Portable Hi-Res Digital Audio Player

Rammi Hameed
Jake Flores
Seth Pregler
Patrick Westmoreland

CONCEPT OF OPERATIONS

REVISION 2.0
23 April 2023

CONCEPT OF OPERATIONS
FOR
Portable Hi-Res Digital Audio Player

TEAM 25

APPROVED BY:

Seth Pregler
_____
Project Leader Date

_____
Porf. Wonhyeok Jang Date

_____
T/A Date

Change Record

Rev.	Date	Originator	Approvals	Description
1.0	02.07.2023	Rammi Hameed Jake Flores Seth Pregler		Draft Release
2.0	04.23.2023	Seth Pregler Jake Flores		Changes made based on grader feedback.
3.0	12.02.2023	Seth Pregler		Slight modifications to power. Addition of Patrick

Table of Contents

List of Tables

List of Figures

Glossary

DAP:	Digital Audio Player
MCU:	Microcontroller Unit
FLAC:	Free Lossless Audio Codec
MP3:	MPEG Audio Layer 3
WAV:	Waveform Audio File Format
DAC:	Digital to Analog Converter
GUI:	Graphical User Interface
I2C:	Inter-Integrated Circuit
I2S:	Inter-Integrated Circuit Sound
OS:	Operating System
RTOS:	Real-Time Operating System
LCD:	Liquid Crystal Display
USB:	Universal Serial Bus
PCB:	Printed Circuit Board

1. Executive Summary

This concept of operations aims to serve as a roadmap for the development and implementation of a high-resolution Digital Audio Player (DAP). We have found that there is a growing interest among individuals who are passionate about high fidelity audio equipment. Our chief purpose is to develop a best-in-class music experience for these individuals. This document first identifies system deficiencies among current DAPs commercially available and develops a design criterion for the development of our proposed device. To make this document self-contained, a brief technical background is provided for the reader in, *2.1.3 Technical Background,* before providing a system overview. Section *3, Operating Concept,* explores a proposed design and details how the system shall operate. A timeline for development and deployment of our project is presented in the final section.

2. Introduction

The consumer electronics market has undergone significant developments in the recent past. Notably, in past decades, smartphones have become the dominant form factor, being able to serve a myriad of applications, from cameras and music players to navigation systems and gaming; smartphones exemplify the qualities of usefulness and versatility.

But for people willing to pay a premium for the best digital music experience available, the sacrifices made to make smartphones versatile and attractive to the average consumer leave much to be desired. To this end, dedicated high resolution Digital Audio Players (DAPs) can provide best-in-class sound quality, a more enjoyable user experience and improved power efficiency when compared to smartphones.

2.1. Background

2.1.1. Product Evaluation

Most DAPs on the market today are designed with the average consumer in mind, resulting in compromises when it comes to audio quality and user experience. As part of our research, our team conducted an in-depth product evaluation of popular DAPs on the market and discovered key system deficiencies. Table 1-1 summarizes the most significant system deficiencies common among DAPs. Subsequent sections describe the system deficiencies in greater detail and propose solutions for improvement.

Table 1: DAP System Deficiencies Summary

Attribute	Deficiencies
Sound Quality	• Leading smartphones lack dedicated audio components that result in inferior audio playback • Several smartphone manufactures have opted to remove headphone jacks resulting in worse sound quality
Power	• Power supply not optimized for audio applications
User Experience	• Limited equalization functionality • Lacking balanced playback • Lacking Graphical User Interfaces (GUIs)
Price	• Many manufacturers of DAPs retail devices at unreasonable prices

2.1.2. Design Criterion

To address current system deficiencies our team has developed a list of priorities which will be used to drive engineering decisions in the development of our proposed DAP, as well as to be used as a verification criterion used in later phases of design, namely:

- High fidelity audio
- Optimized power system
- Streamlined user experience
- Equalization adjustability

These key values were used to determine the best engineering approach. More information shall be presented section *2.2 Overview.*

2.1.3. Technical Background

To understand our proposed DAP, a rudimentary understanding of audio processing is required. When a source, such as a musical instrument, vibrates, producing sound, it causes an acoustic wave to propagate the surrounding medium. This propagating wave is a time varying quantity and therefore can be considered as an analog signal. This signal is continuous in time and may take on infinitely many values.

The goal of producing high fidelity digital audio can be, in part, reduced to how we represent an analog signal digitally in a way such that there is no perceivable distortion between the digitized and analog versions. This process of representing an analog quantity digitally is called quantization and is the first design decision of practical importance we must be concerned with. Naturally, the more bits allocated to quantization, the higher quality the digitized signal, thus a DAP must be designed such that it handles data with a high enough bit width to ensure minimal loss of information.

The next design decision we must be concerned with is how we choose to encode such data. Like bit width, the choice of audio encoding has a direct influence on sound quality, as well as storage space consumption and resource allocation. There are many such ways to go about encoding, however, they all fall under two types, lossy and lossless. Lossy encoding, or compression, is a type of encoding that reduces the size of a file by permanently removing some of its information. Lossless compression, on the other hand, reduce file size without permanently removing any of the original signal information. This means that loss of audio quality is inherent in lossy encoding schemes. While lossless compression proves superior in terms of fidelity, lossless compression suffers in that file sizes are more extensive. Common file standards include WAV, MP3, and FLAC, just to name a few. In the following sections, we will evaluate the strengths and weaknesses of these formats.

2.2. Overview

2.2.1. System Architecture

In keeping with our design priorities presented in section *2.1.2 Requirements Analysis,* we have focused available resources in creating a streamlined user interface that allows the user to configure audio settings for an adjustable end user experience. This includes both an optimized content management application, empowering the user with the ability to select built in presets or custom tune the audio to their liking, as well as the hardware necessary to provide such functionality, including on-board equalizers and tactile volume control. Moreover, input to the device will be predominantly through a touchscreen interface.

Our proposed architecture can be organized into five main subsystems, namely, the power supplies, MCU, control interface, audio codec, and audio ‘front-end’. The system architecture is presented in Figure 1.

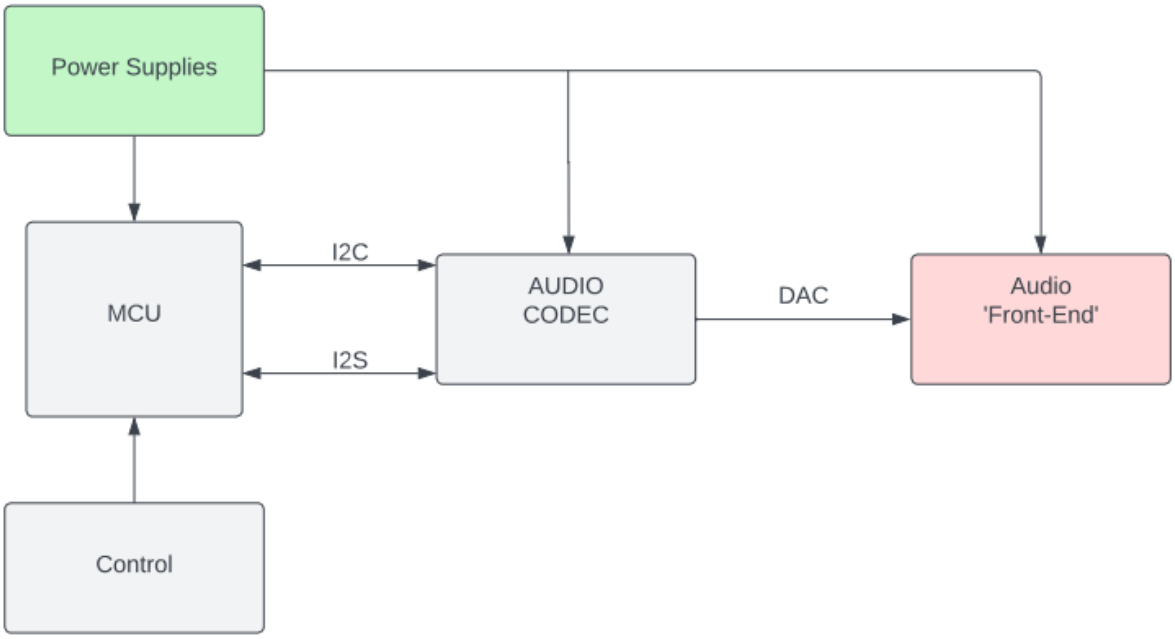


Figure 1. Digital Audio Player Block Diagram

2.2.2. Additional Functionality

The device will feature a Universal Serial Bus (USB) for power delivery and music download. Additionally, both balanced and unbalanced headphone jacks will be available to accommodate a wide variety of headphones.

2.3. Referenced Documents and Standards

[1] Marwedel, Peter. *Embedded System Design Embedded Systems Foundations of Cyber Physical Systems*. Springer, 2011.
[2] I2S bus specification – Phillips Semiconductors, 1986
[3] AN10216-01 I2C Manual – Phillips Semiconductors, 2003

3. Operating Concept

3.1. Scope

The DAP will provide an affordable way for users to listen to great quality music. With an optimized audio path, cutting edge Microcontroller, DACs, amplifiers, and storage, the user gets a lot of bang for their buck. We oriented our design towards the average person, focusing on ease of use, durability, longevity, and accessibility. Downloading audio into the on-board storage is very intuitive with Bluetooth or USB, and with support for balanced headphones, there will be negligible loss of audio quality.

3.2. Operational Description and Constraints

The DAP is going to be a portable device that stores and plays audio through balanced headphones. The resolution of the audio is going to exceed that of CD quality, the DAP will also give the user the chance to manipulate their audio files, by creating playlists, shuffling, accessing albums, etc. The user interface will be intuitive with an app controlling the whole thing. The DAP will have 64 GB of storage through an SD card, allowing for more than 40,000 minutes of audio.

Some of the biggest constrains in the system are the following:

- It must be a small package. The DAP must fit into a person's pocket, requiring the electronics inside to be on a small PCB.
- The audio must be at least that of CD quality, which means we need at least 16 bits and 44.1kHz refresh rate.
- The MCU and digital equalizer must be able to communicate, this is also true for the DAC and the analog path, but that doesn't seem to be an issue for this DAP.
- Since it is a portable device, the battery must be able to operate the device for extended periods of time without dying.
- The MCU, CODEC, and Front-end Audio all have an operating voltage of 3.3V, so the 5V battery voltage must be able to step down to 3.3V.
- The Operating System (OS) must have very little overhead which means little memory footprint.
- The processor used must be powerful enough to run the OS, content management application, and provide control to the audio codec.

3.3. System Description

3.3.1. Audio Processing

When the user selects a song to play using our GUI, the MCU retrieves the selected song from internal storage as a compressed bitstream. This bitstream is then transmitted serially using the Inter-IC Sound (I2S) bus to the audio codec where it is processed digitally. Control signals from the processor to the audio codec will be sent using the Inter-Integrated Circuit (I2C) communication protocol. The audio codec will then use a decompression algorithm to restore the original digital audio samples using the MP3 standard. Additionally, the processor will offload digital signal processing work to the codec using I2C.

3.3.2. Content Management and User Control

Our content management application will provide the user with a visual interface to play, pause, stop, skip and adjust volume of audio tracks. Additionally, a file management system will run on top of a Real-Time Operating System (RTOS). The RTOS has a small memory footprint which makes it ideal for our application, while providing enough abstraction.

3.3.3. Power System

The system will be powered by a 3.7V Li-Ion/Li-polymer battery. Since the MCU, Codec, and Front-end audio can only withstand an input voltage of 3.3V, a buck-boost converter will be

used to power all the 3.3V digital components while a voltage regulator will be used to power all the 3.3V analog components. For the device to be portable, a chip will be used that has features such as a micro-USB battery charging connection, LEDs to tell when the battery is charging/finished charging, and through hole connections to connect the battery and charging chip to the power subsystem to make a fluid and cohesive power subsystem. For the mechanical side of the project, an on/off switch will be placed on the display to make the device power on and off. Also, a volume gauge will be portrayed on the display to control the volume levels coming out of the device.

3.3.4. Device Enclosure

The entire system will be encased in a premium enclosure. The final dimensions of the device are: 4-inch x 6.5-inch x 1.5inch.

3.3.5. Audio Path

The user's audio files start as digital data in an SD card which is placed on a chip that allows the transmission of the info on the SD card into the microcontroller, which then will allow the user to tune the sound through a built-in mixer that adjusts the different frequencies of the signal before it gets to the DACs. The design will support balanced headphones, which need a right and left channel outputs, therefore the digital audio signal will be split into a left ear and right ear path, each of those bitstreams will go into a Digital to Analog Converter, then each of the analog signals will go through an amplifier and finally out of the respective headphone jack. The DAP will use MP3 files in its' operation. MP3 files are very compact, yet they maintain an impressive amount of sound quality. The audio in the WAV format is uncompressed, therefore it has the best quality, but it takes up about 10 MB of space per minute of audio, MP3 takes up about 1 MB of space per minute in comparison. MP3 is compressed by removing frequencies of sound that humans can't hear, which is why it maintains so much of the sound quality. At 10 times more music in the same SD card, the small sacrifice in quality is more than worth it. MP3 is also the most widely used form of audio files, making it very easy for the user to find music to upload into their DAP through micro-USB or Bluetooth.

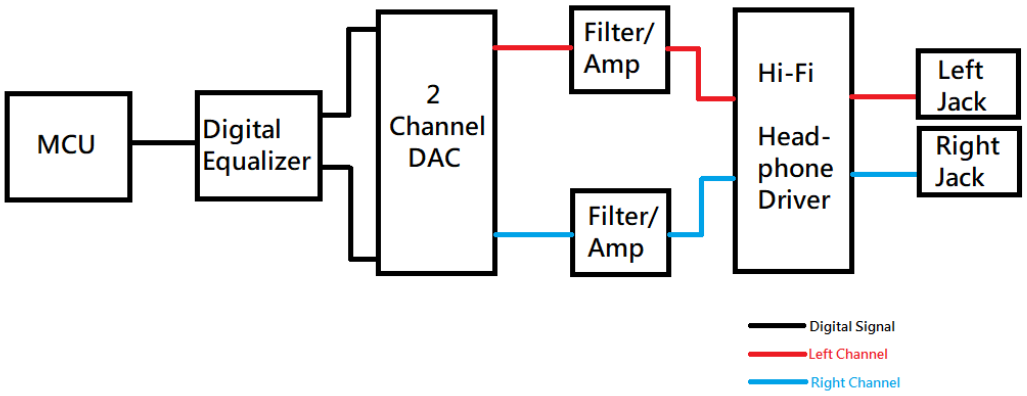


Figure 2. Audio Path Block Diagram

3.4. Modes of Operations

The system will have two modes of operation:

- Active mode: In active mode everything will be on, this is where the user can scroll through and select songs, playlists, and download music. In this mode, the display will stay on while music is playing.
- Listening mode: In this mode only the necessary components for audio transmission will be on, the display and all associated components will be off in order to save power. The DAP will automatically engage Listening mode if the touch screen isn't used for 3 minutes.

3.5. Users

The target audience for the portable hi-res digital audio player is music listeners of all age groups and musical inclination. User qualifications include the knowledge/ability to download music onto the device and an understanding of the English language, since the interface will only be in English as of the initial product release. Users with deeper knowledge about music and audio can also enjoy this DAP, with tuning capabilities through the on-board equalizer and many built in presents, advanced users can tune all their audio to their liking.

3.6. Support

A user manual will be written and distributed along with the digital audio player. The user manual will contain information such as device parameters, basic functions, appearance figure, and music download instructions. In addition, a guide on music tuning and recommended settings for certain genres of music will also be provided for users who want to amplify their experience. A user support/help line will also be provided.

4. Scenario(s)

4.1. The Advanced user

A music fanatic wants a DAP to miss with on the train ride or the plane in order to figure out his favorite equalizer settings for each genre of music. With the digital equalizer that comes with this DAP, the long battery life, and the compact package, this makes the perfect DAP for him, he can experiment with the mixer on the fly, and if he has a commercial equalizer at home or work, he can transfer his learnings and further his experiment there.

4.2. The Average User

A college student wants a quality DAP that supports his balanced headphones but doesn't want to spend too much. He has accumulated a large library of music over time and needs his device to be portable so he can take it to class. He doesn't know much about music, but enjoys a wide variety of genres, so he buys this DAP. With separate audio paths, large storage, a small package, and many preset equalization settings, it's the perfect choice.

