Bluetooth Stereo Network[1]

ECE 445

Jeff Wheeler, Rishi Ratan, Jerry Sun[2]

December 12, 2012

[2]Prof. Andrew Carl Singer, Prof. Brian Lilly. TA: Justine Fortier

Abstract

Our group designed and built a stereo network that streams audio from a Bluetooth-enabled device and automatically handles playback from the speakers closest to the user. The system is comprised of a hub to which the user's device connects to, and speaker adapters, which enable any speaker with a 3.5mm audio jack to interface wirelessly to our hub. Each unit consists of a Bluetooth chip and a microcontroller. The hub broadcasts audio to the speaker adapter, which is constantly sending back signal strength data about the user's audio device. Each of our components is powered by a lithium-ion battery and charged over USB. With our Bluetooth stereo network, a user can enjoy his/her music uninterrupted from the living room to the bedroom without having to worry about connecting or muting any speakers.

Contents

List of Figures

List of Tables

Listings

1 Introduction

1.1 Project Overview

We sought to design and develop a Bluetooth-powered stereo network that accepts audio data from user equipment (UE) and seemlessly transfers this data wirelessly to speakers closest to the UE location. Numerous Bluetooth-powered speakers currently exist in the consumer market, however none of them provide location-specific audio playback. The product we have developed over this semester is unavailable in the wireless audio marketplace and greatly streamlines the wireless audio playback experience for consumers.

1.2 High-level Design

We have developed a stereo network in which a user can play music from their phone or other device over Bluetooth (BT) to a centralized hub, which will then forward the audio data to the speaker network, choosing the appropriate speaker to play audio depending on its proximity to the UE. The key to our design is the requirement that the UE pairs only with the hub, and not the individual speakers, which greatly enhances the user experience. Networks built with our product include speaker adapters like those currently available in the market, but modified to observe how strongly they "see" the user equipment (by measuring the Received Signal Strength Information, RSSI, of the BT signal transmitted from the streaming device) and to relay this information to the hub. Figure 1 shows the highest-level topology of our products, forming a typical network with the UE and two speakers.

The most significant features and benefits of our product are included in Table 1.

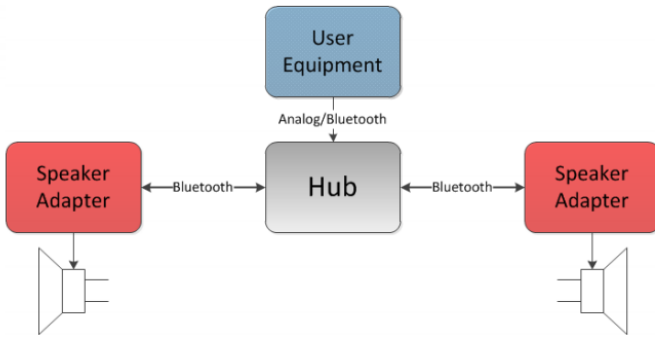

Figure 1: Overall System Design

Features	Benefits
1. Compatible with any Bluetooth enabled audio device.	1. User-friendly with plug-n-play functionality.
2. Compatible with all existing speakers on the market that support 3.5mm audio jack.	2. Superior form-factor and no additional HW/SW needed.
3. CD quality uncompressed audio.	3. Seamless audio packet handoff amongst speakers and range for BT playback extended due to addition of hub in central location.
4. Audio range of up to 30 feet from hub to speaker and from speaker to UE.	

Table 1: Features and Benefits

1.3 Design Revisions

We faced many design challenges and revised our overall system level designs several times in response to these challenges. Compared to earlier stages of the project, we were able to reduce complexity by removing several discrete ICs, which both reduced the parts cost (every component we now use was used in the original circuit) and reduced design overhead. We removed the circuits for the PurePath wireless audio, our audio codec, and the power management IC. In our updated designs, we have demonstrated a solution that is very well integrated: the internal I^2S support, battery charging, and Advanced Audio Distribution Profile (A2DP) streaming capabilities of the Bluegiga WT32 BT IC are used in order to achieve superior performance.

1.3.1 PurePath

The PurePath (PP) unit consisted of a TI-CC8531 PurePath IC and several passive components including a crystal oscillator. The PP IC in the hub was responsible for encoding and decoding the PP radio signals. The MCU would have used SPI to speak to the PP IC in order to control which speaker adapter the device transmitted audio data to. In the original design, the BT unit would have sent I^2S data it received over A2DP to the PP unit for rebroadcasting; a new BT IC replaces the PP unit for this functionality.

1.3.2 Analog Audio Frontend

Our original design had audio data received by the PurePath unit in the speaker adapters. The PP IC can only output this data as I^2S so it was originally necessary to have an analog audio front-end that would transform this I^2S data into analog. Because the BT IC that we chose has integrated analog output, this component was no longer necessary.

1.3.3 Power Management

In our updated design we further reduce cost and device area by utlizing the battery charging circuit integrated into our BT ICs. The advantage of our revised solution is easy to recognize, as it eliminates the need for additional external power management IC. By using the battery charging circuit of the BT ICs we have demonstarted charging the 3.8V Li-Ion batteries, which can deliver roughly 8-hours on a single charge.

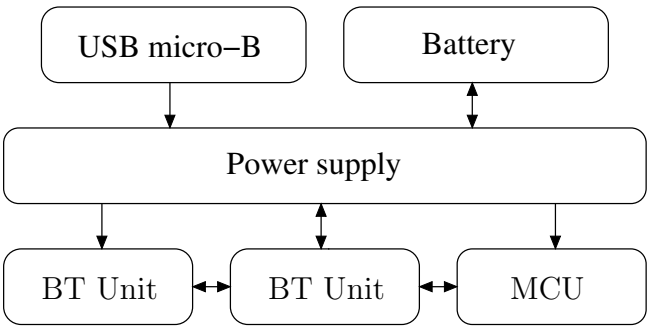

Figure 2: Hub block diagram

2 Design

The design of our project is split into two parts: a hub and speaker adapter.

2.1 Hub

The hub is responsible for streaming the wireless audio that it receives from the UE to the speaker adapters. It also receives RSSI values indicating adapter proximity to UE via the BT RFComm channel from the speaker adapters; the hub selects the adapter to stream to based on the RSSI values that it receives.

A high-level block diagram of the hub is shown in Figure 2.

