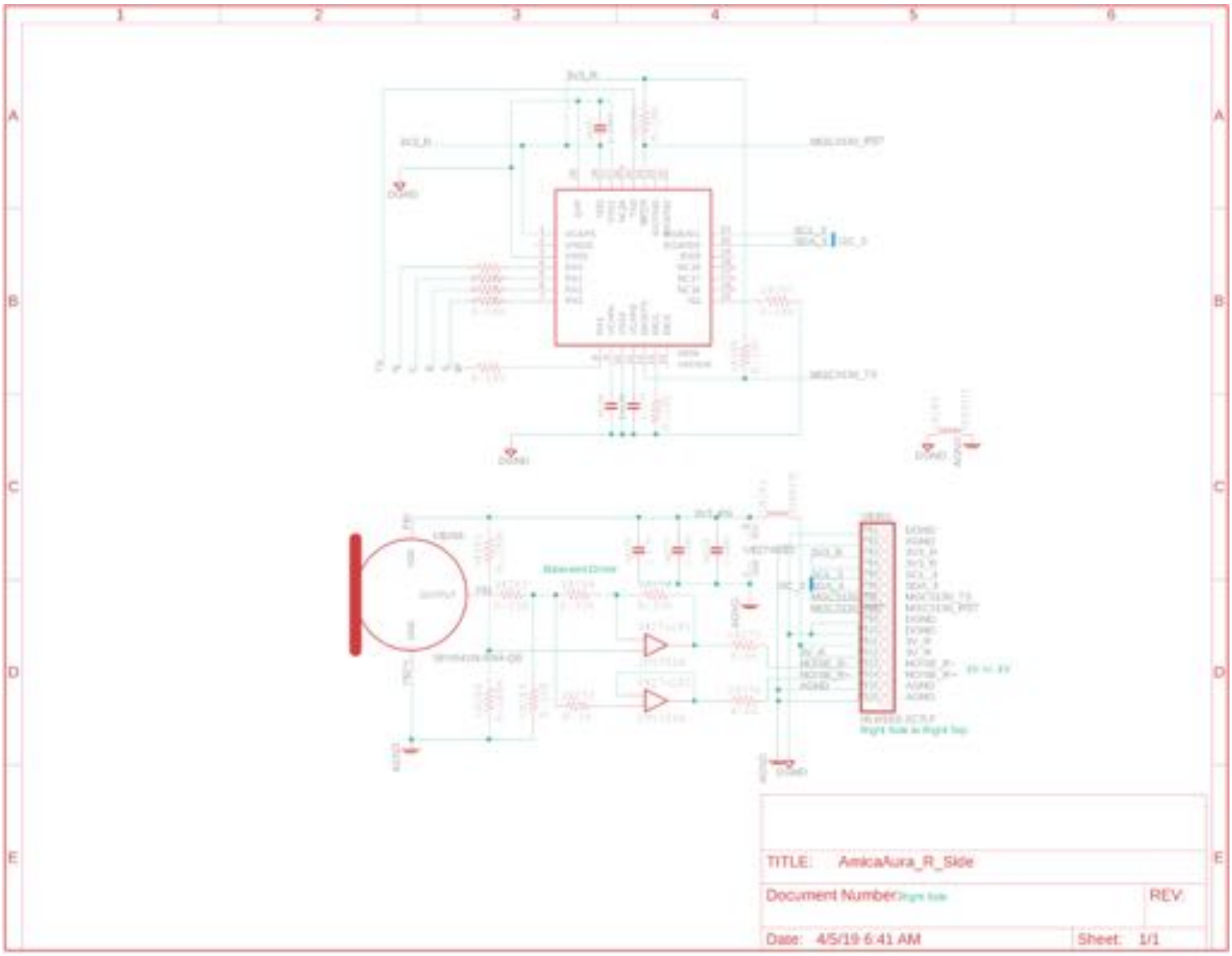


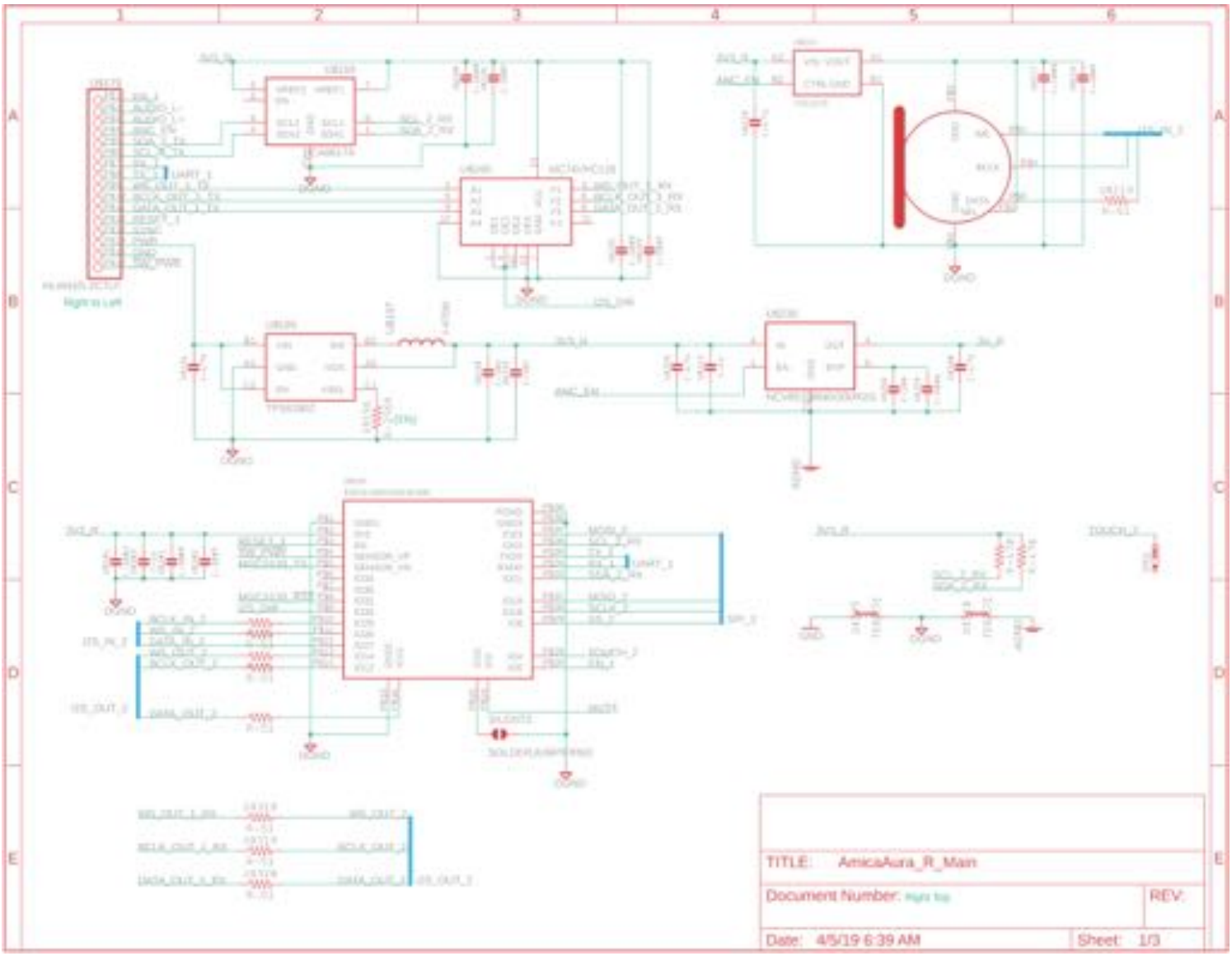


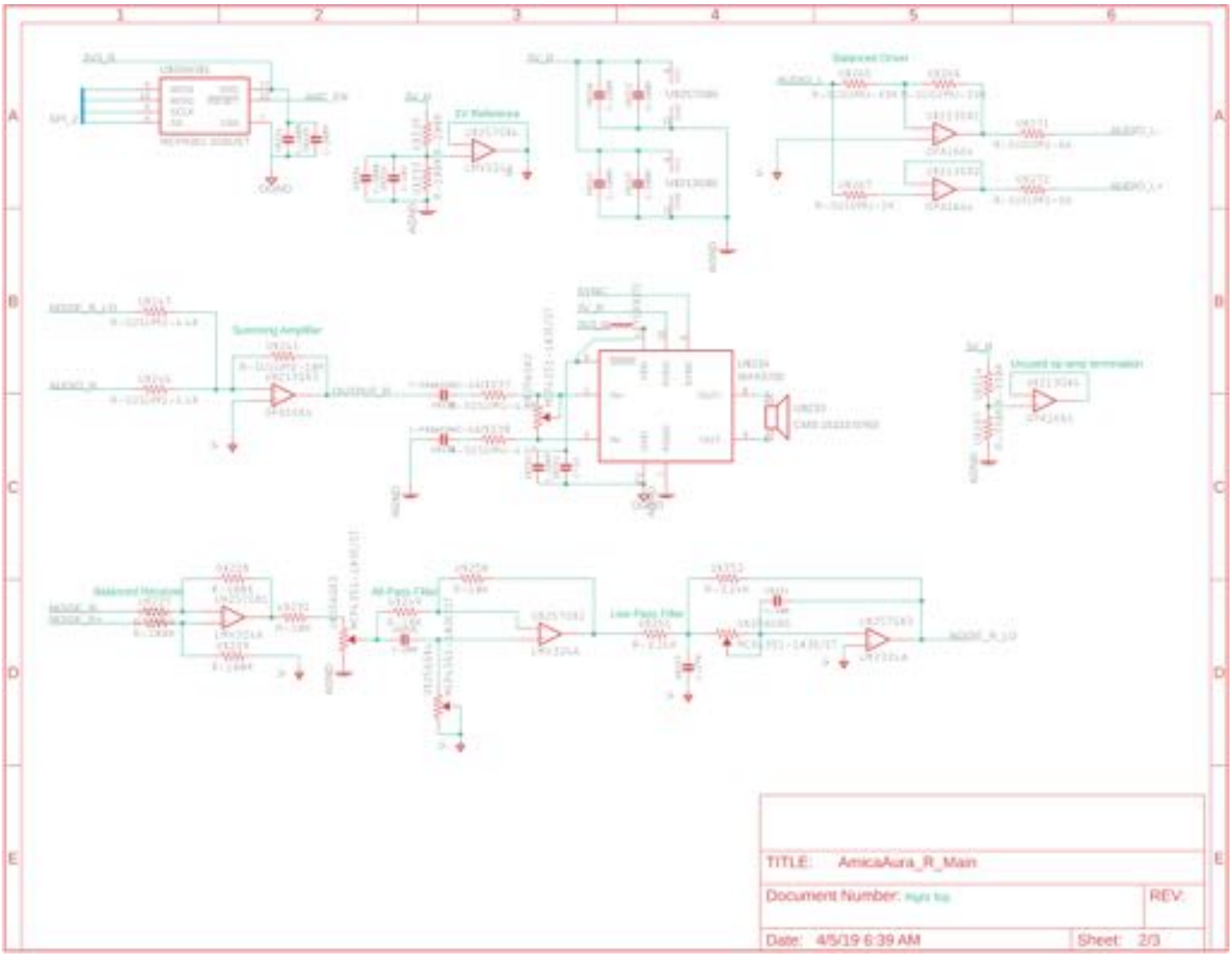