5. Analysis

5.1. Summary of Proposed Improvements

- MCU – Cutting edge microcontroller technology will be used to make it easy for users to download and play music from the device.
- Battery – A power supply consisting of a rechargeable 5V battery and back up battery applications will allow users to listen to music for extended periods of time without having to charge the battery.
- Audio – A clear digital audio signal allows the user to listen to MP3 files clearly without any unwanted noise signals coming through the balanced headphones.
- User Experience – An easy-to-use touchscreen and manual user interface allows the user to make certain changes to the music they want to listen to and enhances the overall user satisfaction.

5.2. Disadvantages and Limitations

- Size – The size of the device has a limitation because the device must be portable.
- Battery life – Since the device is portable and isn't plugged into a wall, battery life will always be a concern on the user's mind.
- Lack of a Speaker – A good speaker can be bulky and power hungry, and with our design oriented towards listeners with an appreciation for music, we decided that no speaker is better than sub-par speaker.

5.3. Alternatives

- FLAC – Free Lossless Audio Codec is a very popular compressed lossless audio format. The file size of a FLAC format may not be as large as others, but the audio quality is the same as other audio files.
- WAV – Waveform Audio File Format is an uncompressed, lossy audio file format. Most DAPs can use WAV files, but the file sizes are large compared to other making WAV

files impractical to use unless format restoration or editing will be included in the project.

5.4. Impact

- Environmental - The Li-Ion/Li-polymer battery can be referred to as harmful to the environment. When not disposed of properly, a Lithium battery can contaminate the land and air.
- Society - Device allows people to enjoy music on the go.
- Ethics - The DAP is an ethical alternative to people listening to music over loudspeakers and disturbing others.

5.5. Development/Deployment Timeline

Our team of electrical engineers has been working diligently. Over the next 3-6 months, we will be focusing on finalizing product design, completing software development, and conducting software and hardware validation and verification testing for individual subsystems. Upon completion, we will begin integrating a final prototype. Our goal is to have a final prototype ready for demonstration at the end of 2023.

Portable Hi-Res Digital Audio Player

Rammi Hameed

Jake Flores

Seth Pregler

Patrick Westmoreland

FUNCTIONAL SYSTEM REQUIREMENTS

REVISION – 3.0

Dec 02 2023

INTERFACE CONTROL DOCUMENT

FOR

High Resolution Digital Audio Player

TEAM 25

APPROVED BY:

Seth Pregler

_____

Project Leader Date

_____

Porf. Wonhyeok Jang Date

_____

T/A Date

Change Record

Rev.	Date	Originator	Approvals	Description
1.0	02.18.2023	Rammi Hameed		Initial Release
2.0	04.23.2023	Seth Pregler Jake Flores		Addressed grader's comments
3.0	12.02.2023	Seth Pregler		Modified power and removed section on MP3. Added Patrick.

Table of Contents

List of Tables

No table of figures entries found.

List of Figures

1. Introduction

1.1. Purpose and Scope

The DAP shall be a portable device that plays audio at or exceeding CD quality. The biggest part in achieving the desired audio quality is the audio path. CD quality audio is 16 bits and 44.1 kHz refresh rate. This DAP shall meet said requirements and may even exceed them if the other subsystems allow, the audio path components should support up to 24 bits and 96kHz refresh rate.

Figure 1. Hi-Res DAP Conceptual Image

The following definitions differentiate between requirements and other statements.

Shall:	This is the only verb used for the binding requirements.
Should/May:	These verbs are used for stating non-mandatory goals.
Will:	This verb is used for stating facts or declaration of purpose.

1.2. Responsibility and Change Authority

All four team members share the responsibility of ensuring requirements are met and appropriate changes are made to each subsystem as needed. Verification will be done by Professor Nowka as needed. The subsystem responsibilities are as follows:

Audio Path: Rammi Hameed

Power and Mechanical: Jake Flores

Embedded Systems – Audio Processing: Seth Pregler

Embedded Systems – User Interface: Patrick Westmoreland

2. Applicable and Reference Documents

2.1. Applicable Documents

The following documents, of the exact issue and revision shown, form a part of this specification to the extent specified herein:

Document Number	Revision/Release Date	Document Title
IEEE 1857.5	01/30/2017	IEEE standard for advanced mobile and speech audio

2.2. Reference Documents

The following documents are reference documents utilized in the development of this specification. These documents do not form a part of this specification and are not controlled by their reference herein.

Document Number	Revision/Release Date	Document Title
1	02/12/2020	Guidelines for the creation of digital collections.
2	Feb. 2015	TPA6120A2 High Fidelity Headphone Amplifier.

2.3. Order of Precedence

In the event of a conflict between the text of this specification and an applicable document cited herein, the text of this specification takes precedence without any exceptions.

All specifications, standards, exhibits, drawings or other documents that are invoked as "applicable" in this specification are incorporated as cited. All documents that are referred to within an applicable report are considered to be for guidance and information only, except ICDs that have their relevant documents considered to be incorporated as cited.

3. Requirements

This DAP must be portable, play CD quality audio at minimum, have a respectable battery life, and an intuitive user interface. The audio quality and size restrictions are the biggest consideration in the audio path subsystem, the weakest link in the audio path must at least support 16 bits and 44.1kHz refresh rate to maintain CD quality, which isn't difficult on its own, but fitting it into a small, power efficient package is where the challenge lies.

3.1. System Definition

The challenge is to make a DAP that is portable and plays good quality audio while also being power efficient. The main subsystems involved are the audio path, microcontroller and software, power, and the enclosure. The DAP plays audio by retrieving the data from an SD card, sends it through the microcontroller which then sends it into the digital to analog converter. The converted signal is then sent through amplifiers and headphone drivers and into the listener's ear.

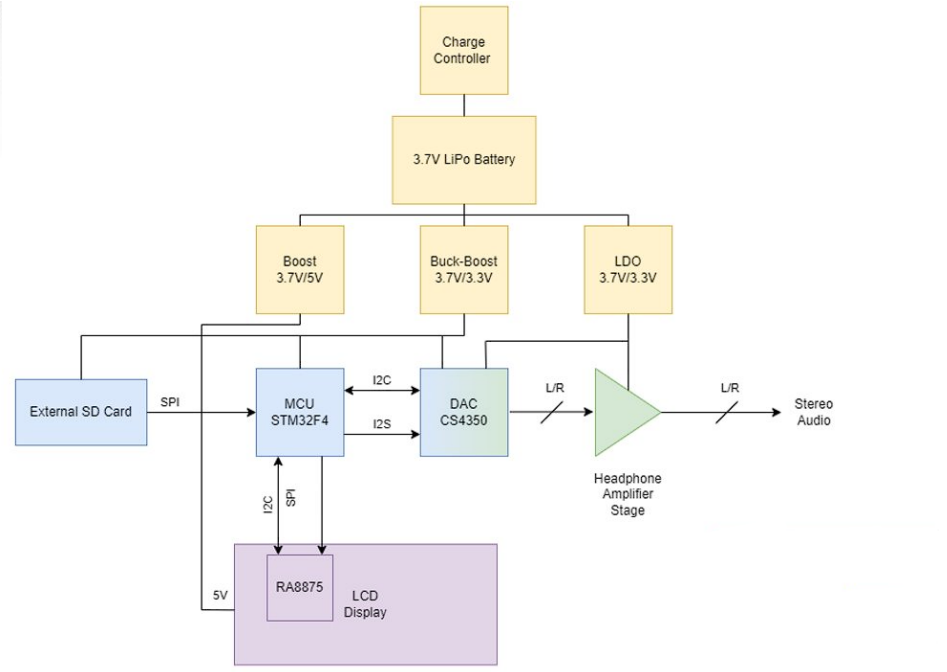


Figure 2. Block Diagram of System

The block diagram shows the path of the signal, after the data is pulled out of the SD card, it goes through the shown path into the DAC, and splits into a left and right channel, which then goes through the respective channels in the headphone jack. The headphone jack will be a 2.5mm 4 contact jack which supports balanced headphones. Balanced headphones provide much better audio quality by separating the left and right channels, giving the user a live performance-like experience through their headphones.

3.2. Characteristics

3.2.1. Functional / Performance Requirements

3.2.1.1. Audio Quality

The device shall output, at minimum, CD quality audio and abide by the following specifications:

- *Resolution:* Data resolution shall be at least 16 bits and sampled at a minimum of 44.1 kHz.
- *Total Harmonic Distortion:* Maximum allowable distortion level shall not exceed –40 dB total harmonic distortion.
- *Dynamic Range:* A minimum dynamic range of 96 dB.
- *Frequency Response*: A frequency response on the interval 10 Hz – 30 kHz. The deviation across this audible range shall be no more than +/- 0.5 dB.

- Channel bleed-over: there shall around 1% of signal bleed-over between the left and right channels.
- Load capabilities: The system shall handle up to 60 ohms of load.

Rationale: This is a requirement imposed by the project sponsor to ensure that a minimum audio specification is met. These requirements ensure a high level of fidelity.

3.2.2. Physical Characteristics

3.2.2.1. Portability

The DAP must fit in the user's pocket, that puts restrictions on the enclosure size, which then restricts the size of the circuitry that can be used. Making the device portable is essential for the design.

3.2.3. Electrical Characteristics

3.2.3.1. Power

The power supply must be small enough to fit inside of a portable enclosure and still be able to power the device for an extended period.

- *Battery Power:* The Digital Audio Player will be powered by a 3.7v Li-Ion/Li-Pol battery.
- *Charging Capabilities:* An integrated charging device mainly used for small portable applications will be used to charge the DAP. The charging device charges the battery in three phases: conditioning, constant current, and constant voltage. The charging device can operate from a micro-USB port.
- *DC-DC Buck-Boost Converter:* A DC-DC Buck-Boost Converter 3.7V-3.3V will be implemented to step down voltage from battery to all the digital components providing an efficient voltage signal
- *Voltage Regulator:* A voltage regulator 3.7V-3.3V will be implemented to step down the voltage from the battery to all the analog signals providing a clean voltage signal
- *Load Outputs:* The load will have a power output of 0.967W and a max amp draw of 0.4A.

3.2.3.2. Digital Input

The device shall support USB 2.0 High Speed data transfer at a rate of up to 480 Mbps.

3.2.3.3. Display and Graphics

The device shall feature a 16-bit color LCD display with a capacitive touch sensor. The display. Further system requirements are outlined below:

- *Screen Resolution*: The display shall have enough pixels to provide a clear and crisp image of the GUI.
- *Refresh Rate*: The display shall have a high-enough refresh rate to ensure low latency animations.
- *Color Depth:* The display shall feature 16-bit color.
- *Power Consumption:* The display shall be incorporated in such a way as to ensure minimal power consumption.

3.2.3.4. Control Interface

The device shall possess two main interfaces for control, namely, haptic control for powering the device and adjusting volume, as well as a touchscreen allowing the user a more intuitive mechanism of control. System requirements for the touchscreen is outlined below:

- *Touch Support:* The touchscreen shall be able to accurately detect and interpret touch inputs from the user.
- *Responsiveness:* The device touchscreen shall provide the user with responsive feedback and minimize input lag.
- *Integration:* The touchscreen shall be seamlessly integrated within the device.
- *Accuracy*: Touch inputs from the user shall be accurately detected, enabling precise interaction.

3.2.3.5. User Storage

The device shall provide a minimum of 64 GB of external storage to allow for storage of user data and system files via microSD cards with capacity up to 128 GB. Storage shall be non-volatile and capable of retaining data in the event of power loss.

3.2.3.6. Environmental Specifications

3.2.3.6.1 Temperature

- *Operating Temperature*: -40℃ – 85℃

The device enclosure shall have small perforations to aid airflow and manage temperature.

3.2.3.6.2 Moisture

The device holes will be lined with a hydrophobic mesh that is commonly used in portable audio devices. The mesh allows for airflow and clear audio quality while preventing moisture from entering the device.

3.2.4. Software Characteristics

3.2.4.1. Graphical User Interface

The Graphical User Interface (GUI) shall provide clear and intuitive navigation and control and abide by the following specifications:

- *Display:* The UI shall display relevant data and information clearly and effectively.
- *I/O:* The UI shall provide the user with a mechanism to input data, make selections and shall display output with low latency.
- *Accessibility:* The UI shall be designed to be accessible to users with wide ranges of ability.
- *Efficiency:* The UI shall be efficiently implemented to ensure minimal consumption of system resources.

3.2.4.2. Audio Playback Features

The system software will allow a user to play, pause and skip audio tracks. It will also allow a user to adjust the volume.

4. Support Requirements

The user will need a laptop or computer with a USB port to load audio files into the DAP, the user will be provided with a user manual detailing how to use the device, upload audio files, and manage storage.

Portable Hi-Res Digital Audio Player

Rammi Hameed
Jake Flores
Seth Pregler
Patrick Westmoreland

INTERFACE CONTROL DOCUMENT

REVISION – 3.0
02 December 2023

INTERFACE CONTROL DOCUMENT
FOR
High Resolution Digital Audio Player

TEAM 25

APPROVED BY:

Seth Pregler

_____

Project Leader Date

_____

Porf. Wonhyeok Jang Date

_____

T/A Date

Change Record

Rev.	Date	Originator	Approvals	Description
1.0	02.22.2023	Rammi Hameed Jake Flores Seth Pregler		Initial Release
2.0	04.23.2023	Seth Pregler Jake Flores		Addressed grader's comments. Removed USB interface section.
3.0	12.02.2023	Seth Pregler		Modified dimension. Updated display sections. Changed from PIC to STM. Updated DAC. Added Patrick.

Table of Contents

List of Tables

List of Figures

1. Overview

This Interface Control Document (ICD) for the Digital Audio Player (DAP) is a comprehensive document that outlines the physical interfaces, electrical specifications, and communication protocols used to integrate the system outlined in the Concept of *Operations* and *FSR* documents. In Section 3 of this ICD, the physical characteristics of the device are outlined. Section 4 details the electrical interfaces of the device, namely, input power requirements, signal interfaces, the user control interface, and microcontroller pin interfaces. Section 5 of this document covers communications interfaces such as SPI, I2S, and I2C. This document describes pin configurations, signal characteristics, data formats, and timing information characterizing each interface. This document serves to ensure the seamless integration of functional components.

2. References and Definitions

2.1. References

Concept of Operation for Portable Hi-Res Digital Audio Player (Rev. 1.0 – 07 Feb. 2023)

Functional System Requirements for Portable Hi-Res Digital Audio Player (Rev. 1.0 – 22 Feb. 2023)

I2S Bus Specification and User Manual (Rev. 3.0 – 17 Feb. 2022)

I2C Bus Specification and User Manual (Rev. 7.0 – 1 Oct. 2021)

2.2. Definitions

DAP:	Digital Audio Player
MCU:	Microcontroller Unit
DAC:	Digital to Analog Converter
GUI:	Graphical User Interface
SPI:	Serial Peripheral Interface
I2C:	Inter-Integrated Circuit
I2S:	Inter-Integrated Circuit Sound
LCD:	Liquid Crystal Display
USB:	Universal Serial Bus
TX:	Transmit
RX:	Receive
mA:	Milliamp
mW:	Milliwatt
MHz:	Megahertz

3. Physical Interface – Power and Mechanical Subsystem

3.1. Weight

- The expected weight of the PCB is: 2.2 ounces (+/- 50% for all measurements)
- The expected weight of the touch screen is: 2.4 ounces (+/- 50% for all measurements)
- The expected weight of the enclosure is: 3.3 ounces (+/- 50% for all measurements)
- The expected weight of the digital audio player is 8.1 ounces (+/- 50% for all measurements).

3.2. Dimensions

- The power PCB dimensions are: 3-inch x 3.5-inch
- The audio path PCB dimensions are: 3.5-inch x 4.5-inch
- The touch screen measurements are: 2.75-inch x 2-inch
- The dimensions of the DAP are: 4-inch x 6.5-inch x 1.75-inch

4. Electrical Interface

4.1. Primary Input Power – Power Subsystem

The DAP will be powered by a rechargeable 3.7v Li-Ion/Li-Pol battery. The 3.7V battery will have the voltage stepped up and down to accommodate software and audio path components that need either 5V or 3.3V input voltages.