2.1.1 Microcontroller Unit

The Microcontroller Unit (MCU) manages the operation of the hub and is responsible for network initialization. In order to do this, it coordinates the Bluetooth units at the hub. It communicates via Universal Asynchronous Receiver/Transmitter (UART) interface with the source BT IC to facilitate audio rebroadcast to speaker adapters. After receiving signal strength data from the Bluetooth unit (see discussion below), the MCU processes this info into selecting the speakers with the higher signal strength.

The microcontroller stores the last RSSI value received from each speaker adapter. Whenever it receives a new RSSI value, it replaces the previous value, and then determines which speaker adapter it should stream to. If it is different from the one it is already streaming to, it will disconnect from the current adapter and connect to the new stream target. The code to perform the disconnections and connections is included in Listing 2. Observe that the algorithm implemented in that code requires a few key contracts to be met in order to execute correctly. Most importantly, the link identifiers must match the contract listed in Table 2, or the hub will misbehave. To switch connected devices, the `switch_a2dp` function explicitly closes link ID 2 (which also closes link ID 3, because they're sharing the same A2DP stream), and then reopens an A2DP which reuses the same link IDs 2 and 3.

Listing 3 shows the `process_rx_decoded` function, which processes all incoming messages over UART. Because the WT32 is in multiplexing mode (mux mode), all messages are encoded with their source. Incoming control messages are identified by the link ID `0xFF` (which is an otherwise-invalid link ID) and we handle those messages first. We receive a series of control messages from the WT32 roughly every five seconds

Link ID	Required Connection
0	RFComm channel for first-connected device
1	RFComm channel for second-connected device
2	A2DP communication channel
3	A2DP data channel

Table 2: Hub A2DP link ID contracts

because we call the `LIST` command in a timer with that frequency. The `LIST` command returns the list of existing, current links; each result is returned in its own packet allowing us to process them independently. When we receive a response packet, we scan the link ID and check which device it's connected to and save that information. The current version of the algorithm waits for both devices to connect before opening the first A2DP connection.

The condition on line 5 of the listing ensures that we do not open an A2DP connection before both devices are connected, which ensures the contract in Table 2. Once both devices have RFComm channels open, we connect to the first one that sends a RSSI lower than the default (which will occur the first time we receive any RSSI of the UE). Then, every time we receive an RSSI value we determine whether we need to switch, and do so if necessary.

2.1.2 Bluetooth Unit

The Bluetooth Unit on the hub consists of two BT ICs, one acting as a sink and the other operating as the source. The sink BT IC is responsible for receiving audio data from the UE via the A2DP BT profile and then outputting this audio to the source BT IC via I^2S digital audio. The source BT IC is programmed to broadcast the audio packets received via I^2S data lines over the A2DP BT profile to one of the various speaker adapters around the room.

In our design we used the Bluegiga WT32 BT ICs, which expose a high-level firmware, iWRAP4, available over UART to configure Bluetooth parameters and access functionality typically hidden behind a much more complicated host-controller interface[1].

To simplify authentication, we always implemented Secure Simple Pairing (SSP) on every BT module resulting in a seamless connection through just a call. We can configure SSP through these settings:

```
1 SET BT AUTH * 0000
2 SET BT SSP 3 0
```

To configure the A2DP source, we use these settings,

```
1 SET CONTROL AUDIO I2S_SLAVE I2S_SLAVE
2 SET PROFILE A2DP SOURCE
3 SET CONTROL PREAMP 1 1
4 SET CONTROL GAIN 8 8
```

(In contrast, the sink IC is configured with audio routing `I2S I2S` and the `A2DP SINK` profile.)

2.1.3 Power Supply Unit

The primary power supply of each hub and speaker adapter is from micro-USB cable. The 5Vpower coming from the cable regulates down to 3.3Vusing a TI LM317L voltage regulator. Though such linear regulators fall short of other types of regulators in terms of efficiency, we ultimately decided its ease of use allowed us to focus on the central purpose of our system. To properly bias the output voltage to 3.3Vwe use the following equation to determine biasing resistor value,

$$V_O = V_{ref}(1 + \frac{R_2}{R_1})$$

where V_{ref} is the difference between output and adjustment voltages given by 1.25V, R_1 is the resistance between output and adjustment voltages (set at 470Ω per datasheet recommendation), and R_2 is the resistance between adjustment and ground[2].

$$3.3\text{V} = 1.25\text{V}(1 + \frac{R_2}{470\Omega})$$

Solving the above equation yields 770.8Ω. Using the resistors available to us, we connect 750Ω and 20Ω resistors in series to achieve proper output voltage on the regulator.

The charging circuit also provides a status indicator via LED that will blink on a low duty cycle. The LED has a forward bias current of 20 mA and a voltage drop between 1.8-2.2V. Below is the equation for the external series resistance value for the LED,

$$R_{LED} = \frac{VDD - V_f}{I_{LED}} - R_{ON}$$

where R_{ON} is the pad resistance, which is recommended to operate at a pad voltage no greater than 0.5V[3]. Making the safe assumption that pad voltage is negligible, the required resistance is

$$R_{LED} = \frac{5\text{V} - [1.8\text{V}, 2.2\text{V}]}{20\text{mA}} = [227\Omega, 278\Omega]$$

From these calculations we decided to use a 270Ω resistor in series with the LED.

One of the features of the integrated WT32 BT IC we used in our design is the constant-voltage and current charging circuits built into the chip, designed specifically for charging the lithium-ion battery we selected. The battery charging circuit operates in three states[3]:

Trickle Charge Mode: If the battery connected to the chip supplies a voltage below 2.9V.

Fast Charge Mode: If the battery connected to the chip supplies a voltage is above 2.9V.

OFF/Standby: In absence of battery its in an Off state and at full charge its in standby.

In our design scheme, the Li-Ion battery always provides a constant voltage of 3.6-3.8V so during normal operation the battery charging circuit is in Fast Charge Mode. Additionally, at full-charge the LED will cease to blink signifying that the battery is fully-charged and the charging circuit operates in standby mode.

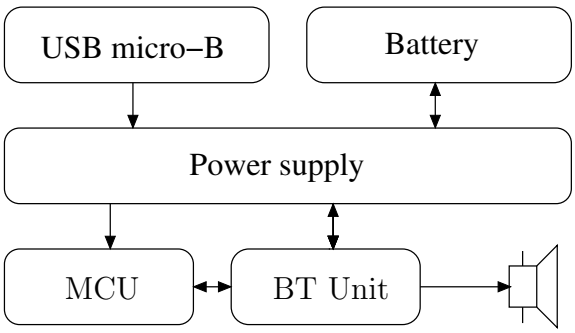

Figure 3: Speaker adapter block diagram

2.2 Speaker Adapter

The speaker adapter converts any standard off-the-shelf speakers with a 3.5mm audio jack into wireless BT speakers. It also records the RSSI values of the UE within its proximity and transmits this data back to the hub so that the hub can select the appropriate speaker to stream to. The high-level block diagram of the speaker adapter is shown in Figure 3.