APPENDIX V: GANTT CHART

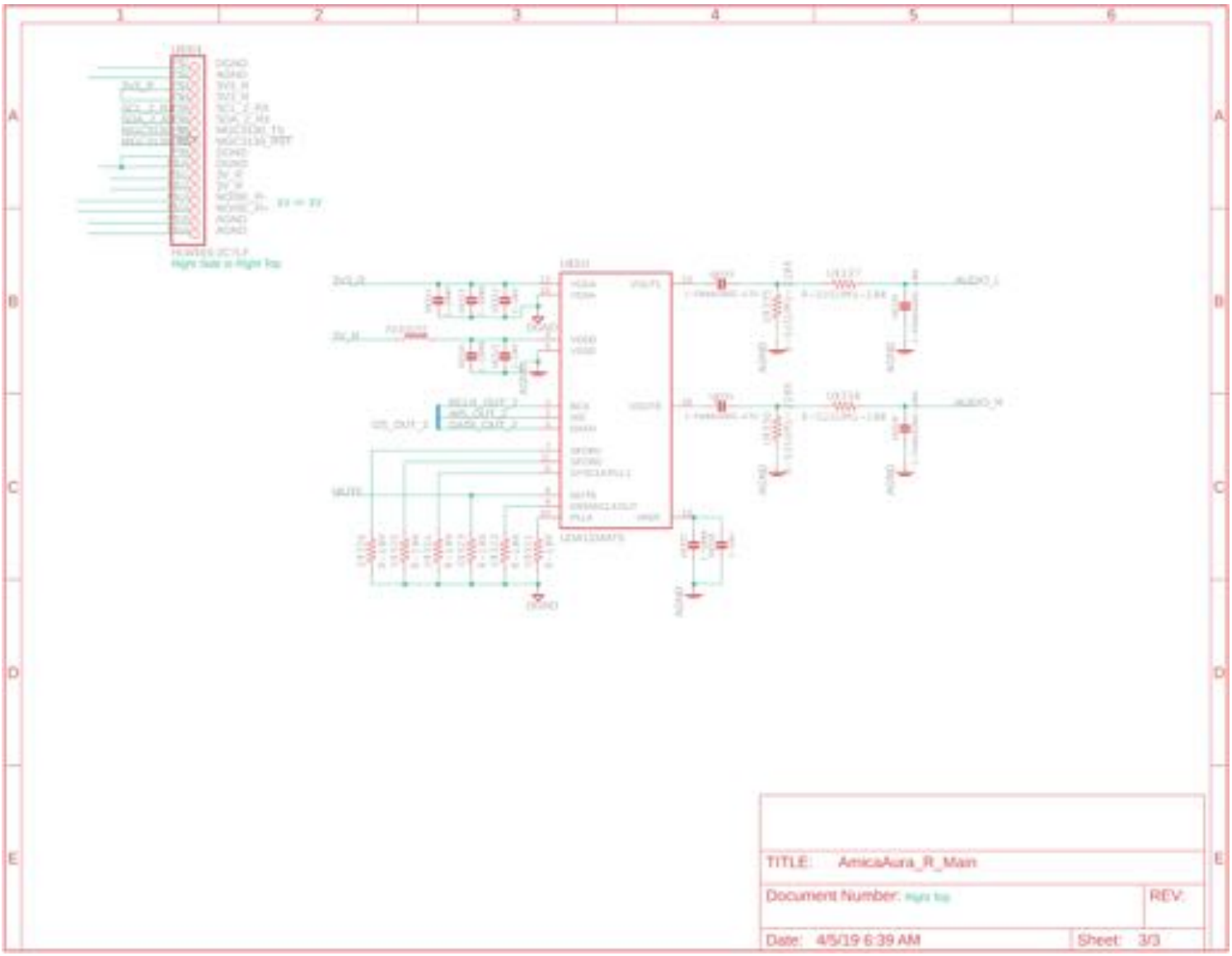


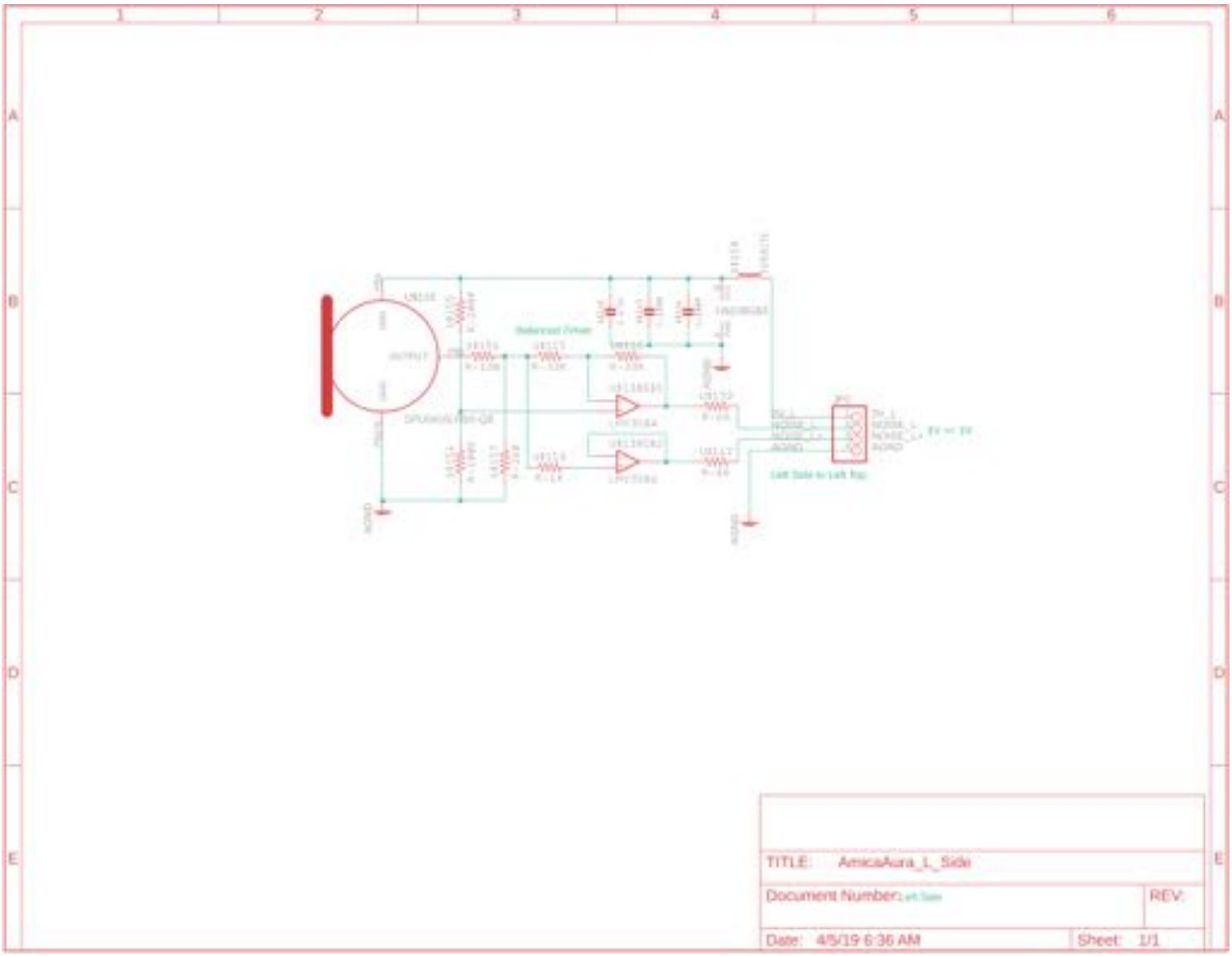
APPENDIX VI: SCHEMATICS