4.2. Signal Interfaces – MCU and Audio Path Subsystems

The audio signal starts out as digital WAV file, then goes through equalization and into a DAC, which then puts it through an amplifier and then a headphone driver and into the listener's headphones.

4.3. DAC Pin Interface – MCU and Audio Path Subsystems

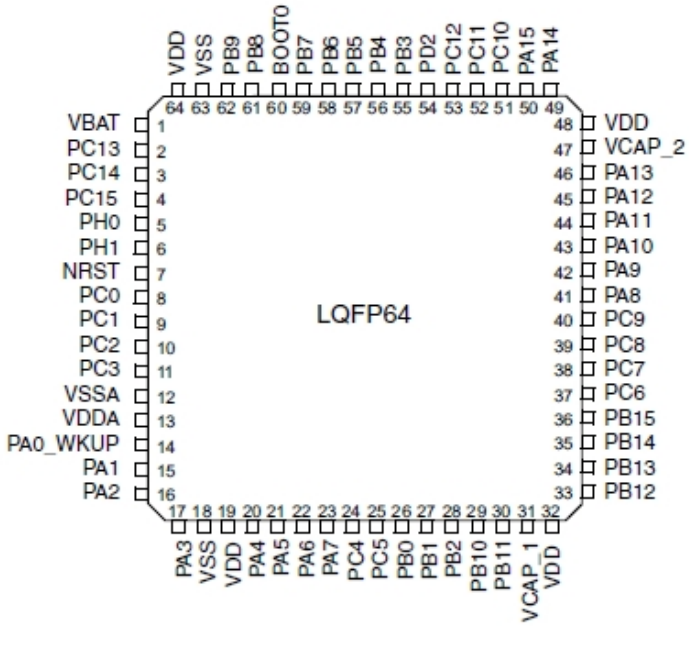
The tables below outline the pin interface for the CS4350 DAC.

Pin Name	TSSOP #	QFN #	Pin Description
VLC	5	2	**Control Interface Power** (*Input*) **-** Positive power for the hardware/software control interface
VD_FILT	6	3	**Regulator Voltage** (*Output*) - Filter connection for internal voltage regulator
GND	7, 19	4,16	**Ground** (*Input*) - Ground reference
RMCK	8	5	**Recovered Master Clock** (*Output*) - Outputs a master clock derived from LRCK
VLS	9	6	**Serial Audio Interface Power** (*Input*) **-** Positive power for the serial audio interface
SCLK	10	7	**Serial Clock** (*Input*) - Serial bit-clock for the serial audio interface
SDIN	11	8	**Serial Audio Data Input** (*Input*) - Input for two's complement serial audio data
LRCK	12	9	**Left/Right Clock** (*Input*) - Determines which channel, Left or Right, is currently active on the serial audio data line
TSTO	13	13	**Test Output** - These pins need to be floating and not connected to any trace or plane.
AOUTA+,- AOUTB+,-	14, 15, 22, 23	11, 10 19, 20	**Differential Analog Outputs** (*Output*) - The full-scale differential output level is specified in "DAC Analog Characteristics - Commercial (-CZZ,-CNZ)" on page 9.
$\overline{\text{AMUTEC}}$ $\overline{\text{BMUTEC}}$	16, 21	12 18	**Mute Control** (*Output*) - Control signals for optional mute circuit.
VBIAS+	17	14	**Positive Voltage Reference** (*Output*) **-** Positive reference voltage for the internal DAC
VA	18	15	**Analog Power** (*Input*) **-** Positive power supply for the analog section
VQ	20	17	**Quiescent Voltage** (*Output*) - Filter connection for internal quiescent voltage
$\overline{\text{RST}}$	24	21	**Reset** (*Input*) - When pulled low, device will power down and reset all internal registers to their default settings.
Control Port Definitions			
AD1/CDOUT	1	22	**Address Bit 1/Serial Control Data Out** (*I/O*) - Chip address bit 1 in I²C Mode or data output in SPI Mode
AD0/$\overline{\text{CS}}$	2	23	**Address Bit 0/Chip Select** (*Input*) - Chip address bit 0 in I²C Mode or Chip Select in SPI Mode
SDA/CDIN	3	24	**Serial Control Data In** (*I/O*) - Input/Output for I²C data. Input for SPI data
SCL/CCLK	4	1	**Serial Control Port Clock** (*Input*) - Serial clock for the control port interface
Stand-Alone Definitions			
DIF0 DIF1 DIF2	1 3 4	24 1 22	**Digital Interface Format** (*Input*) - Defines the required relationship between the Left Right Clock, Serial Clock, and Serial Audio Data
DEM	2	23	**De-emphasis** (*Input*) - Selects the standard 15 µs/50 µs digital de-emphasis filter response for 44.1 kHz sample rates
Thermal Pad *(QFN package only)*			
Thermal Pad	n/a	—	Thermal relief pad for optimized heat dissipation. See "QFN Thermal Pad" on page 40 for more information.

4.4. Microcontroller Pin Interface – MCU subsystem

The tables below outline the pin interface for the STM32F429 MCU.

Table 1: STM32F4 Pinout



5. Communications / Device Interface Protocols

5.1. Overview

The DAP supports communication with external and internal peripheral devices using the Serial Peripheral Interface (SPI), Universal Serial Bus (USB), Inter-IC Sound (I2S), and Inter-Integrated Circuit (I2C) interfaces. The SPI interface is used to communicate with the microSD card for music storage and the LCD display. The I2S and I2C interfaces are used for audio data transfer with external audio devices such as the audio codec as well as touch control features of the display.

5.2. SPI Interface – MCU Subsystem

The SPI interface is a full-duplex, synchronous, four-wire interface that supports communication with both the microSD card and the LCD display. The SPI bus consists of four (4) lines:

- *Clock (SCK):* Clock for serial peripheral interface.
- *Master Out Slave In (MOSI):* Output buffer for controller.
- *Master In Slave Out (MISO):* Input buffer for controller.
- *Slave Select (SS):* Used for selecting device.

The following parameters are defined for the SPI Interface:

- *Maximum Clock Frequency*: 66 MHz
- *Bit Order*: MSB first
- *Data Width:* 8- to 16-bit
- *TX/RX Buffer*: 4-level FIFO.
- *Interrupts*: Programmable interrupt generation for TX/RX operations.

5.3. I2S Interface – MCU Subsystem

The I2S interface is used for the transfer of digital audio data between the MCU and external audio devices such as the audio codec. The I2S buts consists of three (3) lines:

- *Serial Bit Clock (BCK):* BCK is the serial audio bit clock, and it is used to clock the serial data present on DATA into the serial shift register of the audio interface.

- *Word Select (LRCK):* This input selects the data word (L-Channel=Low, R-Channel = HIGH).

- *Serial Data Line (DATA):* This is a two (2) channel time-multiplexed serial data line for audio data.

Additionally, the codec requires a system clock (SCK) input for operating the digital interpolation filters and advance segment DAC modulators. These signals are shown in Figure 1 and Figure 2.

Figure 1: I2S Interface Diagram

The following parameters are defined for the I2S interface:

- *Data Word Length*: 16-, 24-bit selectable
- *Bit Order:* MSB first
- *Data Format:* PCM or DSD
- *Frame Synchronization*: Left-justified.

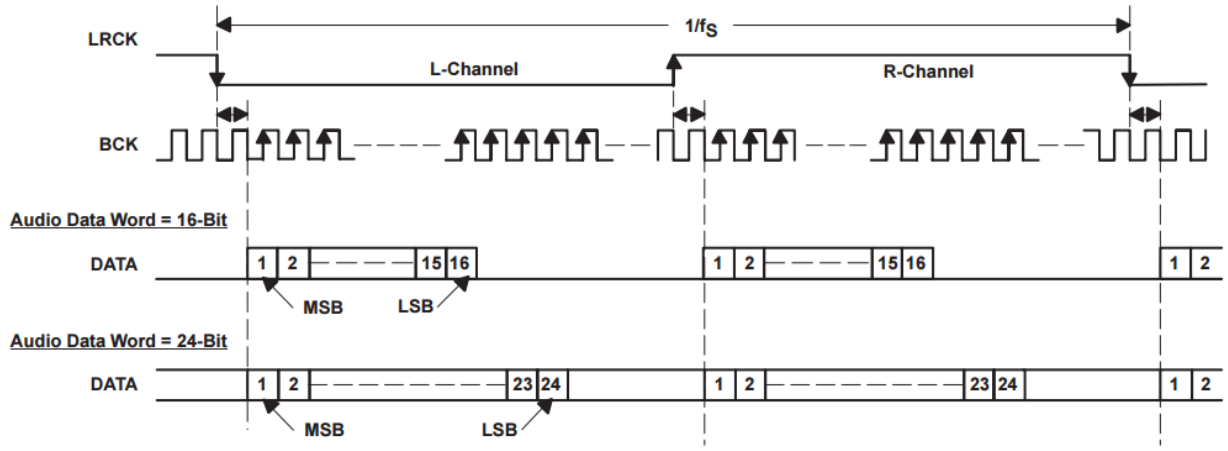


Figure 2: I2S Audio Data Input Format

5.4. I2C Interface – MCU Subsystem

The I2C interface is used for control and configuration of external audio devices such as the audio codec. The I2C bus consists of two (2) signal lines:

- Serial Data (SDA): 8-bit oriented, bidirectional.
- Serial Clock (SCL): Clock pulse generated by controller.

The following parameters are defined for the I2C interface:

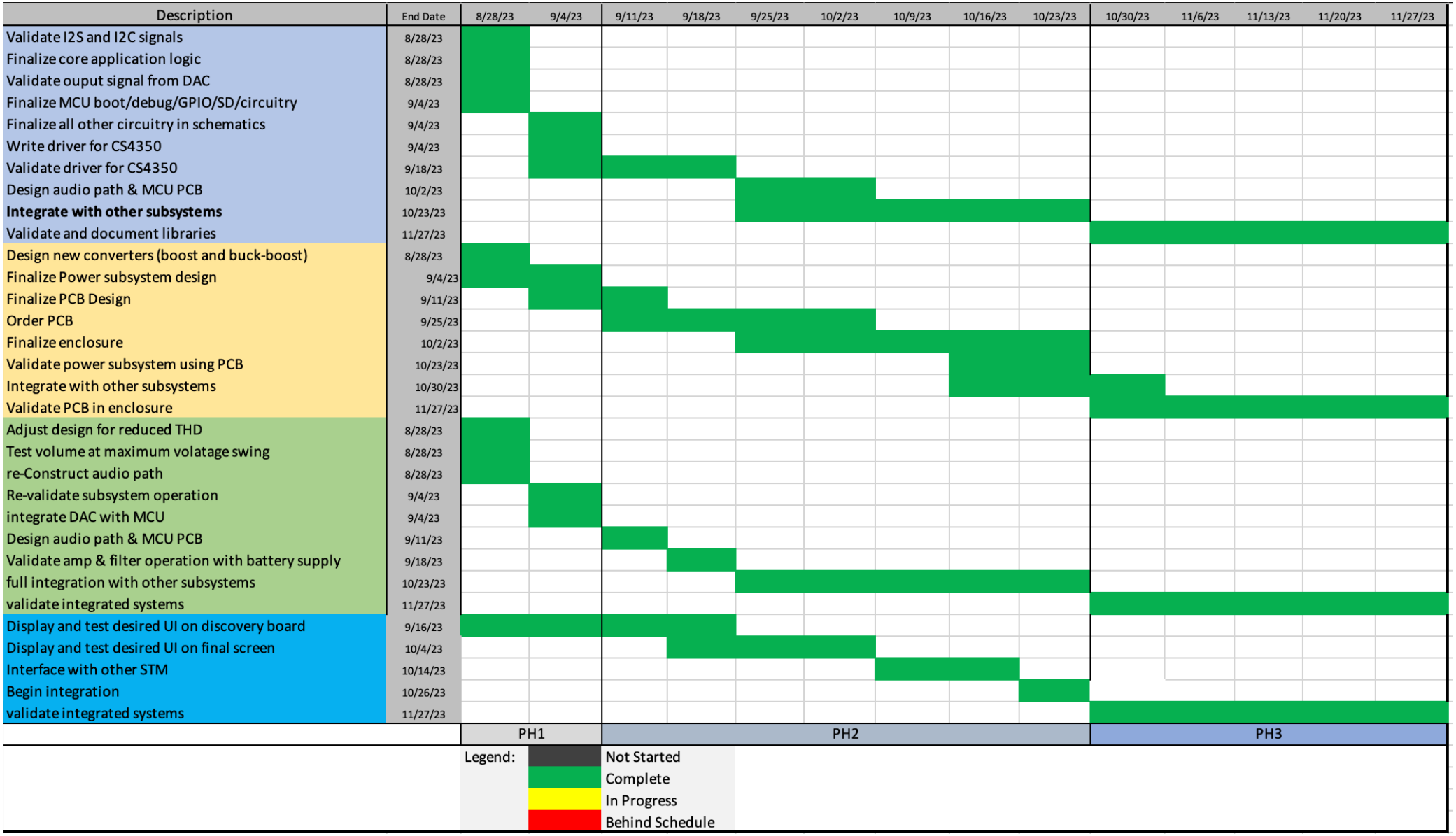
- Data Rate: 100 kbps (Standard-mode), 400 kbps (Fast-mode).
- Addressing: 7-bit addressable.
- Protocol: Standard-mode or Fast-mode.

Portable Hi-Res Digital Audio Player

Rammi Hameed
Jake Flores
Seth Pregler
Patrick Westmoreland

404 SCHEDULE

Team 25: Portable Hi-Res Digital Audio Player
Milestones



Team 25: Portable Hi-Res Digital Audio Player
Validation Plan

Test Name	Success Criteria	Status	Result	Responsible Engineer(s)
Battery Life	Battery lasts for extented period of time without needing to be plugged in to charge	TESTED	PASS	Jake Flores
Charging Capability	Charging Device recharges battery	TESTED	PASS	Jake Flores
Voltage Regulator	Voltage regulator steps down a clean voltage from 3.7V to 3.3V	TESTED	PASS	Jake Flores
DC-DC Converters	Converters step the voltage up to 5V and down to 3.3V	TESTED	PASS	Jake Flores
ON/OFF Switch	The DAP will begin operations when the power is on, and the DAP will have no power when the switch is in the off position	UNTESTED		Jake Flores
Volume Gauge	The volume levels will increase and decrease due to adjustments made on the volume gauge	TESTED	PASS	Jake Flores
DAC output	Reconstructed sin wave at the output of DAC has frequency less than or equal to +/-1% of original waveform	TESTED	PASS	Seth Pregler
Control Interface	LCD driver sends interrupts to MCU whenever user touches screen.	TESTED	PASS	Seth Pregler
User Storage Read/Write	Device is able to perform read and write from/to files on microSD.	TESTED	PASS	Seth Pregler
Audio Decoding	The device can read a .wav header and correctly buffer .wav data (compare vals using serial term) until EOF	TESTED	PASS	Seth Pregler
Internal Communications	MCU can interface with DAC using I2C. (Verify using scope and serial terminal)	TESTED	PASS	Seth Pregler
Internal Communications	MCU can interface with external LCD driver using I2C, Interupts, & SPI. (Verify using scope and serial terminal)	TESTED	PASS	Seth Pregler
Audio Quality	Audio quality must at least be CD quality, 16 bits 44.1kHz sampling rate.	TESTED	PASS	Rammi Hameed
Distortion	Maximum allowable distortion level shall not exceed -40 dB	TESTED	PASS	Rammi Hameed
dynamic range	Minimum dynamic range of 96 dB	UNTESTED		Rammi Hameed
Maximum output swing	Maximum voltage swing shall be 5V peak to peak	TESTED	PASS	Rammi Hameed
Load capability	Ensure the system maintains maximum swing with a 30 ohm load	TESTED	PASS	Rammi Hameed
Frequency response	A frequency response on the interval 10 Hz – 30 kHz. The deviation across this audible range shall be no more than +/- 0.5 dB.	TESTED	PASS	Rammi Hameed
Screen buttons	Display play, stop, buttons on LCD, when pressed GPio = 1highlight send digital pulse, call audio function (verify with audio)	TESTED	PASS	Patrick Westmoreland
Screen animation	Properly display selected images on discovery board at correct locations.	TESTED	PASS	Patrick Westmoreland
Song title display	Display potential song names and artists on screen that are read from file	TESTED	FAIL	Patrick Westmoreland
Display Driver	Device is able to output the above onto an external LCD	TESTED	PASS	Patrick Westmoreland

Portable Hi-Res Digital Audio Player

Rammi Hameed

Jake Flores

Seth Pregler

Patrick Westmoreland

SUBSYSTEM REPORTS

REVISION – 2.0

02 December 2023

SUBSYSTEM REPORT

FOR

Portable High-Resolution Digital Audio Player

TEAM 25

APPROVED BY:

Seth Pregler

_____

Project Leader Date

_____

Porf. Wonhyeok Jang Date

_____

T/A Date

Change Record

Rev.	Date	Originator	Approvals	Description
1.0	04/28/2023	Rammi Hameed Jake Flores Seth Pregler		Original Release
2.0	12/02/2023	Seth Pregler		Updating subsystem designs and project requirements

Table of Contents

List of Tables

List of Figures

1. Introduction

This documentation presents a detailed account of what our design team was able to accomplish over the past 12 months. Our system has been divided into five subsystems, namely, power, audio path, MCU, user interface, and mechanical; the scope of each is presented in the following sections. Each subsystem has been meticulously engineered to align with the design criterion introduced in the *Concept of Operations* at the beginning of this document.