2.2.1 Microcontroller Unit

The MCU on the Speaker Adapter coordinates the Bluetooth unit. It communicates via UART with the BT IC to facilitate receiving A2DP audio and outputting it as analog data to the speakers connected to the adapter. We use the internal timer in the MSP430 to interrupt the main execution loop and send the `INQUIRY` command to the WT32 every four seconds. The period of this inquiry is variable, but takes into account that sending commands too quickly to the WT32 could freeze the IWRAP firmware, and that a short `INQUIRY` timeout will find fewer results. The inquiry cannot be too slow either as this will lead to poor performance and lag between as the user moves between locations. Four seconds is reasonable and a clean division of the MSP430's 32kHz clock [4].

The interrupt handler routines were the trickiest part of our logic to write, because they need to execute extraordinarily quickly, especially when using a high baud rate. Without a sufficiently fast UART receive interrupt handler, it's very easy to drop bytes in the receive buffer. To help with this, we both used a clever buffering technique in the handler and decreased the baud rate to increase the timing headroom.

In our main RX interrupt handler, we track our position within a frame, so we can try to recognize a start-of-frame or end-of-frame. We were able to simplify the logic down to just these logical operations,

1. Verify that if we haven't read any bytes yet, the byte we're reading must be 0xbf. If not, it's invalid.
2. Don't do anything if we're currently processing an entire frame (otherwise, we might copy into the buffer that it is currently reading). Also, verify that we do not overflow our RX buffer.
3. Copy the received byte into our RX buffer at the next position.
4. If the current byte is a newline or we've reached the length indicated in the frame, conclude that we've completed reading this frame and make note of that so that the main loop can process the buffer.

The frame processor executed by the main loop looks only for control messages from the WT32. From those packets, it looks for packets that include the target equipment's BT address, and then decodes the RSSI value into an integer packed into one byte. The processor transmits the absolute value of the RSSI value back to the WT32 over UART to transmit over the RFComm channel back to the hub, where it's processed as described in Section 2.1.2.

2.2.2 Bluetooth Unit

The Bluetooth Unit on the speaker adapter consists of one BT IC responsible for playing audio packet data to the speakers. The BT IC is programmed to sniff RSSI data from the UE within the respective speaker adapter's proximity and then transmit that data back to the hub. The interrupting RSSI inquiry is handlesd by the MCU. The unit is capable of simultaneously handling both processes.

The firmware provides the `INQUIRY` command which gives a list of BT device addresses and their respective RSSI values for devices near the adapter. The RSSI ranges from -128 (poor) to 20 (good). Using the RSSI allows us to pick the speaker closest to the UE and play from the correct speakers. The Bluetooth unit also provides the Inquiry function, which is used to find other Bluetooth devices in the area [1]. Thus, the Source BT IC receives RSSI data from the various speaker adapters via RFCOMM channel and compares the RSSI values sent by the respective speaker units. Subsequently, the hub chooses the speaker with the highest RSSI value visible to the UE for audio streaming.

2.2.3 Power Supply Unit

The power supply unit on each of the speaker adapter is designed exactly like the one on the hub. A detailed description of the design can be found in section 2.1.3.

3 Requirements & Verification

The first requirement was to have enough programmable microcontrollers to insert into each of our modules. The PCB designed for the MSP430 includes header pins for connectivity to the MSP-FET430UIF USB Debugging Interface. We chose to forgo using such development tools as Launchpads and Arduinos as we wanted to develop as much of a final product as we could.

After countless hours of debugging, unsoldering the various short circuits on the board and solder pads, we were left with four programmable MSP430 boards including a spare board for any emergency situation. These chips successfully load our code as confirmed by the Code Composer Studio software and connect to the UART TX and RX pins, common VDD, and common ground on perf board.

We then verified that the MSP430 could interface with the WT32 in this configuration by programming test code (as well as our real code eventually) to configure/control the Bluetooth chip on both the hub and the speaker adapters. Specific to each block, the hub must accept the UE's Bluetooth pairing while the speaker adapter's Bluetooth chip must silently gather signal strength information while not disturbing the music playback. These were accomplished with proper configuration and programming to handle the tasks. Example code is provided in Section F.

Volume bits	Speaker volume level	Decibels (dBA)
00	50%	52.0
10	50%	60.2
20	50%	70.1
30	50%	79.2
3F	50%	82.4
10	75%	71.1
20	75%	82.3
30	75%	88.0
3F	75%	90.0
10	100%	75.8
3F	100%	91.0

Table 3: Audio amplifier performance

On the speaker adapter end the audio frontend must also provide proper playback out of the analog audio jack into the speaker itself. Simply hearing audio is not a sufficient enough test, as we previously quantified our sound measurements and measured the performance of the now-defunct amplifier using the decibel meter 18″ away from the the speaker. We adjusted the volume two ways. First, by sending the I2C commands to reset the register 0x02. The audio amplifier volume ranges from 00 for mute up to 3F. Second, by physical control of the volume knob on the speakers. The results we captured are shown in Table 3. (Note that the first row indicates the ambient noise floor.) We also observed that the un-amplified audio was approximately 3dB down at each level from what is shown below, but nonetheless provided clearly audible music.

Afterwards we verified we had left and right channel separation by playing mono streams of each side, as one ought to expect out of a Bluetooth Stereo Network.

The verification of each component's design requirements revealed much of our project's verification truly sat at the system level discussed in the next section. Though getting each block working individually was no trivial task, we face countless challenges integrating the components from a software standpoint. We realized that setting the baud-rate of the BT IC to standard rates like 9600, 115200 is recommended to ensure proper system functionality. Failure to do so might disrupt the settings within the chip making it unresponsive and thereby requiring a special hard reset over SPI interface with special equipment supplied by the manufacturer[1]. At the system level, cooperation between each block becomes much more intricate.