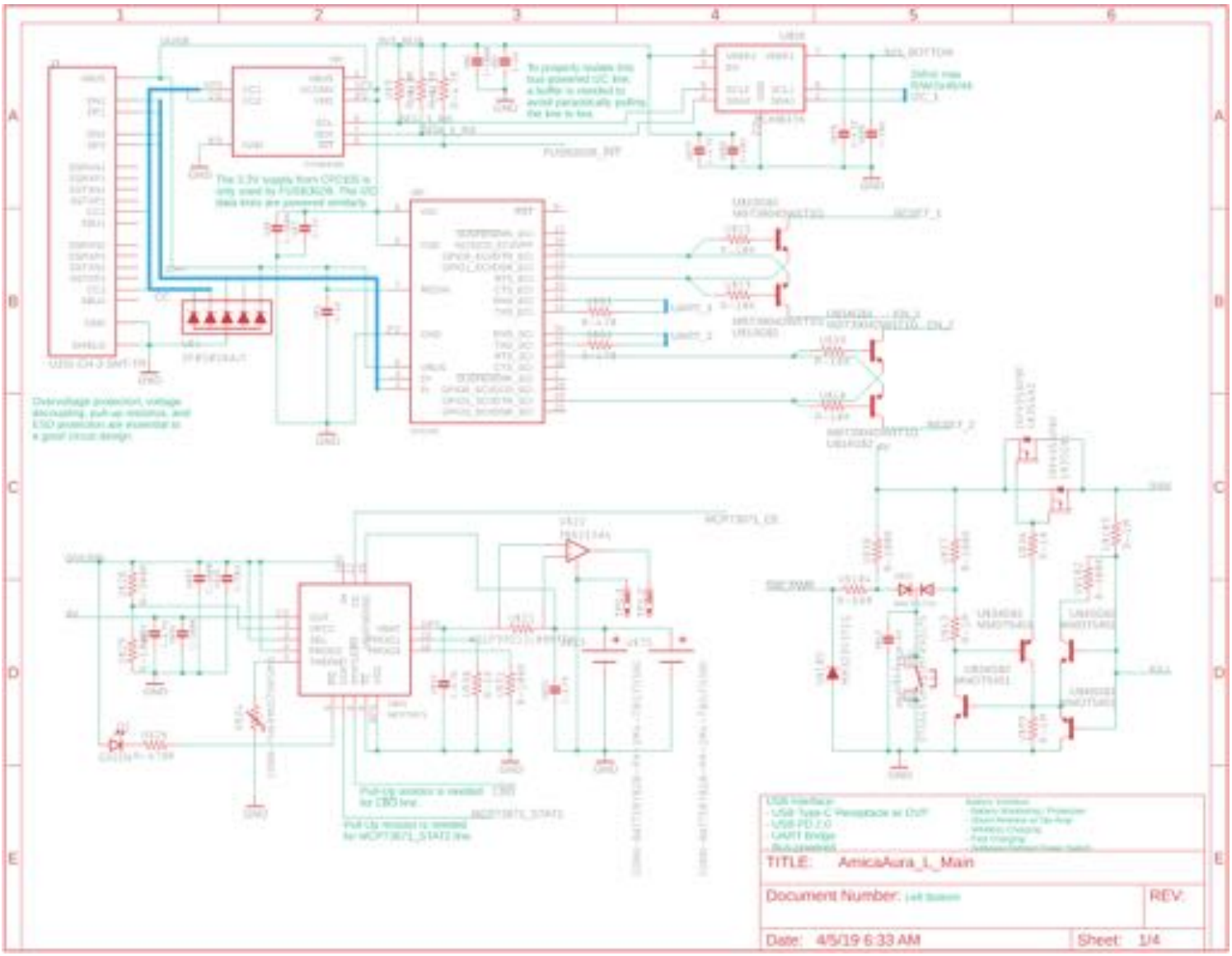


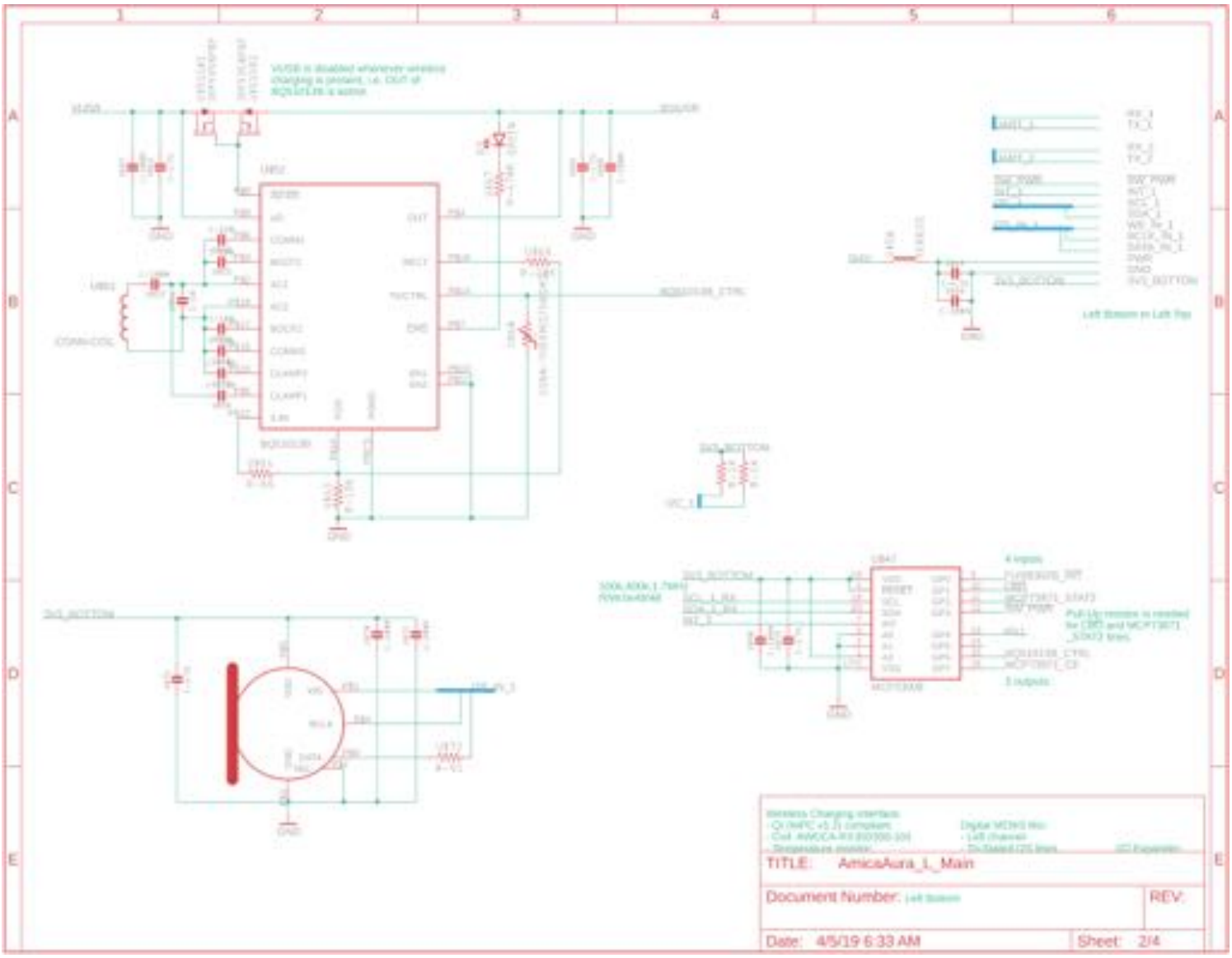

TITLE: AmicaAura_R_Main
Document Number:
REV:
Date: 4/5/19 6:39 AM
Sheet: 1/3

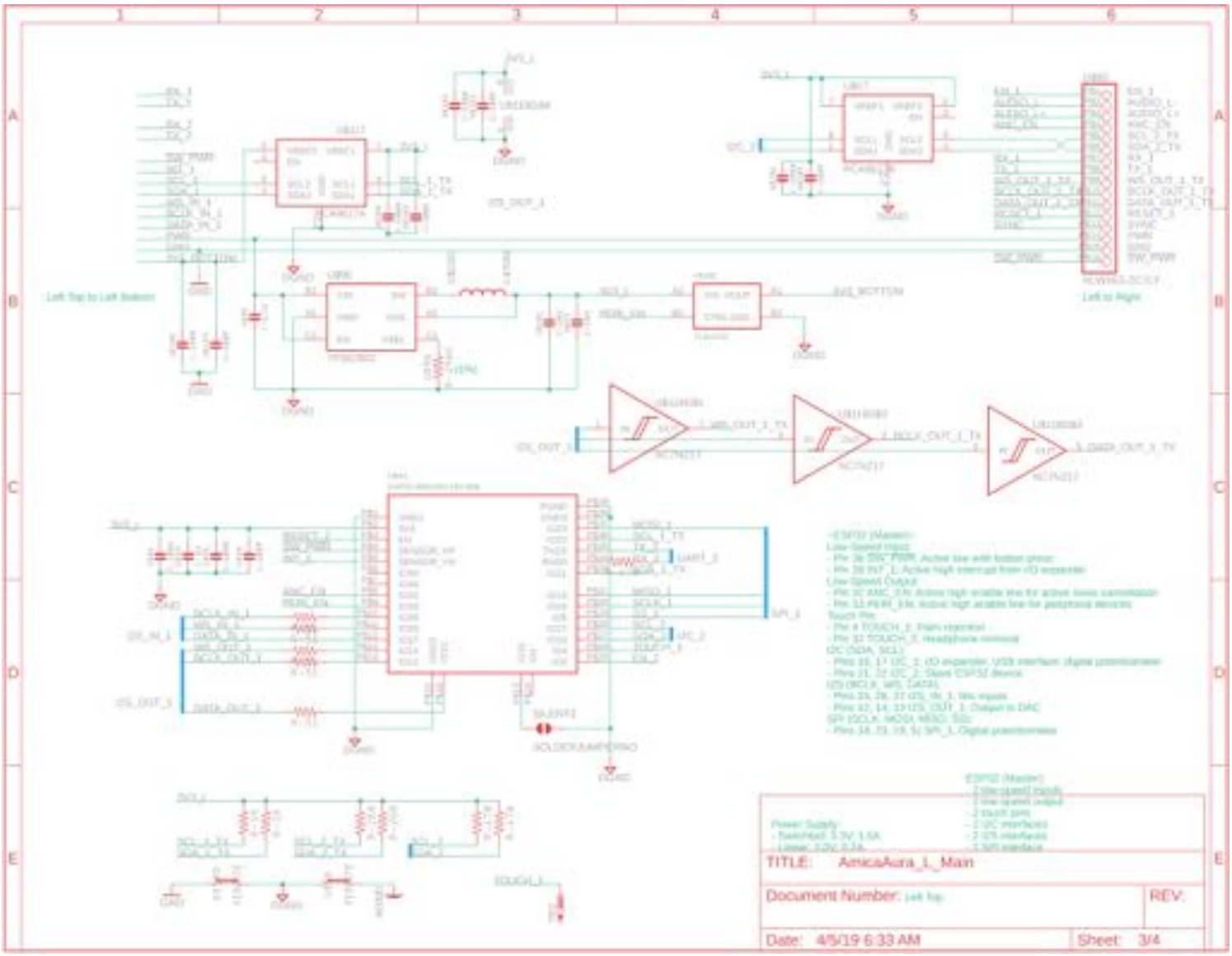

TITLE: AmicaAura_R_Main
Document Number:
REV:
Date: 4/5/19 6:39 AM