2. Power Subsystem Report

2.1. Subsystem Introduction

The power subsystem is designed to ensure optimal functioning of the digital audio player. It is responsible for powering all the software and audio path components that enable the device to deliver a high-quality audio output. Also, as the device is portable, the power subsystem also serves the purpose of charging the battery that powers the entire device. Therefore, the power subsystem plays a critical role in the performance and longevity of the digital audio player.

2.2. Subsystem Details

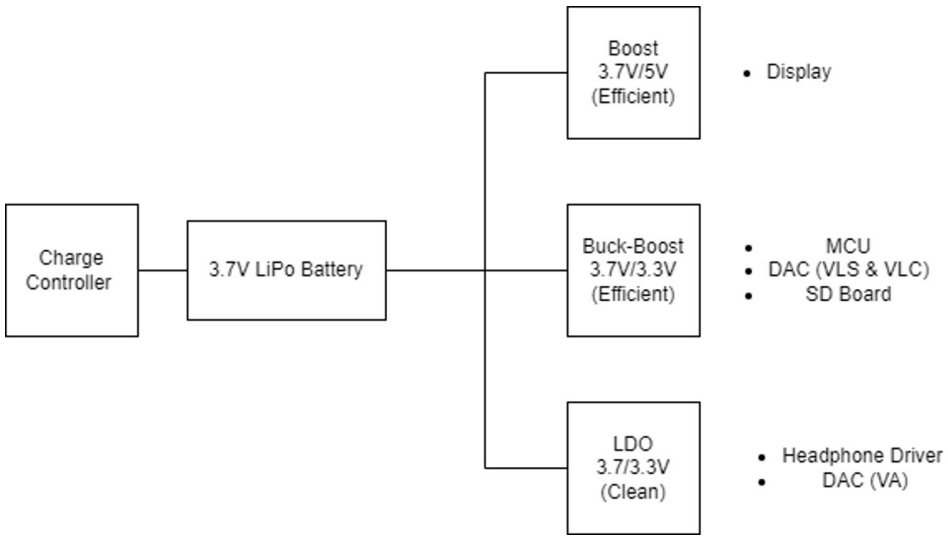


Figure 1: Power Subsystem Block Diagram

2.2.1. Voltage Output 5V

The 5V output of the power subsystem is achieved by utilizing the TI TPS61023DRLR Boost Converter. Engineered for efficiency and adaptability, this device is designed to boost the voltage from 3.7V to 5V. Initially integrated to power the display, recent design iterations have made the 5V output unnecessary. Despite this shift, the TI TPS61023DRLR retains its place on the power PCB for the power subsystem to be prepared for potential future expansions or adjustments.

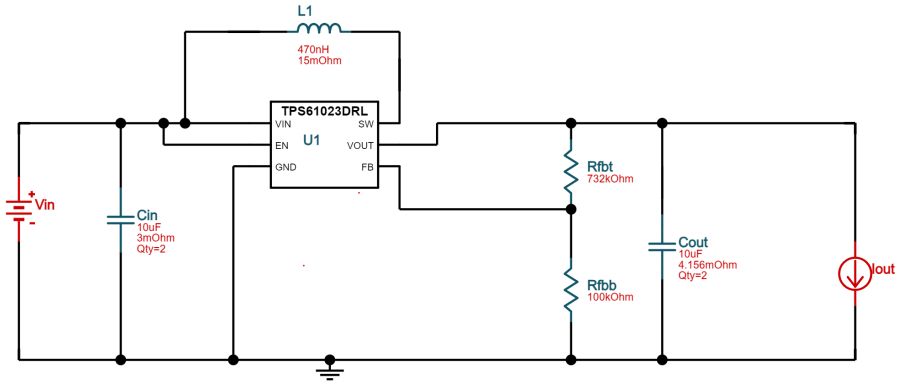
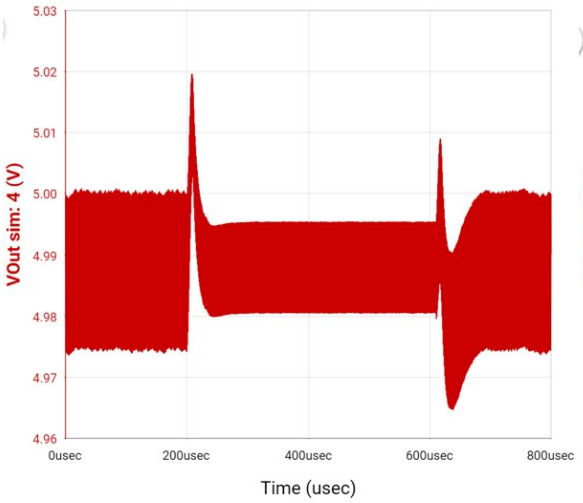


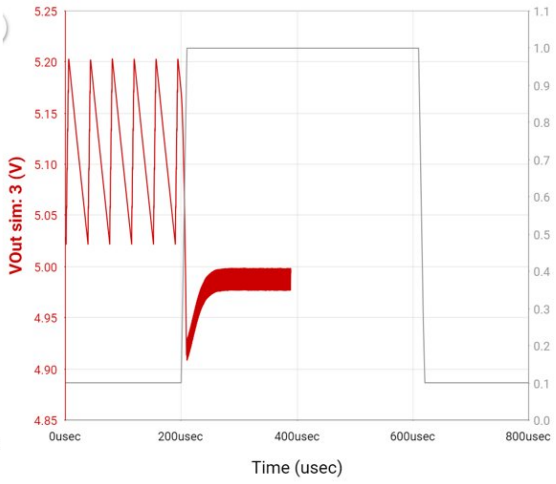
Figure 2: Boost Converter Schematic

After the boost converter was initially designed, the 5V output was tested and simulated using the TI Webbench Power Tool. This tool is known for its versatility, by examining factors such as voltage stability, current handling, and overall circuit efficiency. The designed boost converter circuit has an efficiency of 91.9%.



Performance Summary

Vout Maximum	Vout Minimum	Average Vout @Vin=4.2	Average Vout @Vin=3.0	Overshoot Settle Time	Undershoot Settle Time	Overshoot	Undershoot
5.02 V	4.96 V	4.99 V	4.99 V	0.03 ms	0.07 ms	0.03 V	0.03 V



Performance Summary

Sim ID	Vout Maximum	Vout Minimum	Average Vout @Iout=1.0	Average Vout @Iout=0.1
3	5.20 V	4.91 V	4.99 V	2.56 V

Figure 3: Boost Converter Simulations

2.2.2 Voltage Output 3.3V (Digital)

The 3.3V output for all the digital components is accomplished though the implementation of the TI TPS630252YFFR buck-boost converter. The buck-boost converter is designed to efficiently regulate the voltage, lowering the voltage down from 3.7V down to 3.3V. In the digital audio player, all the components that the power subsystem will be powering are either digital or analog.

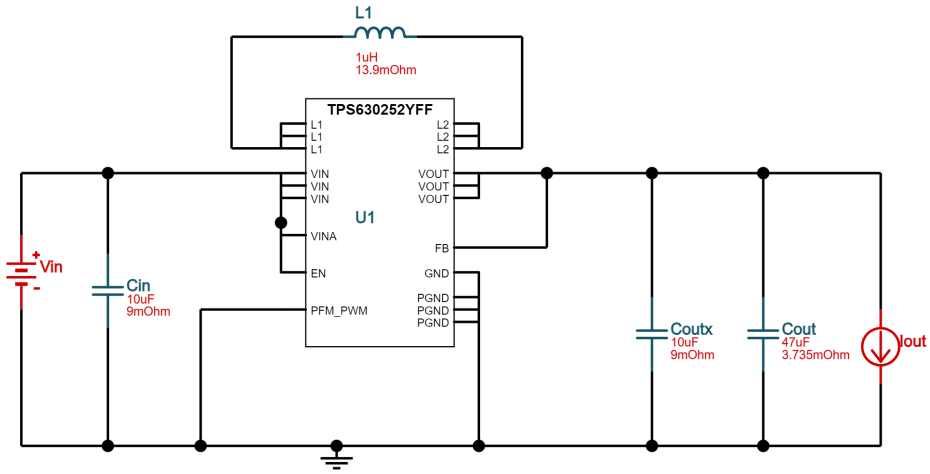
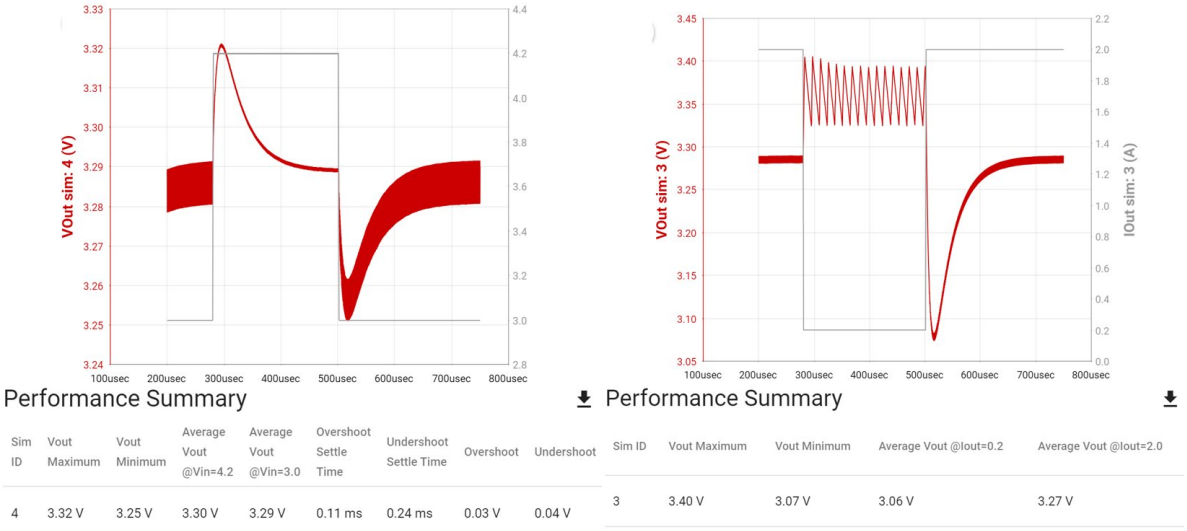


Figure 4: Buck-Boost Converter Schematic

The reason that this converter is used to only power all the digital components vs. all the components is because the digital components (such as the microcontroller) need to be efficient, and the potential noisy signal of the converter does not hinder the digital components at all. The rest of the components are analog components associated with the audio path which cannot have any type of noisy voltage signal. The analog components will be explained in the next section.

After the buck-boost converter was initially designed, the 3.3V output was tested and simulated using the TI Webbench Power Tool. This tool is known for its ability to examine and simulate circuit designs through various factors such as voltage stability, current handling, and overall circuit efficiency. The designed buck-boost converter circuit has an overall efficiency of 94.4%.



Performance Summary

Sim ID	Vout Maximum	Vout Minimum	Average Vout @Vin=4.2	Average Vout @Vin=3.0	Overshoot Settle Time	Undershoot Settle Time	Overshoot	Undershoot
4	3.32 V	3.25 V	3.30 V	3.29 V	0.11 ms	0.24 ms	0.03 V	0.04 V

Performance Summary

Sim ID	Vout Maximum	Vout Minimum	Average Vout @Iout=0.2	Average Vout @Iout=2.0
3	3.40 V	3.07 V	3.06 V	3.27 V

Figure 5: Buck-Boost Converter Simulations

2.2.2. Voltage Output 3.3V (Analog)

The 3.3V output for all the analog components is accomplished by using the Analog Devices ADP3339-3121784 voltage regulator. As a low dropout regulator (LDO), this component's main job is to maintain stable output voltage amidst varying input levels. With the minimal dropout, it minimizes power dissipation, enhancing overall energy efficiency. The regulator is also equipped with protective features against overcurrent and overtemperature events.

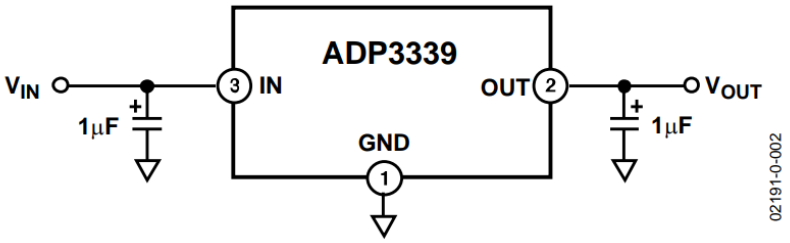


Figure 6: Voltage Regulator Schematic

The reason this component is used to power all the analog components (audio path components) is because the audio path needs clean voltage to have the best sound quality from a voltage standpoint.

2.2.3. Charging Capabilities

The charging operations are facilitated through the utilization of the Adafruit 1904 MiniLiPo Battery charger. This charger is equipped with a micro-USB connection to charge the battery and LEDs to inform the user when the battery cell is charging (red light) and when it is fully charged (green light). Furthermore, the charger features through-hole connections, establishing a direct link between the battery and power PCB, creating a cohesive system. The charging process itself is performed in three stages: preconditioning charging, constant current, constant voltage. This approach ensures the battery goes through a comprehensive and efficient charging cycle. When looking at the charging rate, you can estimate how long it takes to charge the battery by using the following equation:

$$\frac{battery\ capacity}{rate} \cdot 125\%$$

The 3.7V LiPo battery used for the digital audio player has a capacity of 1200mAh, and the charging rate is 500mA. The estimated time it would take the battery to fully charge is about 3 hours.

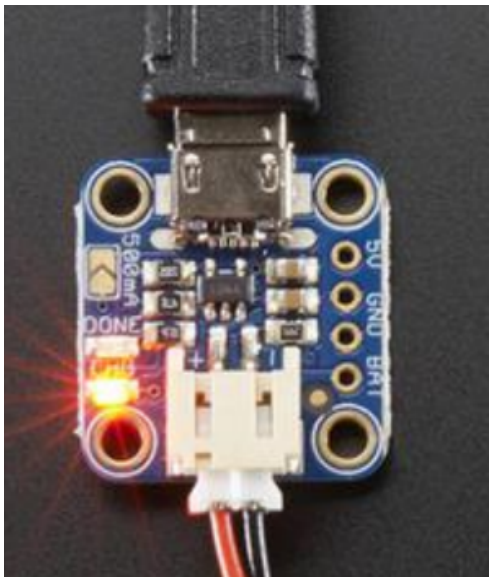


Figure 7: Charging Circuit

2.3. Subsystem Validation

2.3.1. Voltage Output 5V

Due to modifications in the display design that rendered the 5V boost converter circuit redundant, the circuit did not complete the validation process.

2.3.2. Voltage Output 3.3V (Digital)

For the 3.3V (Digital) output validation, line and load tests were run using an E-Load machine. These tests were run to determine the system's behavior under conditions such as: no load, max load, and nominal load.