3.1 Other Testing

In addition to our official requirements and verifications, we performed extra module and system level testing to confirm our designs and progress. We developed UART interface test programs in C to speak at binary packet levels. Simply put, we wrote a program to translate the ASCII of the IWRAP firmware into hex. This was necessary to employ a mode in IWRAP called Multiplexing Mode that allows use of all IWRAP commands regardless of Bluetooth profile. This was vital to our project as we used multiple profiles such as Serial Port and Advanced Audio Distribution. Multiplexing mode transmits data and commands in the same mode. According to the IWRAP firmware description, switching from data to command modes and vice versa could take more than two seconds in the worst case [1]. For a speaker adapter that needs to receive wireless audio and transmit RSSI values to the hub, the switching between command and data modes would

have resulted in not being able to properly hear music. Our program reduced the complexity of programming in hex by translating the typical ASCII IWRAP commands into the hex commands required in multiplexing mode. We verified IWRAP command framing code worked as we programmed our entire project this way.

We accomplished most of our debugging via reading the UART TX and RX lines of either the WT32 or MSP430 (since they are connected together) on the hub with a Bus Pirate. The Bus Pirate connects to a computer via USB and allows communication in a variety of electronic protocols via a terminal. Through the Bus Pirate, we could see the incoming RSSI data from the two speaker addresses on the hub. Below is a snippet of a stream of TX data coming out of the WT32 (RX into the MSP430).

```
1 \0xbf\0x01\0x00\0x01\0x44\0xfe
2 \0xbf\0x00\0x00\0x01\0x01\0xff
3 \0xbf\0x00\0x00\0x01\0x42\0xff
4 \0xbf\0x00\0x00\0x01\0x01\0xff
5 \0xbf\0x01\0x00\0x01\0x01\0xfe
```

Each line represents a frame in the multiplexing protocol. The protocol is as follows:

1. The first byte, 0xbf, always represents the start of the frame
2. The second byte is either 0x00 or 0x01, which represent the link IDs of speaker adapter A and speaker adapter B, respectively.
3. The next six bits hold the frame flags, and should always be 0x00
4. The next 10 bits hold the data size in bytes, which in our example is one.
5. Next comes our data. In our case, this is the RSSI data that we are supposed to see from each speaker adapter. The 0x01 is sent back during the inquiry period if the speaker adapter's `INQUIRY` picks up a device, but not necessarily the paired device of interest. We still send the 0x01 back to the hub as a sign to the hub the speaker adapter is still inquiring and transmitting.
6. The final byte displayed is the nLINK which is the Link ID XOR'd with 0xFF. Thus we see 0xfe on link 0x01 and 0xff on link 0x00[1].

There were also "common-sense" tests we conducted to ensure proper functionality. We verified our Bluetooth chips could pair with various user equipment such as phones, computers, and MP3 players. Another sanity check was to update the naming of the Bluetooth chips with each programming write as a way to verify our code was getting updated. For example, "SPEAKERADAPTERA1" would show up discoverable as "SPEAKERADAPTERA2" after a program revision. At the system level, we verified the design met our goals by walking back and forth between the speakers and successfully observing proper audio playback. We also performed a rough link-budget analysis of our system and verified that the RSSI values obtained at the hub were accurate.

4 Costs

Since our device is the only device that provides proximity based playback, and is compatible with any Bluetooth audio device without the need for any special app (iOS, Android, etc.), we expect there to be a significant market for this system. A notable wireless streaming system by Sonos, does allow streaming to specific speakers using a mobile app to control playback, but speakers for the system go well beyond the $300 range. Additionally, if the user lacks an Android or "iDevice," the user must buy a proprietary remote for $349, putting the Sonos system out of range for most consumers.

Current Bluetooth speaker adapters range from $20 to $40 yet do not offer the functionality ours do. Selling our base 2-speaker system for $200, which we believe will make our system more than competitive against the current crop of speaker adapters as well as popular portable Bluetooth speakers, such as the Jawbone Jambox or Beats Pill, which cost upwards of $200 as well. A rough discussion of costs is included below, as well as in Section B.

4.1 Parts

Each of the WT32 modules cost $30. When ordered for mass production, the cost drops to $24. When considering all such parts in mass quantities, including 2-layer PCB with bottom side silk-screen, we estimate the cost to be around $50 per unit, bringing our parts cost down to $150 from $161.09. A detailed listing is provided in Table 7.

4.2 Labor

These costs are built on the assumption each of us worked 10 hours a week on the project though in reality our time worked on project skyrocketed in the final week or two. However sticking with our estimated 150 hours worked at a rate of $75 per hour, our total non-parts cost is $84,375. A detailed listing is provided in Table 6.

5 Conclusions

Through our work over the course of the semester, we have successfully demonstrated a Bluetooth stereo network that plays to the speakers closest to the user. By walking across a room from one set of speakers to another, we observed audio switch automatically to the speaker beside the user. Thus, our project provides the solution to streaming music to different rooms without automatic audio packet switching between speakers.

5.1 Future Work

Our project was successful, but there are certainly improvements and modifications that we can work on in the future. The most obvious undertaking is the audio quality. By using analog audio in, we experienced much degradation in our sound coming from the UE. This is something that can be avoided by implementing an all-digital receiver frontend, and using the I^2S audio we demonstrated in out mockup, or by improving our wiring on all the analog inputs/outputs.

This digital frontend is not without tradeoffs, however, as we found our final design to be newly compatible with any audio device, not just Bluetooth enabled-ones. Should we decide universal compatibility is important enough proceed with our current design, improving audio quality is a matter of improved soldering and wiring on the analog inputs and outputs.

We hope to in the future add support for more than two speakers as well. For cost and time purposes, demonstrating our system with more than two speakers would not be feasible. This is but a matter of assembling more speaker adapters and programming the hub to be able to handle more of them (or revisiting PurePath as our method of handling wireless audio transmission). Software is always open to revision to make our network operate faster and more robustly.

Finally, to make this into a product that is mature enough for the consumer market, we must add proper device housing, PCB with silkscreen, range-boosting antenna, and a physical power switch onto all the units. These are mostly cosmetic changes as well as some performance-boosting modifications we can add now that we have a working system.

5.2 Ethics

Throughout building these devices, we have been very conscious of the IEEE Code of Ethics in Section 7.8 of IEEE Policies[5]. In particular, these policies are particularly important to remember or particularly relevant to our project:

3. *"[To] be honest and realistic in stating claims or estimates based on available data."*

   It is extremely critical to be honest and accurate in the claims of performance on products that feature either batteries or radios, and our project contains both. Therefore, we have sought to underestimate the battery performance and to claim RF performance that does not exceed the minimum acceptable performance for the radio components we are using.