Sheet: 2/3

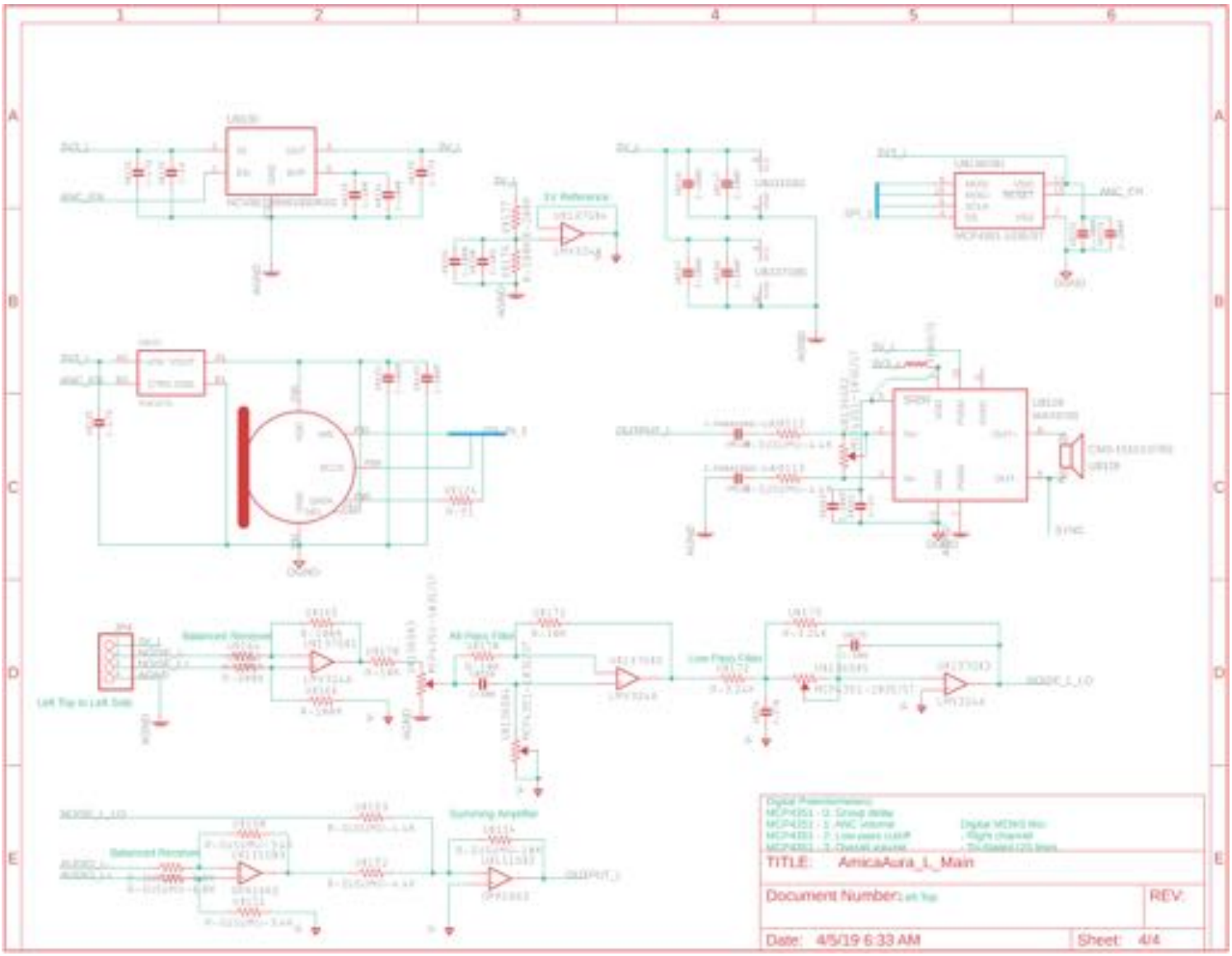

TITLE: AmicaAura_R_Main
Document Number: Right Top
REV:
Date: 4/5/19 6:39 AM
Sheet: 3/3


TITLE: AmicaAura_L_Side
Document Number:
REV:
Date: 4/5/19 6:36 AM
Sheet: 1/1


TITLE: AmicaAura_1_Main
Document Number:
REV:
Date: 4/5/19 6:33 AM
Sheet: 1/4


TITLE: AmicaAura_1_Main
Date: 4/5/19 6:33 AM
Sheet: 2/4


TITLE: AmicaAura_1_Main
Document Number:
REV:
Date: 4/5/19 6:33 AM
Sheet: 3/4


TITLE: AmicaAura_L_Main
Document Number:
REV:
Date: 4/5/19 6:33 AM
Sheet: 4/4