Table 1: 3.3V (Digital) E-Load Test

Load Test			Line Test: (No load)			Line Test: (Load)		
Vin	Vout	I	Vin	Vout	I	Vin	Vout	I
3.7	3.276	0	3.0	3.325	0	3.0	3.279	0.148
3.7	3.271	0.038	3.2	3.326	0	3.2	3.235	0.148
3.7	3.266	0.092	3.4	3.326	0	3.4	3.237	0.148
3.7	3.262	0.141	3.6	3.327	0	3.6	3.238	0.148
3.7	3.256	0.198	3.8	3.329	0	3.8	3.243	0.148
3.7	3.251	0.237	4.0	3.329	0	4.0	3.245	0.148
3.7	3.247	0.291	4.2	3.3	0	4.2	3.246	0.148
3.7	3.243	0.35						
3.7	3.239	0.407						
3.7	3.238	0.443						
3.7	3.236	0.512						

2.3.3 Voltage Output 3.3V (Analog)

For the 3.3V (Analog) output validation, line and load tests were run using an E-Load machine. These tests were run to determine the system's behavior under conditions such as: no load, max load, and nominal load.

Load Test			Line Test: (No load)			Line Test: (Load)		
Vin	Vout	I	Vin	Vout	I	Vin	Vout	I

Table 2: 3.3V (Analog) E-load Test

3.7	3.661	0	3.0	2.966	0	3.0	2.903	0.04
3.7	3.61	0.022	3.2	3.163	0	3.2	3.102	0.04
3.7	3.601	0.033	3.4	3.363	0	3.4	3.301	0.04
3.7	3.595	0.041	3.6	3.561	0	3.6	3.5	0.04
3.7	3.587	0.055	3.8	3.761	0	3.8	3.699	0.04
3.7	3.585	0.063	4.0	3.96	0	4.0	3.898	0.04
3.7	3.583	0.071	4.2	4.159	0	4.2	4.098	0.04

2.3.4 Charging/Discharge Capabilities

For the charging/discharging validation charging and discharging tests were performed. For the discharging test, the battery was connected to the IMAX B6AC Dual Power Charger and discharged the battery.

Figure 8: Discharge Test

The discharge time was 1hr 23min 21sec at 0.5A. The max amp pull of the digital audio player is about 0.4A, so it is expected that the battery discharge of the digital audio player is about 2 hours.

A charging test was also conducted by connecting the battery to the charging chip and utilizing the micro-USB connection. The test duration spanned three hours.

Table 3: Charging Test

Tim	0:00	0:15	0:30	0:45	1:00	1:15	1:30	1:45	2:00	2:15	2:30	2:45	3:00
Volt	3.29	3.46	3.53	3.54	3.55	3.56	3.58	3.60	3.62	3.63	3.65	3.66	3.67

2.4. Subsystem Changes

Over the duration of this project, the power subsystem underwent major changes from a design standpoint. The design started off with a 5V (boost converter to voltage regulator) and a 3.3V (buck converter) output for both analog and digital components. Ultimately, the power subsystem design was finalized with a 5V output, 3.3V digital output, and 3.3V analog output.

2.5. Subsystem Conclusion

Overall, the power subsystem is responsible for charging the 3.7 LiPo battery and powering all the software and audio path components for the digital audio player. A successful power subsystem was created producing 5V and 3.3V outputs for the appropriate components, but design changes can be made to make the system more efficient and optimize the audio quality from a power standpoint.

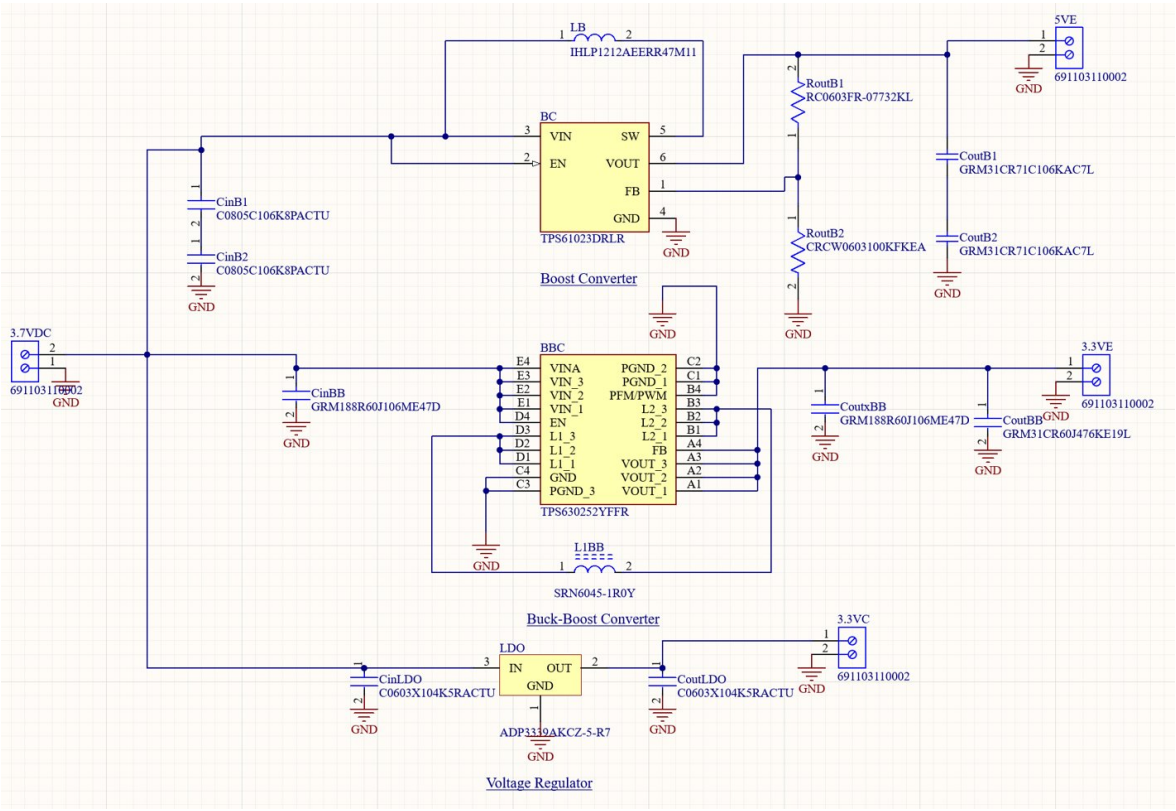


Figure 9: PCB Schematic

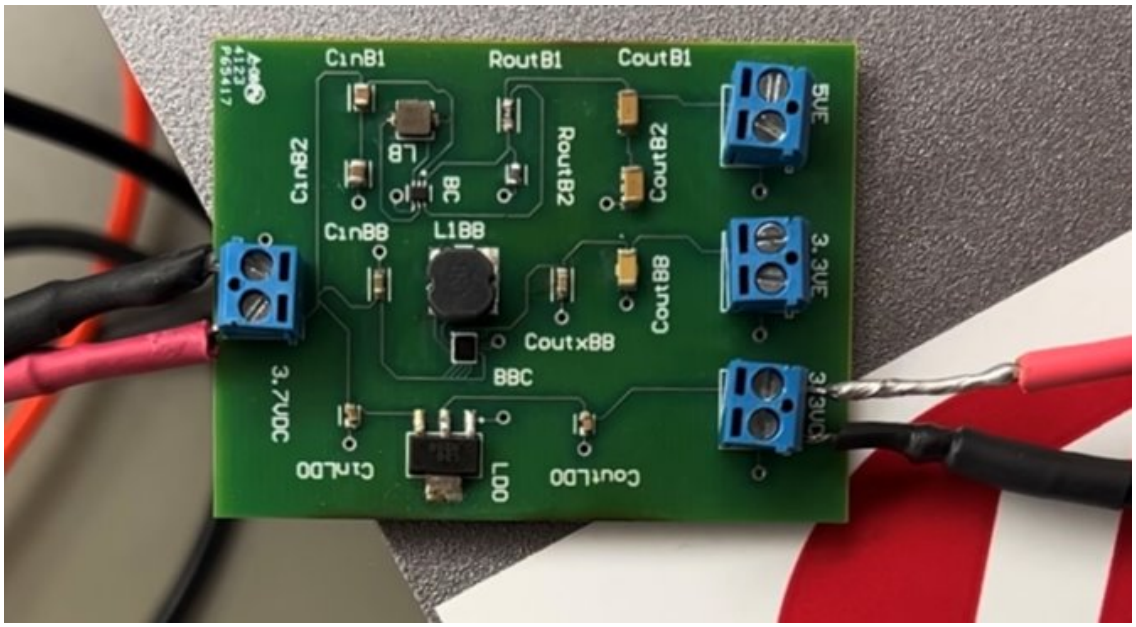


Figure 10: PCB Layout

3. Audio Path Subsystem

3.1. Subsystem Introduction

The audio path subsystem is where the audio quality is made, in order to translate that to hardware, this subsystem needs to convert the digital signals from the microcontroller into an analog sinusoid that can be filtered and amplified and eventually heard by the user in the form of audio. The biggest considerations when designing an audio path that maximizes audio quality are cutoff frequencies, gain, and total harmonic distortion. The human hearing range is about 20 Hz to 20 kHz, and those are theoretically the perfect cutoff frequencies for the filters, but in reality, frequencies slightly higher than 20kHz provide signal stability that is necessary for good audio quality. Gain and total harmonic distortion go hand in hand, not enough gain and the audio would not be loud enough, but T.H.D. goes up with gain, so too much gain in one stage causes T.H.D. to be too high. This subsystem is comprised of a DAC and one amplification and filtering stage. The system diagram is shown below in *Figure 11.*

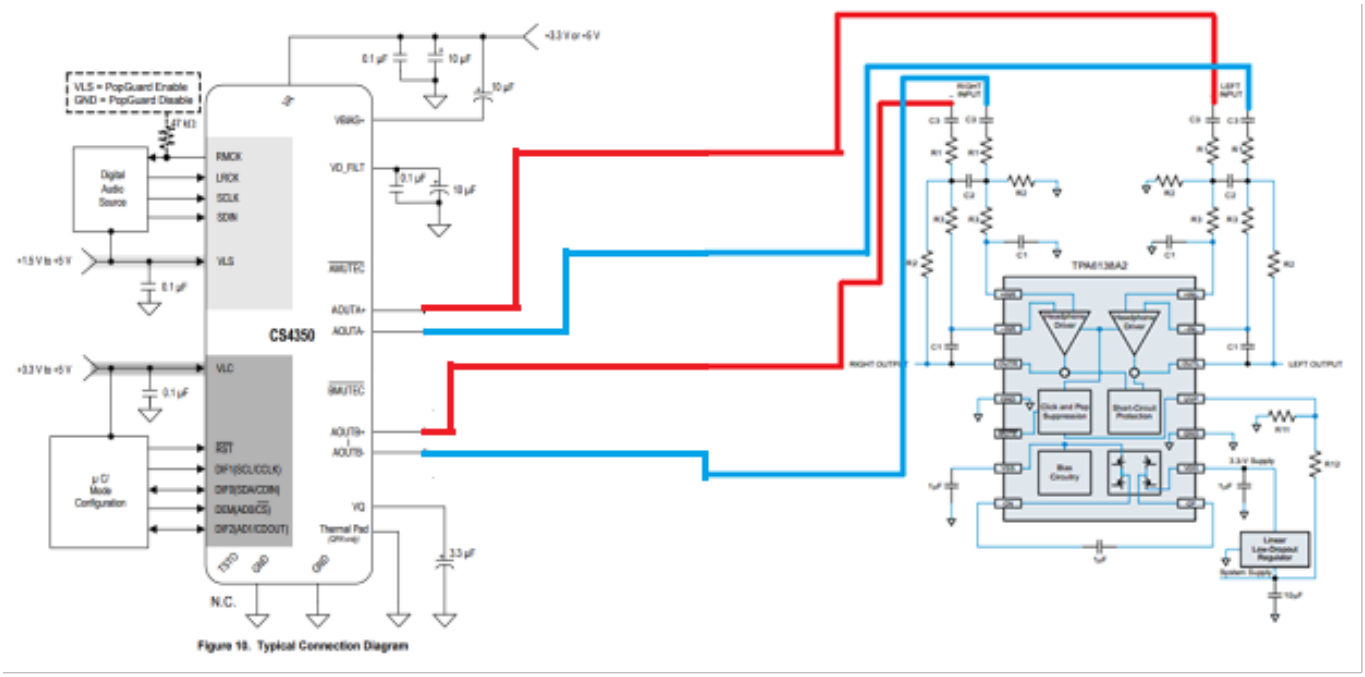


Figure 11: Audio Path Subsystem Diagram

3.2. Subsystem Details

3.2.1. Digital to Analog Converter

The first part of this subsystem is a Digital to analog converter from Cirrus Logic that takes in the digital signals from the MCU and turns them into the analog sinusoids that get filtered and amplified in the following stage. Some of the highlights of the DAC are the very high refresh rate, goes up to 192 kHz, we need 44.1 kHz for our audio, 24-bit capability, -91 dB T.H.D. attenuation, and a 109 dB dynamic range. The circuit schematic for the DAC is shown in *Figure 12.* This DAC is a voltage output DAC, outputting a 5V peak-peak signal for each channel.

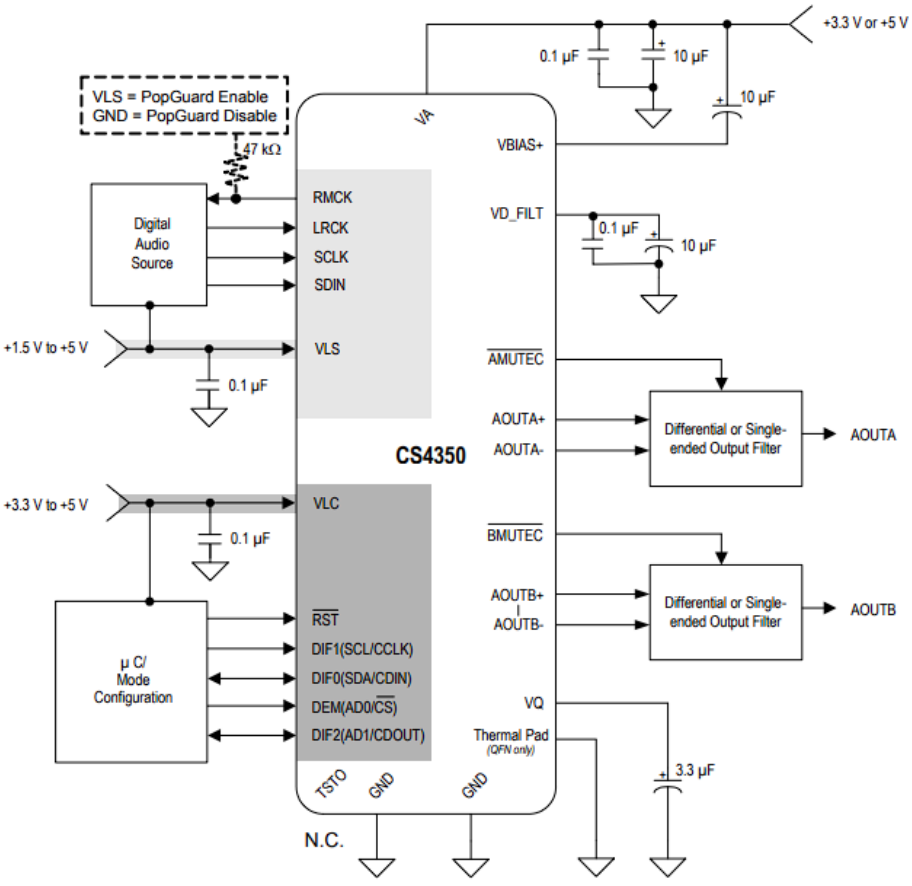


Figure 10. Typical Connection Diagram

Figure 12: DAC Circuit Schematic.

3.2.2. Amplification/Headphone Driver Stage

5V at the output is just large enough voltage for full volume for the user, but the design has a gain of about 1.1 V/V, giving some headroom for the full volume level. This stage will be a band-pass filter with cutoff frequencies at 10 Hz and 28.77 kHz and a gain of 1.1 V/V as mentioned. The amplification here will be done with a Texas Instruments headphone driver, which is comprised of two audio operational amplifiers, which take in a differential input for the left and right channels and output a single voltage for each channel, pop suppression circuitry that make muting and unmuting pop-free for the user, and an internal charge pump that lets this this driver use a single positive supply and creates the negative rail internally. The schematic for the amplification circuit is shown in *Figure 15*, and the bode plot for this stage is shown in *Figure 16*.