6. *"[To] maintain and improve our technical competence and to undertake technological tasks for others only if qualified by training or experience, or after full disclosure of pertinent limitations."*

   This requirement has been important when representing ourselves in this course, including to partners, the TAs, and the professors. We have been careful to emphasize to our peers our level of certainty in design decisions as well as answer any questions posed towards our design.

9. *"[To] avoid injuring others, their property, reputation, or employment by false or malicious action."*

   Our project's express purpose is for location-based music playback. We will not use any malicious hacks on Bluetooth such as bluejacking and bluebugging, which can exploit Bluetooth devices and the user's identity.

In addition to the IEEE Code of Ethics, we must also satisfy FCC Part 15 requirements, which govern acceptable use of radio frequency devices. Under these regulations, our Bluetooth stereo network falls under Class B digital devices, which are marketed to the general public for home use. While part 15 is lengthy, we will make sure to adhere to sections 15.247 and 15.249, which govern the 2.4 to 2.483 GHz band we are using.

Appendices

A Requirements & Verification

A.1 Hub

Table 4: Hub verification procedures

Block	Requirement	Verification
Microcontroller Unit (TI MSP430G2553)	1. Compiles code from Code Composer and accepts the program loaded onto it. 2. The UART connection between the MSP chip and the front-end BlueGiga BT chip is established and a subsequent UART interface is observed. 3. The VDD on the MSP should be configured to 1.8–3.3V.	1. The code is successfully ported onto the MSP chip without compilation errors. 2. Use the Bus Pirate to monitor the two-way UART communication established between the MSP and BT chip via a Serial terminal. 3. Measure 1.8–3.3V across the VDD pins 29, 30 and GND across pins 27, 28.
Bluetooth Unit (Bluegiga WT32)	1. Device must be able to interface with the MSP over UART. 2. We should be able to accept audio data from UE for further broadcast to the speaker adapters.	1. Develop test program that indicates, via UART, for the unit to be visible with a given name. Use a BT device (any phone) to pair with the Bluetooth unit. 2. Develop test program that indicates, via UART, for the unit to present the A2DP BT profile. Then, connect to the unit through UE (any phone) and stream audio. Watch the analog data stream with an oscilloscope to verify continuous audio playback.

Continued on next page

Table 4 – *Continued from previous page*

Block	**Requirement**	**Verification**
Power Supply Unit	1. Accept 5 ± 0.5V line through USB mini receptacle and provides 3.3 ± 0.5V rails. 2. Power charging circuit is able to charge a Li-Ion battery connected to it.	1. Provide 5.0 ± 0.5V input with bench power supply to USB mini receptacle via USB mini cable. Verify that at all valid input voltages yield valid rail voltages (3.0–3.8V) at the output of the regulator. 2. Observe the blinking LED on power circuit to ensure proper charging of battery.

A.2 Speaker Adapter

Table 5: Speaker adapter verification procedures

Block	**Requirement**	**Verification**
Microcontroller Unit	1. Must request signal strength data from Bluetooth unit over UART and publish that data in the form of datagram packets back to the Hub.	1. Power the Bluetooth unit and connect the UART lines to the microcontroller. 2. Program MCU to request list of visible Bluetooth devices via UART. Observe UART data on CuteCom using Bus-Pirate and record RSSI data of UEs within the vicinity.

Continued on next page

Table 5 – *Continued from previous page*

Block	**Requirement**	**Verification**
Bluetooth Unit	1. Silently listens to Bluetooth signals but never actively broadcasts. 2. Able to list available BT devices within a 20ft radius and record the RSSI values of nearby devices that are set to Bluetooth discoverable mode.	1. Be able to connect to the Speaker Adapter module via Bluetooth from UE and transmit continuous audio data stream. Observe audio playback onto the attached speakers. 2. Interface with the Hub in that the RSSI values of nearby devices in discoverable mode should be transmitted to the MCU on hub side. Accept commands to play or not play audio onto a specific UE trying to connect to the respective speaker adapter.
Power Supply Unit	1. Accepts 5.0 ± 0.5V line through USB mini receptacle and provides 3.3 ± 0.3V rail.	1. Provide 5.0 ± 0.5V input with bench power supply to USB micro-B receptacle via cut USB cable. Verify that at all valid input voltages yield valid rail voltages (3.0–3.8V) at the output of the regulator.
Audio Frontend	1. Attached off-the shelf wireline speakers should be converted to wireless speakers and play audio data packets transmitted from phone.	1. Be able to hear the audio data transmitted from UE.

B Cost Analysis

Cost analysis of labor costs is shown in Table 6. Analysis of parts costs is shown in Table 7.

Name	Hourly Rate	Total Hours Invested	Total
Jeff Wheeler	$75.00	150	$28,125
Rishi Ratan	$75.00	150	$28,125
Jerry Sun	$75.00	150	$28,125
Total	–	**450**	**$84,375**

Table 6: Labor costs

Part	Count	Unit Cost ($)	Total Cost ($)
MSP430G2553	3	2.73	8.19
WT32	4	29.67	118.68
Capacitors	15	0.05	0.75
Resistors	16	0.05	0.80
LEDs	3	0.05	0.15
Connectors	6	0.50	3.00
PCBs	3	9.84	29.52
Total	–	–	**$161.09**