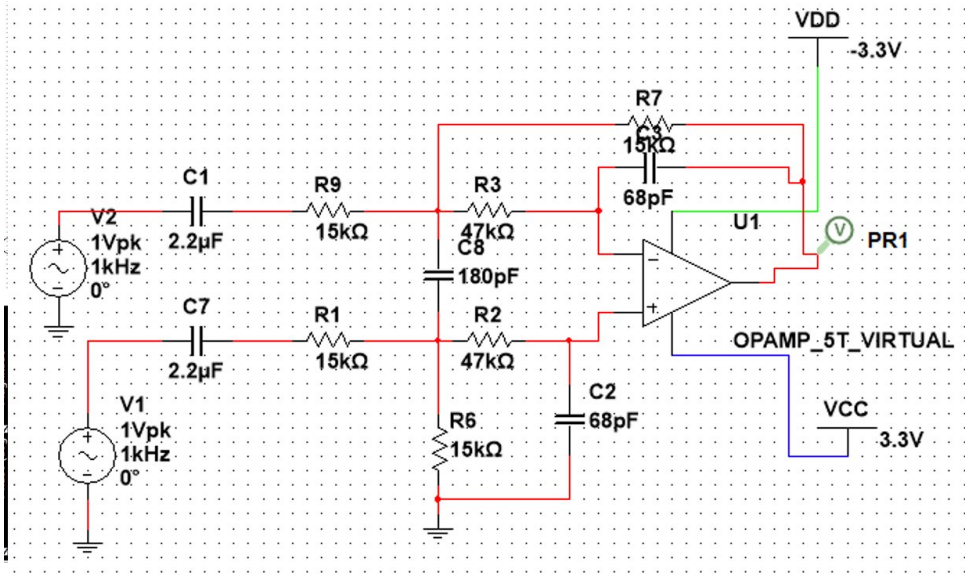


Figure 13: Amplification stage 2 Circuit Schematic
(Two of these will be in the audio path)

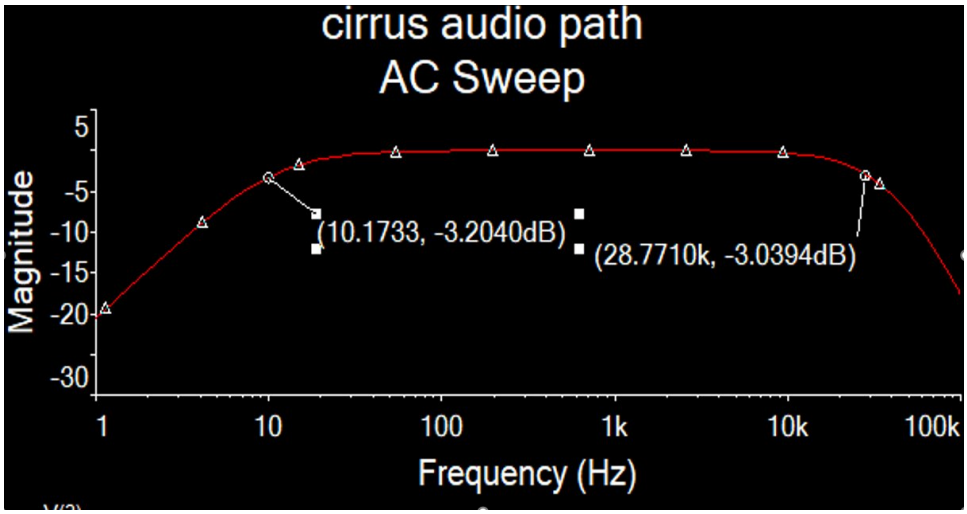


Figure 14: Amplification stage 2 Bode Plot

3.2.3. Output of the subsystem

The bandpass filter/amplifier stage outputs a 5.5 V peak-to-peak signal that is filtered slightly outside of the human hearing range. The circuit schematic shown above will be for one channel only for convenience, the system will have two of those circuits. The output of the subsystem will be connected to a TRRS 4 conductor headphone jack which supports balanced headphones, two of those contacts will be the left and right channel outputs, and the other two contacts will be left and right channel grounds.

3.3. Subsystem Validation

The main things to validate for this subsystem are the frequency response, T.H.D., output voltage swing, dynamic range, and load support.

3.3.1. Frequency response

3.3.1.1. Headphone Driver

This stage needed to have a gain of 1.1 V/V and a pass band between 10 Hz and 28 kHz. The bode plot for this stage is shown in *Figure 23* below.

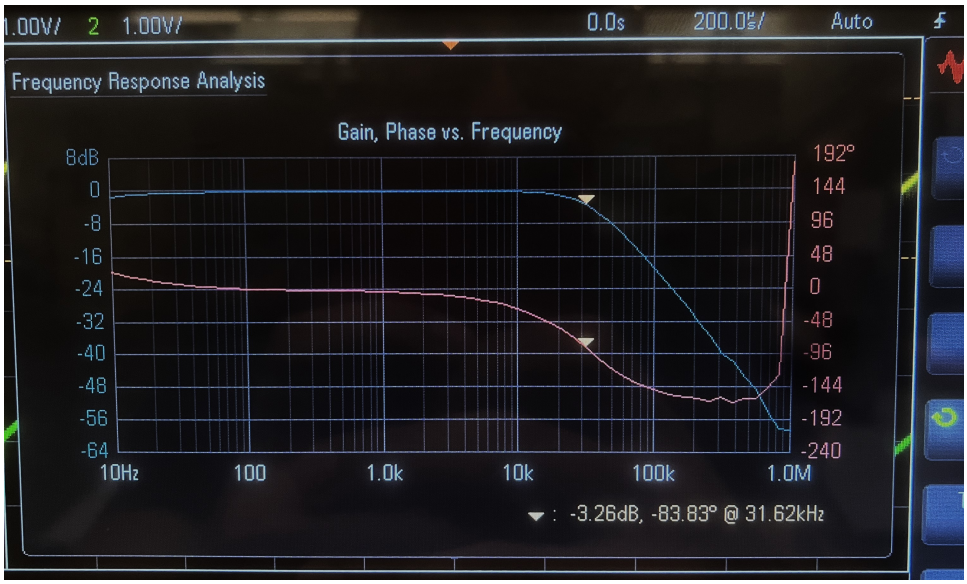


Figure 15: Headphone amp upper cutoff frequency

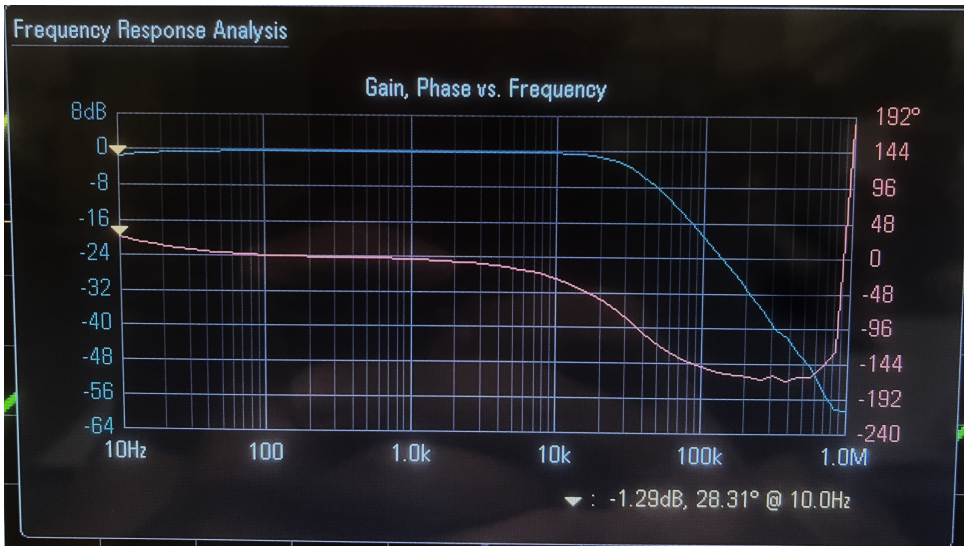


Figure 16: Headphone amp Lower cutoff frequency

The passband gain is just over 0 dB as expected, and although the passband isn't exactly what was intended, a passband between 8 Hz and 30 kHz is still satisfactory.

3.3.2. Total Harmonic Distortion

The target T.H.D. was -40dB for the subsystem, the T.H.D. was measured at -41 dB satisfying the requirement. *Figure 23.*

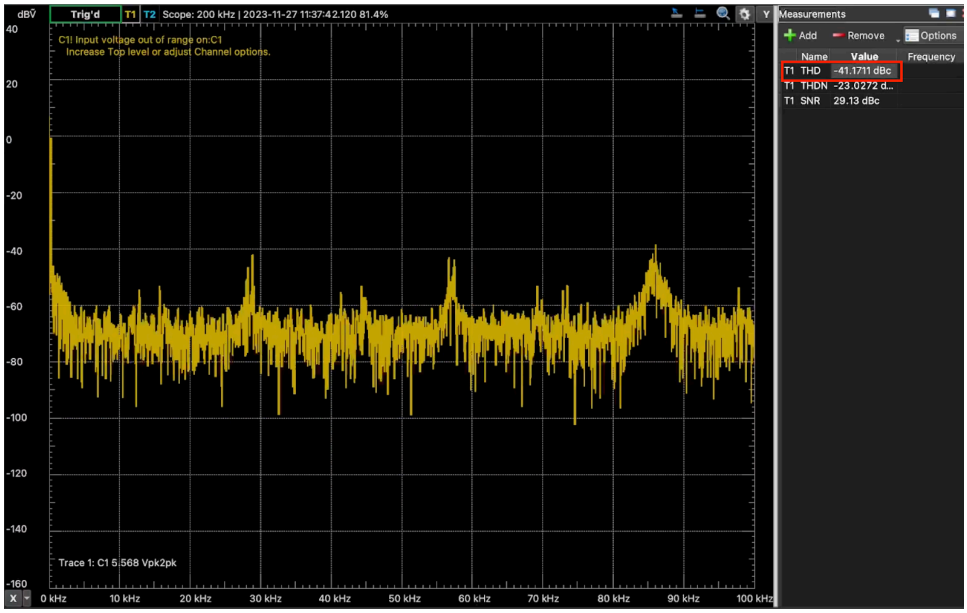


Figure 17: Total Harmonic Distortion of the system

3.3.3. Output Voltage swing

The target output voltage swing was 5V at the minimum, more voltage swing translates more volume capability for the user, so more voltage swing is a positive, as demonstrated in *Figure 24* below, the subsystem output was at 5.5V which exceeded expectations the voltage swing requirement, making the voltage swing aspect of the subsystem a big success. In addition, the system had a maximum unclipped swing of 6.9V with the battery supply, since the battery supply output was slightly higher than 3.3V.

Figure 18: Full Volume voltage swing.

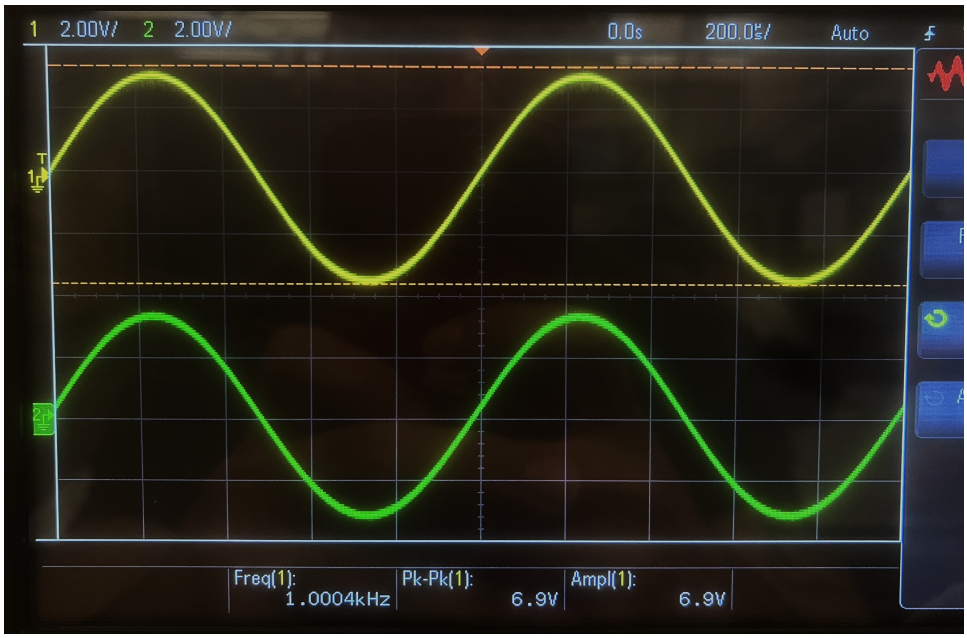


Figure 19: Maximum Unclipped swing at the output

3.3.4. Channel bleed-over

Since our DAP is going to support balanced headphones, minimizing channel bleed-over is very important for audio quality, the target was less than 1% bleed-over, this was not tested since the headphone amplifier IC was damaged and the left cannel output was oscillating rail to rail no matter the input.

3.3.5. Load capabilities and dynamic range

The load capabilities of the subsystem were tested with headphones that have up to 60 ohms of impedance and the maximum voltage swing was maintained, indicating a success that exceeded the design specification.

The dynamic range was untested. In order to test dynamic range, a full voltage signal must be passed, and the amplitude is measured. Then the input signal was to be attenuated by –60 dB and then the fundamental harmonic was to be removed from the output and the remaining amplitude be measured. Then the gain between the maximum swing and the noise was to be the dynamic range. The attenuated input was impossible with the MCU and therefore dynamic range was untested.

3.4. Subsystem Conclusion

Overall, the subsystem met all requirements, with dynamic range and channel to channel bleed over remain untested. Given the data sheets of the IC and the existence of surface mount components and a well-designed PCB, there should be no magnetic coupling, mitigating noise and bleed over. The PCB design is shown below in *Figure 21.*

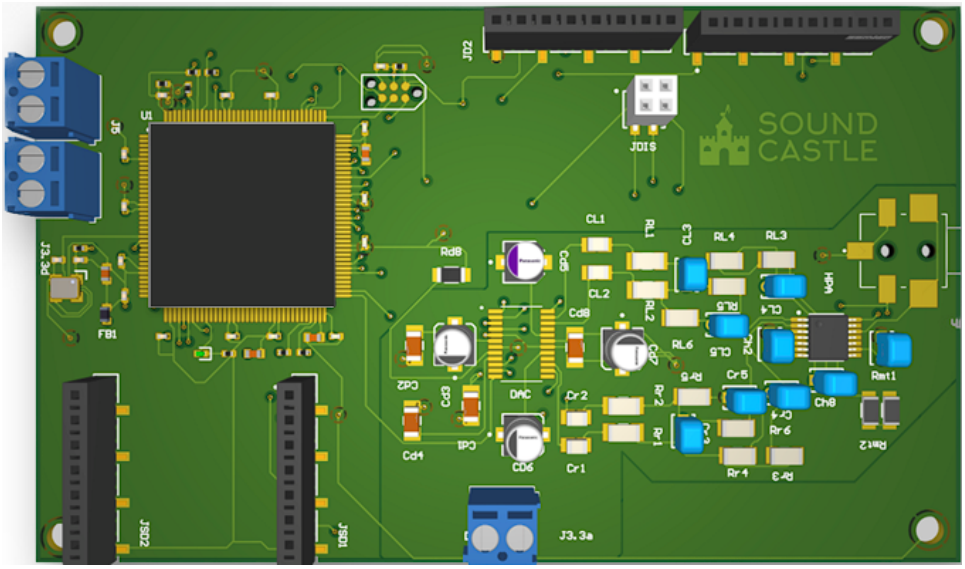


Figure 20: Complete Subsystem PCB

4. Audio Processing Subsystem

4.1. Subsystem Introduction

The audio processing subsystem is responsible for handling user audio files, decoding audio files, the music player application logic, and sending a digital bit stream to the DAC. Additionally, the audio processing subsystem is partly responsible for system integration with the LCD display meaning that the audio processing subsystem must interface with the LCD and touch drivers.

4.2. Subsystem Details

A block diagram of the audio processing subsystem is shown below.

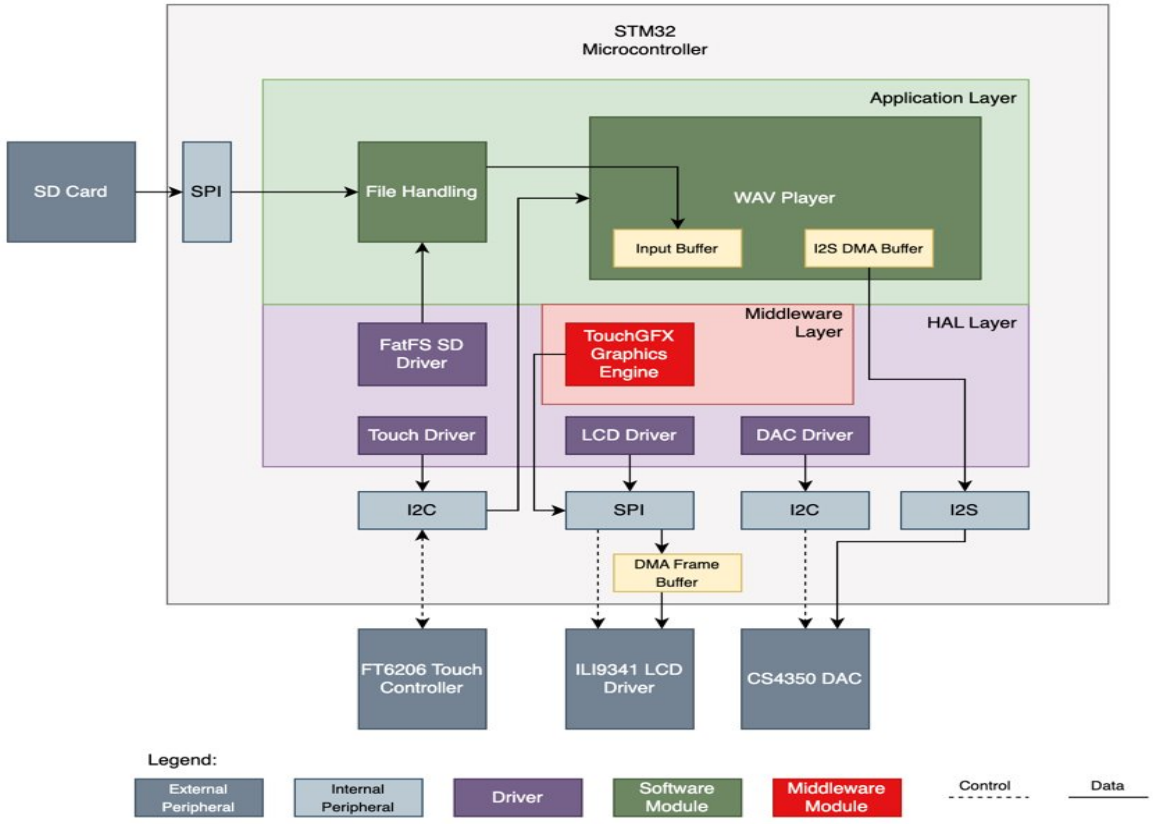


Figure 21: Audio Processing Subsystem Block Diagram

4.2.1. Hardware

The digital audio player is built using the STM32 family of microcontrollers which serves as the main application processing unit and interfaces with various external peripherals. These peripherals include an SD card for music storage, an LCD display, and a DAC. The microcontroller runs at a clock frequency of 180 MHz and includes 2 MB of flash memory and 256 kB of SRAM for program and data memory. The device requires a DC voltage between 2.1V and 3.6V for proper operation.The exact STM32 microcontroller used is the STM32F429ZIT which is a 144-pin TQFP device.

For subsystem development and verification, the STM32F429 Nucleo development board was used because of its rich peripheral set and its resemblance to our final system. Additionally, a breakout board was manufactured which included the Cirrus Logic CS4350 DAC which was chosen for our final system. The development board and DAC breakout boards are presented below.

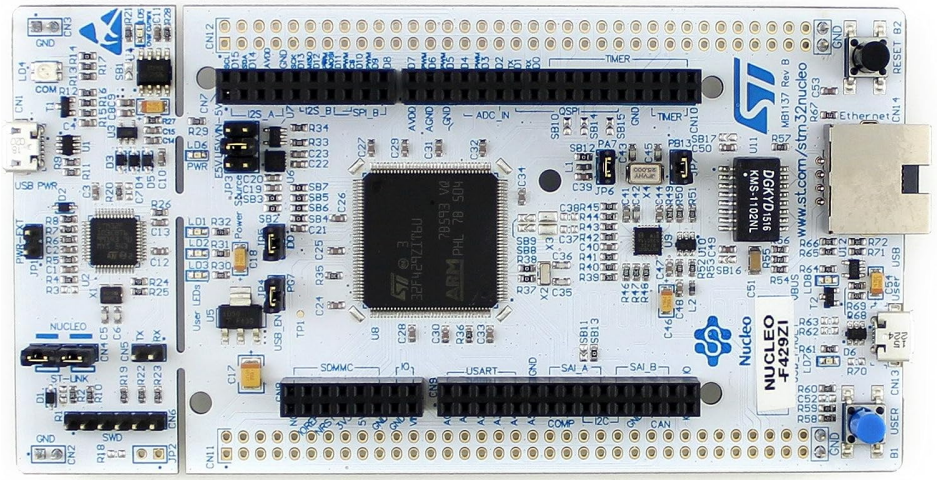


Figure 22: STM32F429 Nucleo Development Board

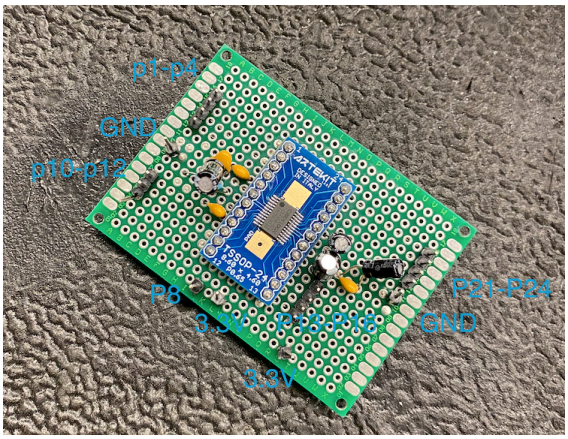


Figure 23: DAC Breakout Board

4.2.1.1. Configuring I/O

One of the challenges associated with embedded system design is working with limited IO. STM32 microcontrollers offer highly configurable IO ports whereby multiple functions are internally mapped to each general-purpose IO (GPIO) pin through multiplexers. This allows for at most one peripheral function to be mapped to an IO pin at any given time. It is up to the embedded programmer to ensure there are no conflicts between peripherals sharing the same IO pin. Each IO pin has sixteen alternative functions that can be configured through the `GIPOx_AFRL` (for pins 0 to 7) and `GPIOx_AFRH` (for pins 8 to 15) registers. Some alternative functions include SPI, I2C, and UART.

Subject to the specific hardware characteristics of each IO pin, each GPIO port can be configured in software in several modes:

- Input floating
- Input pull-up
- Input pull-down
- Analog
- Output open-drain with pull-up or pull-down
- Output push-pull with pull-up or pull-down
- Alternative function push-pull with pull-up or pull-down
- Alternative function open-drain with pull-up or pull-down

To ensure that the hardware selected could provide enough functionality for a working prototype, we planned out what peripherals would be needed and configured them before system integration. Additionally, this allowed us to plan out where we wanted components relative to the MCU to make routing and placement easier. The audio processing subsystem makes use of several alternative functions including I2S, SPI, I2C and UART for communicating with external peripherals. This is shown graphically below.

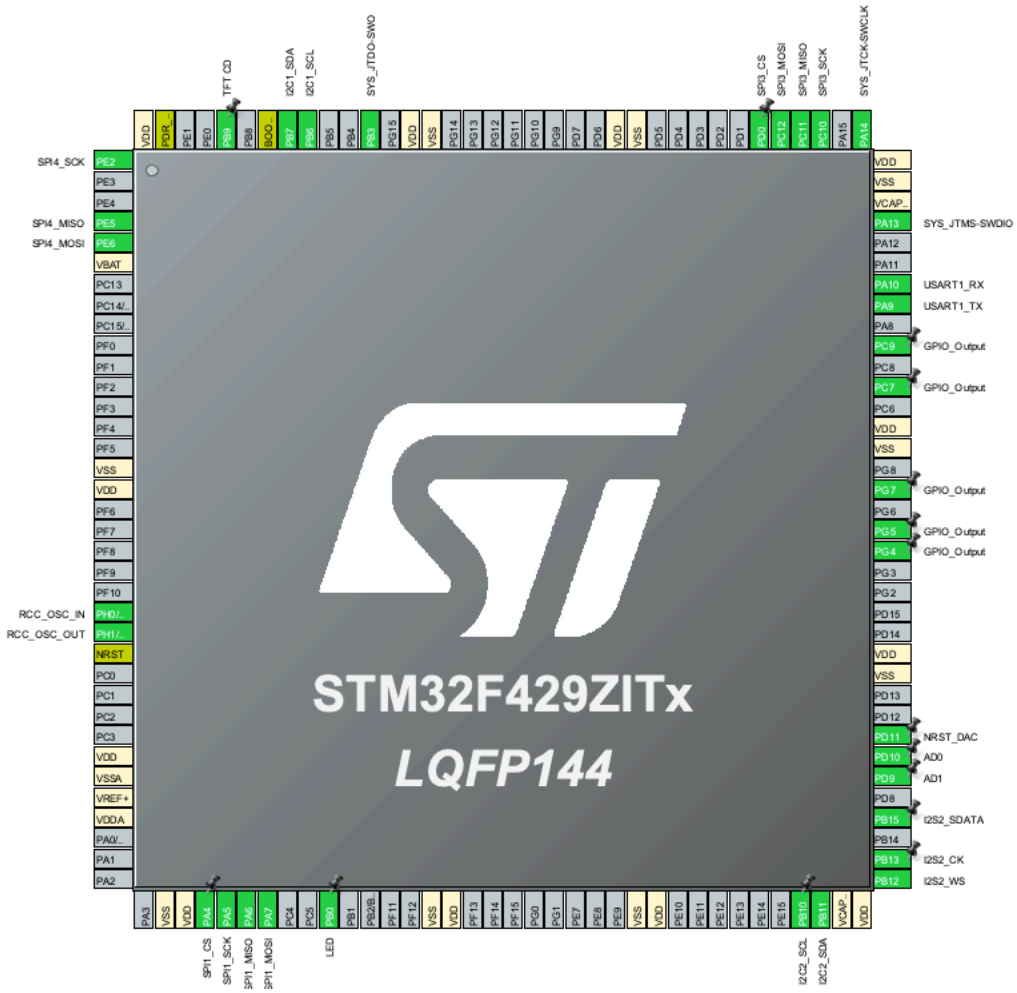


Figure 24: STM32 IO Pin Map

Table 6: Designated Pin Table, provides more detail on how these pins are configured. In addition to the alternative functions presented, additional IO was added including control signals for the LCD, DAC, and some general-purpose IO if needed.

Table 4: Designated Pin Table

Pin Number	Function	Configuration
PB6	I2C1_SCL	Open-drain pull-up
PB7	I2C1_SDA	Open-drain pull-up
PD0	SPI3_CS	Output push-pull no pull-up and no pull-down
PC12	SPI3_MOSI	Output push-pull no pull-up and no pull-down
PC11	SPI3_MISO	Input push-pull no pull-up and no pull-down
PC10	SPI3_SCK	Output push-pull no pull-up and no pull-down
PA9	USART1_TX	Output push-pull no pull-up and no pull-down
PA10	USART1_RX	Input push-pull no pull-up and no pull-down
PB15	I2S2_SDATA	Output push-pull no pull-up and no pull-down
PB13	I2S2_SCK	Output push-pull no pull-up and no pull-down
PB12	I2S2_WS	Output push-pull no pull-up and no pull-down
PB10	I2C2_SCL	Open-drain pull-up
PB11	I2C2_SDA	Open-drain pull-up
PA4	SPI1_CS	Output push-pull no pull-up and no pull-down
PA5	SPI1_SCK	Output push-pull no pull-up and no pull-down
PA6	SPI1_MISO	Input push-pull no pull-up and no pull-down
PA7	SPI1_MOSI	Output push-pull no pull-up and no pull-down
PD11	NRST_DAC	Output push-pull no pull-up and no pull-down
PD10	DAC CTRL0	Output push-pull no pull-up and no pull-down
PD9	DAC CTRL1	Output push-pull no pull-up and no pull-down
PB9	TFT CONTROL/DATA	Output push-pull no pull-up and no pull-down

PC9	SPARE	N/A
PC7	SPARE	N/A
PG7	SPARE	N/A
PG5	SPARE	N/A
PG4	SPARE	N/A

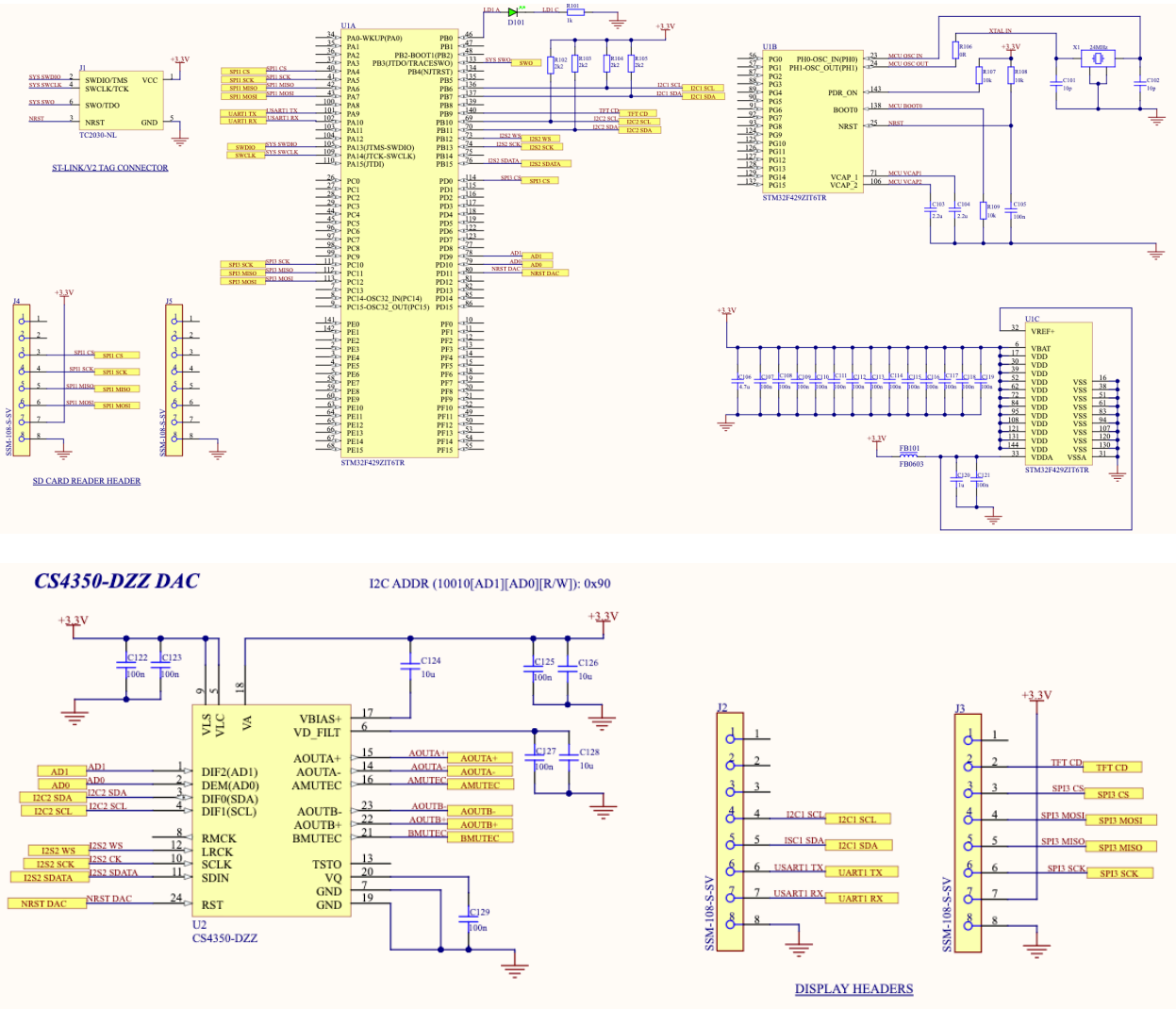


Figure 25: MCU & DAC Schematic

The microcontroller circuitry was designed with system integration in mind. The above schematic includes decoupling capacitors to mitigate voltage fluctuations and noise, clock management circuitry, GPIO expansion headers, debugging headers, and other circuits to ensure reliable operation of the system. These circuits are covered in more depth in the *System Integration Report* included in this document.