Table 7: Part costs

C Schematics and PCB Layouts

C.1 MCU Schematic

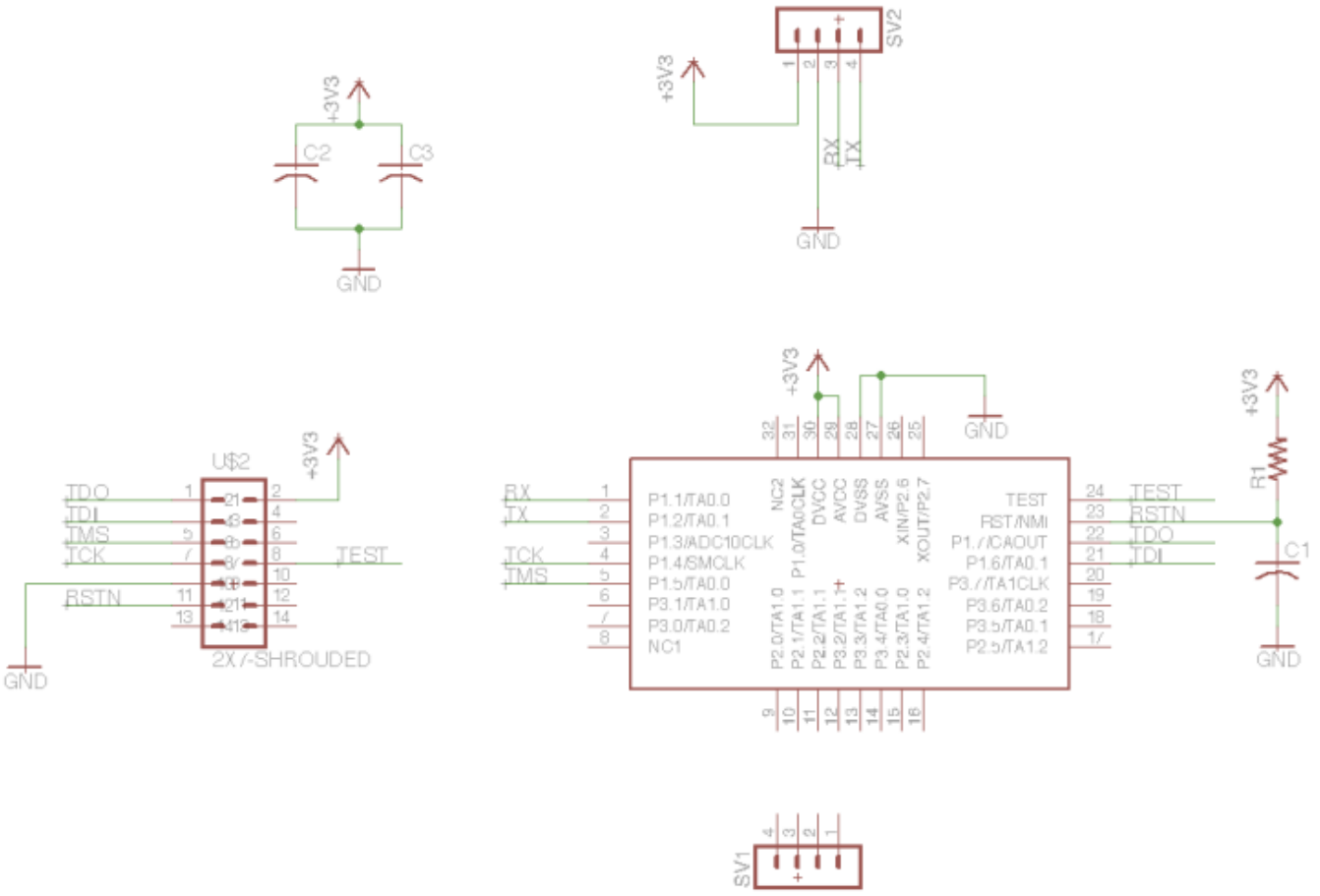

Figure 4: MCU Schematic

C.2 MCU Layout

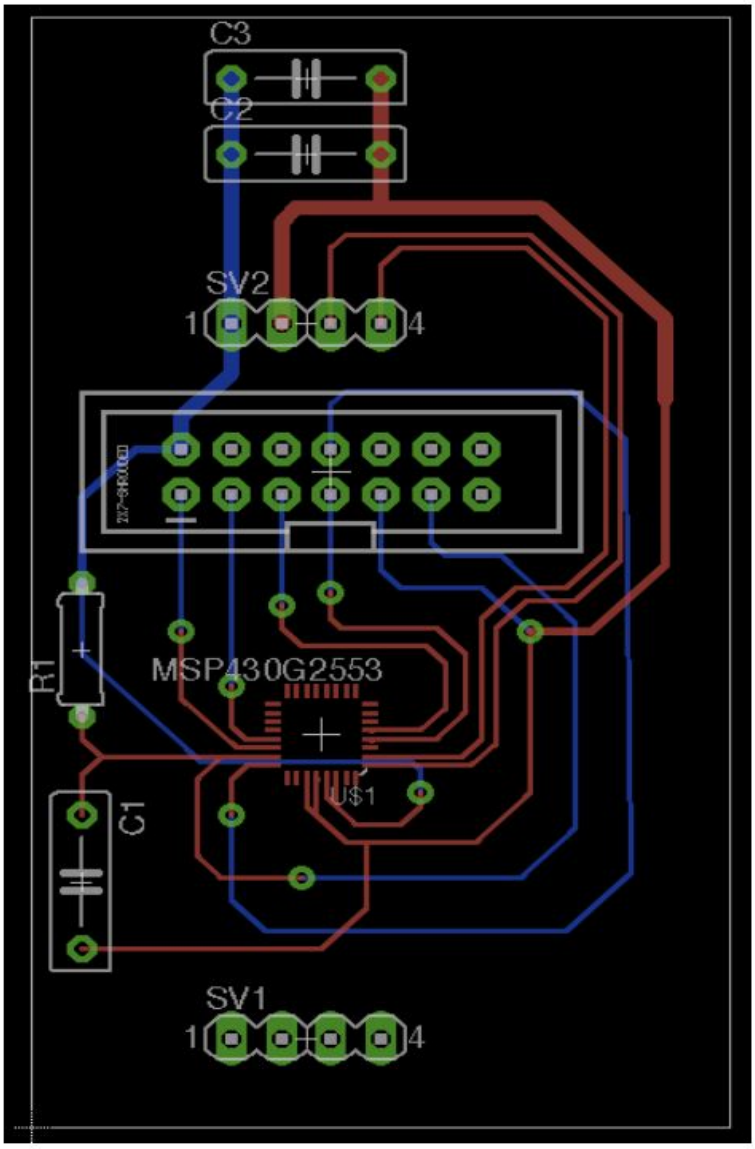

Figure 5: MCU Layout

C.3 BT Unit Schematic

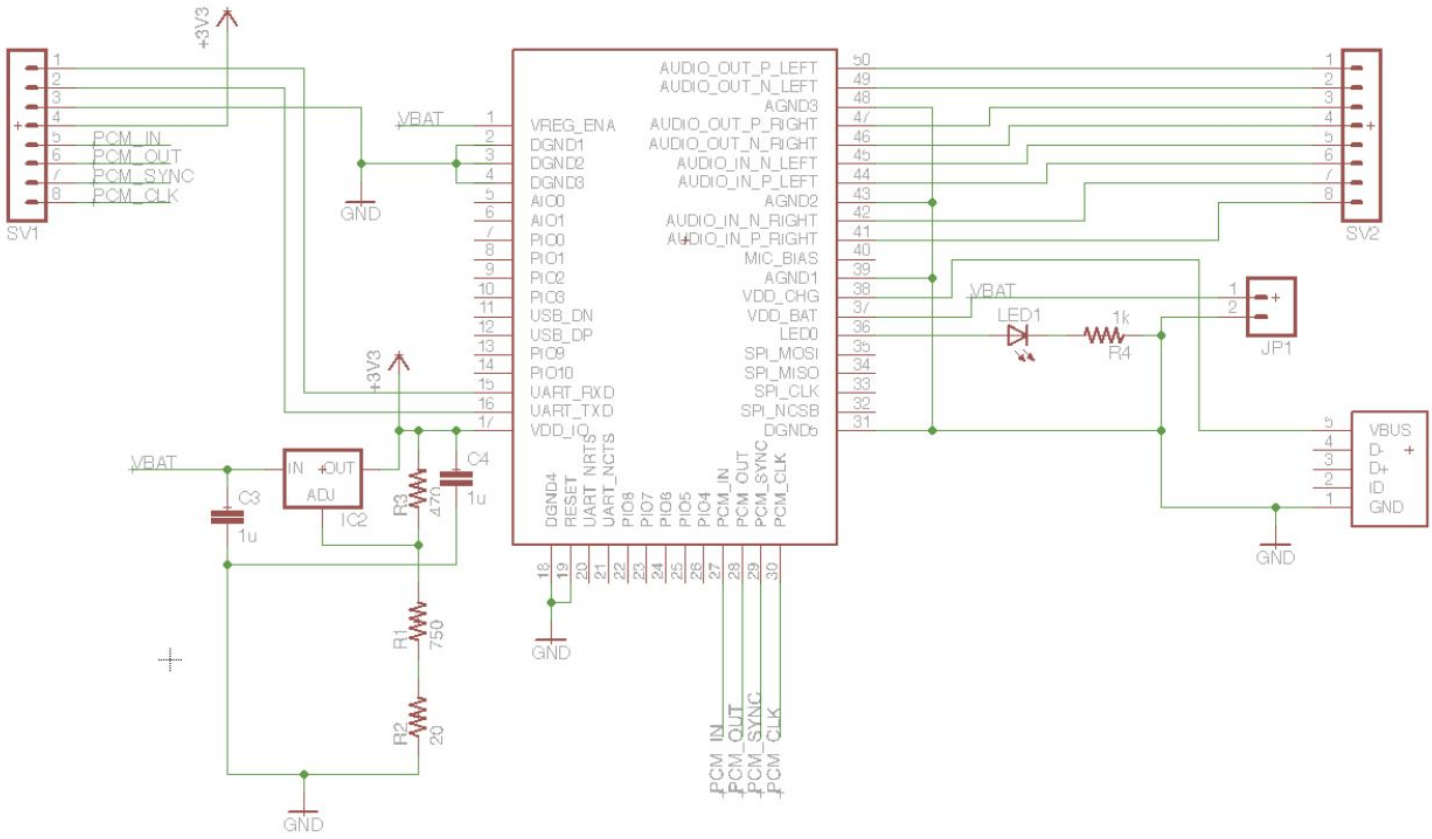

Figure 6: BT Unit Schematic

C.4 BT Unit Layout

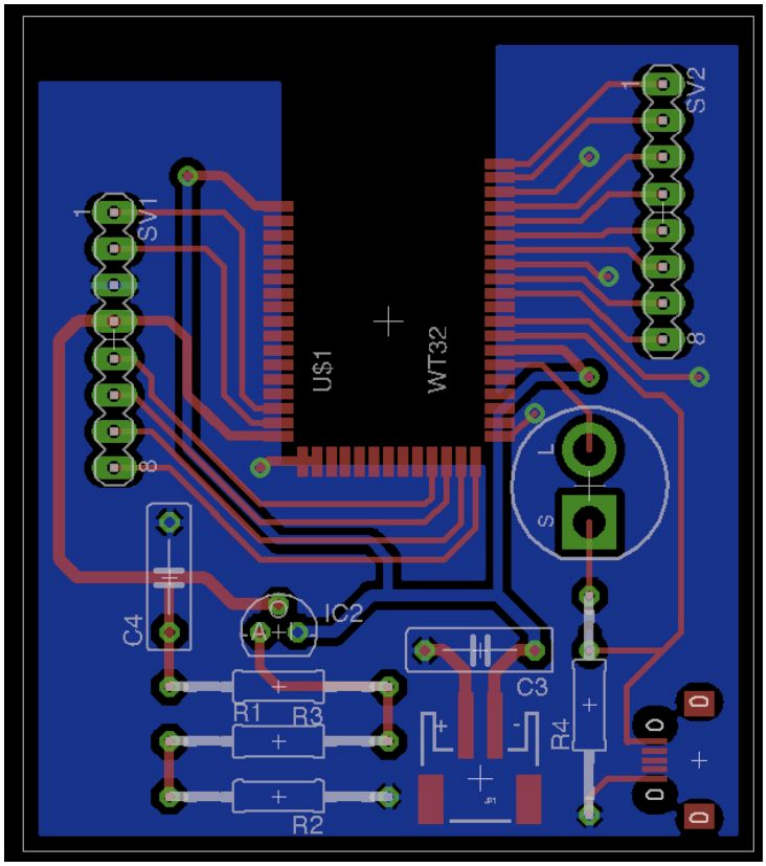

Figure 7: BT Unit Layout

D Industrial Device Designs

D.1 Hub Mechnaical Construction

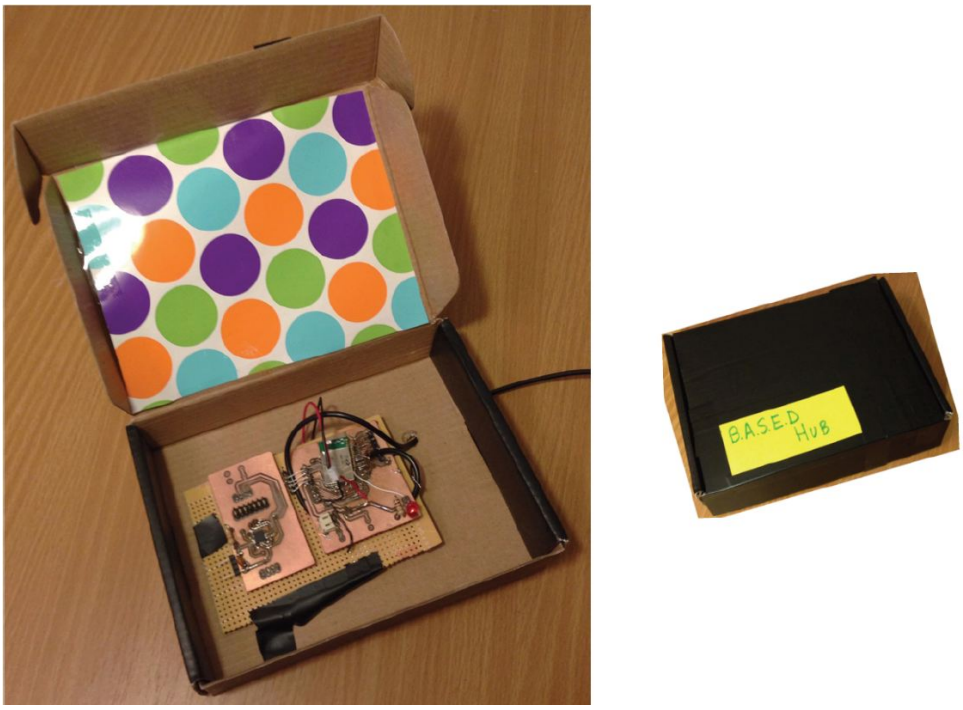

Figure 8: Hub Packaging

D.2 Speaker Adapter Mechanical Construction

Figure 9: Speaker Adapter Packaging

E Chip Resurrection

During the final stages of our design we accidently destroyed the pads on one of our BT ICs for the Hub. However, with patience and careful soldering we were able to resurrect the chip and put it back in use. The following figure showcases this instance in our design cycle.

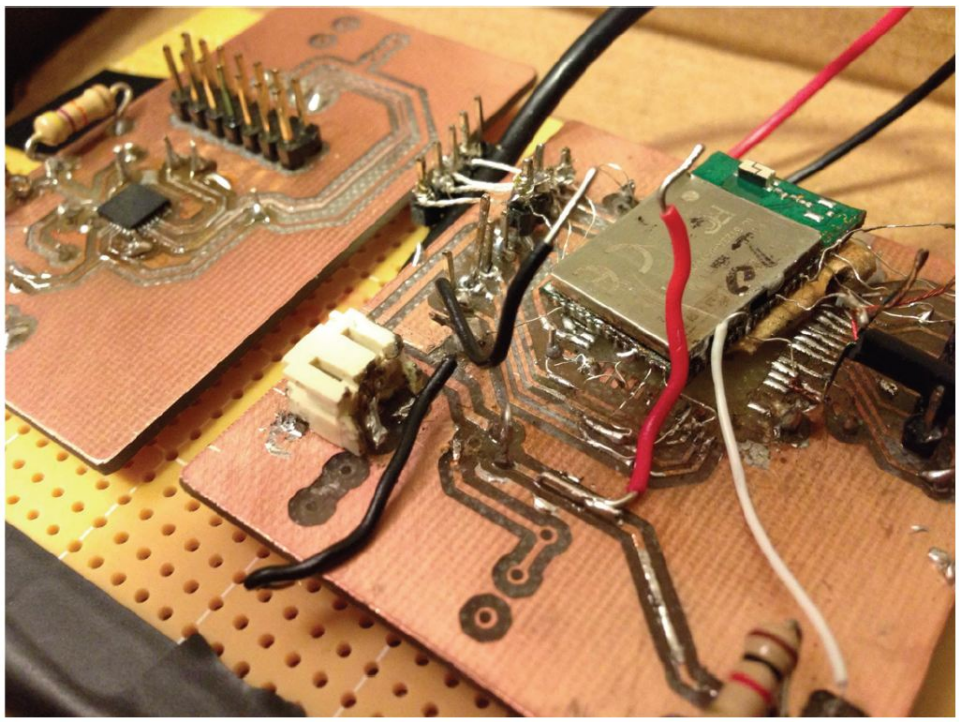

Figure 10: BT IC Resurrection

F Microcontroller control code

There was significant effort in cleanly handling the multiplexing mode (mux mode) of the WT32. The functions below are used to transmit data over UART both in control mode and in mux mode:

Listing 1: UART Mux Mode Handlers

```
void txdata(char c) {
    while (!(IFG2 & UCA0TXIFG));
    UCA0TXBUF = c;
}

void tx_send_bytes(const char *c) {
    while (*c) {
        txdata(*c);
        c++;
    }
}

void tx_string(const char *c) {
    tx_send_bytes(c);
    txdata('\n');
}

void tx_frame(char link_id, const char *c) {
    txdata(0xBF);
    txdata(link_id);
    txdata(0);
    txdata((char)strlen(c));
    tx_send_bytes(c);
    txdata(link_id ^ 0xFF);
}

void tx_control(const char *c) {
    tx_frame(0xFF, c);
}
```

The algorithm to select the adapter to stream to was described in Section 2.1.1. First, the code to switch streaming speaker adapters is shown in Listing 2, and then the code to select which speaker to stream to is shown in Listing 3.

Listing 2: Method to switch speaker adapters

```
void switch_a2dp(char target_link) {
    if (last_connected >= 0) {
        char x[10] = "CLOSE 2";
        tx_control(x);
        __delay_cycles(0.4*ONE_SECOND);
    }
    txdata(target_link + '0');
    char *x;
    switch (last_addr[target_link]) {
        case 0:
            x = "CALL 00:07:80:5f:a3:43 19 A2DP";
            break;
        case 1:
            x = "CALL 00:07:80:53:28:d4 19 A2DP";
            break;
    }

    txdata('\n');
    txdata('\n');
    tx_control(x);
    last_connected = target_link;
}
```

Listing 3: Logic for selecting speaker adapter

```
void process_rx_decoded(char link_id, char length, char *buf) {
    if (link_id == 0xFF) {
        // Maintain list of connected devices
    } else if (link_id == 0 || link_id == 1) {
        if (both_connected) {
            last_rssi[link_id] = buf[0];
            if (link_id == 0 && last_rssi[0] < last_rssi[1] &&
                    last_connected != 0) {
                switch_a2dp(0);
            } else if (link_id == 1 && last_rssi[1] < last_rssi[0] &&
                    last_conected != 1) {
                switch_a2dp(1);
            }
        }
    }
}
```

G References

[1] "iwrap4 user guide," Bluegiga Technologies, March 2011.

[2] *3-Terminal Adjustable Regulator (LM317L datasheet)*, Texas Instruments, October 2011.

[3] *WT32 datasheet*, Bluegiga Technologies, September 2008.

[4] *MSP430 Hardware Tools User's Guide*, Texas Instruments, June 2012.

[5] *7.8 Code of Ethics*, IEEE.