4.2.1.2. Configuring DMA

Direct memory access (DMA) is used to provide high-speed data transfer between memory and peripherals without involving the CPU, allowing the CPU to attend to other tasks. For audio and graphics purposes, DMA is a requirement because these processes are resource intensive, and a real-time system cannot afford to be blocked for the entire length of a data transfer.

For our audio player, DMA is used for both graphics and audio streaming. Our audio player uses a ping-pong buffer whereby DMA is used to continuously send audio data to a DAC. The audio data is read into half of a buffer, allowing the other half of the buffer to be processed and sent to the DAC using DMA. The core alternates between the left and right halves of the ping-pong buffer using interrupts triggered whenever half of the buffer gets filled. The below c code demonstrates the initialization of the DMA stream used to transfer the I2S audio data to the DAC. This process is displayed below.

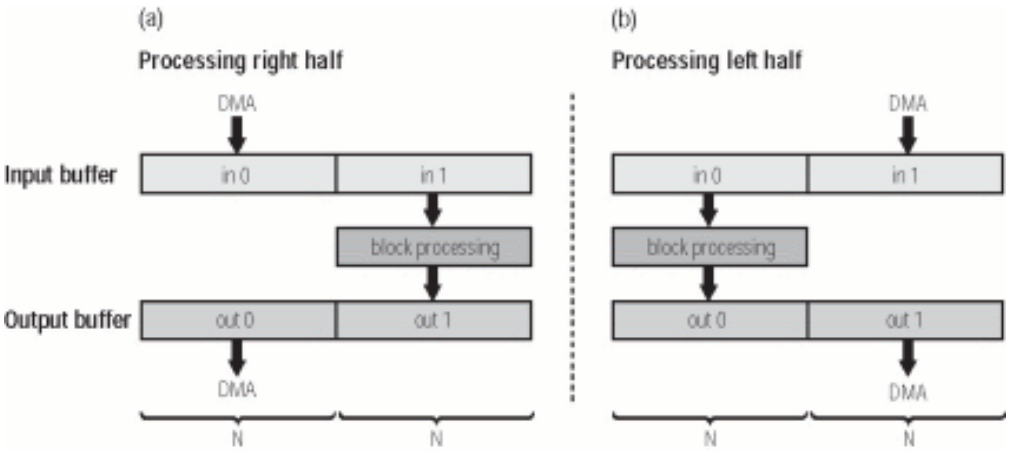


Figure 26: DMA Transfer of I2S Data:
https://audiodsplab.wordpress.com/ping-pong-buffer-audio-stream/

4.2.2. Software

The software of the digital audio player is built using ST's STM32CubeIDE which is a full-fledged Eclipse-based IDE. ST also provides hardware abstraction layer (HAL) generic APIs for interfacing peripherals with the higher-level application layer. This project is entirely written in C and C++ and optimized using Arm GNU build tools. The application uses FatFs- a generic FAT filesystem module for embedded systems, and TouchGFX- a C++ UI framework that drives UI applications and handles user input.

4.2.2.1. Driving the Hardware

Low level peripheral integration is implemented through device drivers. All external peripherals require a driver in order to interface with higher-level modules. This project incorporates four drivers for interacting with the CS4350 DAC, SD card, ILI9341 LCD, and FT6206 touch controller. The CS4350 and SD card drivers are covered here. The other drivers associated with the LCD display are covered in section ____.

4.2.2.1.1 CS4350 Driver

The CS4350 DAC is a 192 kHz stereo DAC with an integrated PLL. In order to use the DAC, we need configure it for use by writing to its control registers. Compared to other stereo DACs, the CS4350 has only a few control registers which are summarized below.

Addr	Function	7	6	5	4	3	2	1	0
1h	Device and RevID	DeviceID4	DeviceID3	DeviceID2	DeviceID1	DeviceID0	RevID2	RevID1	RevID0
	default	1	1	1	1	-	-	-	-
2h	Mode Control	Reserved	DIF2	DIF1	DIF0	DEM1	DEM0	FM1	FM0
	default	0	0	0	0	0	0	0	0
3h	Volume, Mixing, and Inversion Control	VOLB=A	INVERTA	INVERTB	Reserved	ATAPI3	ATAPI2	ATAPI1	ATAPI0
	default	0	0	0	0	1	0	0	1
4h	Mute Control	AMUTE	Reserved	MUTEC A=B	MUTE_A	MUTE_B	Reserved	Reserved	Reserved
	default	1	0	0	0	0	0	0	1
5h	Channel A Volume Control	VOL7	VOL6	VOL5	VOL4	VOL3	VOL2	VOL1	VOL0
	default	0	0	0	0	0	0	0	0
6h	Channel B Volume Control	VOL7	VOL6	VOL5	VOL4	VOL3	VOL2	VOL1	VOL0
	default	0	0	0	0	0	0	0	0
7h	Ramp and Filter Control	SZC1	SZC0	RMP_UP	RMP_DN	Reserved	FILT_SEL	Reserved	Reserved
	default	1	0	1	1	0	0	0	1
8h	Misc. Control	PDN	Reserved	FREEZE	POPG_EN	RMCK_ CTRL1	RMCK_ CTRL0	R_ SELECT1	R_ SELECT0
	default	0	0	0	1	0	0	0	0

Figure 27: CS4350 Control Register Map

Writing/reading to/from the control registers are implemented using the `CS4350_WriteRegister` and `CS4350_ReadRegister` functions which are wrappers for the `HAL_I2C_Mem_Write` and `HAL_I2C_Mem_Read` functions. As indicated by the function names, I2C is used for writing to the control registers. Both the functions used for writing and reading take a pointer to a `CS4350 codec` instance, a register address, and a value to be written (or a reference to a variable to be read into if reading). These wrapper functions are displayed below.

```
HAL_StatusTypeDef CS4350_ReadRegister(CS4350* codec, uint16_t regAddr, uint8_t* data)
{
	HAL_StatusTypeDef status = HAL_I2C_Mem_Read(codec->I2CHandle, CS4350_I2C_ADDR | 0x01,
regAddr, I2C_MEMADD_SIZE_8BIT, data, 1, HAL_MAX_DELAY);

	return status;
}

HAL_StatusTypeDef CS4350_WriteRegister(CS4350* codec, uint16_t regAddr, uint8_t* data)
{
	HAL_StatusTypeDef status = HAL_I2C_Mem_Write(codec->I2CHandle, CS4350_I2C_ADDR, regAddr,
I2C_MEMADD_SIZE_8BIT, data, 1, HAL_MAX_DELAY);

	return status;
}
```

The previously mentioned `CS4350 codec` is simply a typedef'd `struct` which contains an I2C handle, a GPIO port value, and a GPIO pin value. The GPIO port and pin information are used for resetting the device.

The complete `CS4350_Init` function is displayed below.

```
uint8_t CS4350_Init(CS4350* codec, I2C_HandleTypeDef* I2Chandle, GPIO_TypeDef* nrstPinBank,
uint16_t nrstPin)
{
	HAL_StatusTypeDef status;
	uint8_t data;
	uint8_t read;
	char temp[50];

	/* Store I2C handle and reset pin location */
	codec->I2CHandle = I2Chandle;
	codec->nrstPinBank = nrstPinBank;
	codec->nrstPin = nrstPin;

	/* Reset codec */
	CS4350_Reset(codec);

	/* Mode Control (pg.29)*/
	data = 0x95;
	status = CS4350_WriteRegister(codec, MODE_CTRL, &data);
	if (status != HAL_OK)
	{
		printf("Unable to write to register: MODE_CTRL!\r\n");
	}
	status = CS4350_ReadRegister(codec, MODE_CTRL, &read);
	if (status != HAL_OK)
	{
		printf("Unable to read register: MODE_CTRL!\r\n");
	}
	else
	{
		sprintf(temp, "MODE_CTRL: %x\r\n", read);
		printf(temp);
	}

	/* Volume Mixing and Inversion Control */
	data = 0x89;
	status = CS4350_WriteRegister(codec, VOLUME_MIXING_CTRL, &data);
	if (status != HAL_OK)
	{
		printf("Unable to write to register: VOLUME_MIXING_CTR!\r\n");
	}
	status = CS4350_ReadRegister(codec, VOLUME_MIXING_CTRL, &read);
	if (status != HAL_OK)
```

```
        {
                printf("Unable to read register: VOLUME_MIXING_CTRL!\r\n");
        }
        else
        {
                sprintf(temp, "VOLUME_MIXING_CTRL: %x\r\n", read);
                printf(temp);
        }

        /* Set volume */
        data = 240;
        status = CS4350_WriteRegister(codec, CHANNELA_VOL_CTRL, &data);
        if (status != HAL_OK)
        {
                printf("Unable to write to register: CHANNELA_VOL_CTRL!\r\n");
        }
        status = CS4350_ReadRegister(codec, CHANNELA_VOL_CTRL, &read);
        if (status != HAL_OK)
        {
                printf("Unable to read register: CHANNELA_VOL_CTRL!\r\n");
        }
        printf("CHANNELA_VOL_CTRL: %d\r\n", read);

        status = CS4350_WriteRegister(codec, CHANNELB_VOL_CTRL, &data);
        if (status != HAL_OK)
        {
                printf("Unable to write to register: CHANNELB_VOL_CTRL!\r\n");
        }
        status = CS4350_ReadRegister(codec, CHANNELB_VOL_CTRL, &read);
        if (status != HAL_OK)
        {
                printf("Unable to read register: CHANNELB_VOL_CTRL!\r\n");
        }

}
```

4.2.2.1.2 SD Card Driver

The SD card driver is implemented in `fatfs_sd.c`. Unlike the previous driver for the DAC, the SD card driver uses SPI to send commands to the SD card. This driver interfaces with the low-level diskio layer which is implemented in `user_diskio.c` and has an interface that resembles the API used in most Unix systems. The exact interface used is detailed below.

`DSTATUS SD_disk_initialize(BYTE pdrv)`

- *Brief:* This function initializes the SD card for use by higher-level modules.
- *Parameters:*
  - `pdrv`: Physical drive number to operate on.

`DRESULT SD_disk_read(BYTE pdrv, BYTE* buff, DWORD sector, UINT count)`

- *Brief:* This function performs block reads.
- *Parameters:*
  - `pdrv`: Physical drive number to operate on.
  - `buff`: Buffer where read data is to be stored.
  - `sector`: Sector number to start read operation.
  - `count`: Number of sectors to read.

DRESULT SD_disk_write (BYTE pdrv, const BYTE* buff, DWORD sector, UINT count)

- *Brief:* This function performs block writes.
- *Parameters:*
  - `pdrv`: Physical drive number to operate on.
  - `buff`: Buffer containing data to be written.
  - `sector`: Sector number to start read operation.
  - `count`: Number of sectors to read.

DRESULT SD_disk_ioctl (BYTE pdrv, BYTE cmd, void* buff)

- *Brief:* This function performs block writes.
- *Parameters:*
  - `pdrv`: Physical drive number to operate on.
  - `buff`: Buffer where read data will be stored.
  - `cmd`: Control command.

4.2.2.2. I2S Buffer Implementation

As previously mentioned, audio streaming is facilitated using DMA and interrupts to offload work from the CPU. A file begins playing with a call to `player_play` which takes a handle to an I2S instance as input. The function fills the first half of the buffer to be transferred to the DAC and initiates the non-blocking DMA transfer by calling `HAL_I2S_TRANSMIT_DMA`.

```
void player_play(I2S_HandleTypeDef* hi2s)
{
        /* fill first half of buffer */
        f_read(&wavfile, &dac_buffer[0], AUDIO_BUFFER_SIZE, &bytes_read);
        bytes_remaining -= bytes_read;

        /* Begin circular DMA */
        status = HAL_I2S_Transmit_DMA(hi2s, (uint16_t*) dac_buffer, AUDIO_BUFFER_SIZE/2);
}
```

This function takes the I2S handle, the transmit buffer, and the number of 16-bit samples as input parameters.

Whenever the left/right half of the buffer gets filled, an interrupt is triggered which calls either **`HAL_I2S_TxHalfCpltCallback`** or **`HAL_I2S_TxCpltCallback`**. These callback functions simply change a state variable **`player_control_sm`**.

```
/* This function is called when the first half of the buffer has been transferred */
void HAL_I2S_TxHalfCpltCallback(I2S_HandleTypeDef* hi2s)
{
        player_control_sm = PLAYER_CONTROL_HALFBUFFER;
}

/* This function is called when the second half of the buffer has been transferred */
void HAL_I2S_TxCpltCallback(I2S_HandleTypeDef* hi2s)
{
        player_control_sm = PLAYER_CONTROL_FULLBUFFER;
}
```

The state variable **player_control_sm** is an enum that represents the current state of the buffer.

```
/* WAV player buffer states */
typedef enum
{
  PLAYER_CONTROL_IDLE = 0,
  PLAYER_CONTROL_HALFBUFFER,
  PLAYER_CONTROL_FULLBUFFER,
  PLAYER_CONTROL_EOF,
}PlayerControlTypeDef;
```

The heart of the buffer processing is implemented in the `player_process` function shown below.

```
void player_process(void)
{
	switch (player_control_sm)
	{
		case PLAYER_CONTROL_IDLE :
			break;

		case PLAYER_CONTROL_HALFBUFFER :
			bytes_read = 0;
			/* Fill first half of buffer */
			f_read(&wavfile, &dac_buffer[0], AUDIO_BUFFER_SIZE, &bytes_read);
			if (bytes_remaining > AUDIO_BUFFER_SIZE)
			{
				bytes_remaining -= bytes_read;
			}
			else
			{
				bytes_remaining = 0;
				player_control_sm = PLAYER_CONTROL_EOF;
			}
			player_control_sm = PLAYER_CONTROL_IDLE;
			break;

		case PLAYER_CONTROL_FULLBUFFER :
			bytes_read = 0;
			/* Fill second half of buffer */
			f_read(&wavfile, &dac_buffer[AUDIO_BUFFER_SIZE/2], AUDIO_BUFFER_SIZE,
&bytes_read);
			if (bytes_remaining > AUDIO_BUFFER_SIZE)
			{
				bytes_remaining -= bytes_read;
			}
			else
			{
				bytes_remaining = 0;
				player_control_sm = PLAYER_CONTROL_EOF;
			}
			player_control_sm = PLAYER_CONTROL_IDLE;
			break;

		case PLAYER_CONTROL_EOF :
			player_stop(&hi2s2);
			f_close(&wavfile);
			player_reset();
			wavfile_isfinished = true;
			player_control_sm = PLAYER_CONTROL_IDLE;
			break;
	}
}
```

Inside the switch-case statement, the `PLAYER_CONTROL_HALFBUFFER` the `PLAYER_CONTROL_FULLBUFFER` blocks read from the audio file and fill either the first half or second half of the buffer depending on the state variable. This is done using the FatFs function `f_read` which takes as input a file pointer, a buffer to fill, and the number of bytes to read. The number of bytes to read is equal to the number of 16-bit samples because we only fill half of the buffer at a given time. After a read takes place, bytes remaining is decremented if the file still contains data that can be buffered. If the remaining bytes is less than the size of the buffer, the entire file has been read and the player can be reset.

4.2.2.3. Music Library

The music library is implemented as a doubly linked list of `Song` objects. `Song` objects are simply a datatype that includes a `const char` path and a pointer to the next and previous items in the linked list. This implementation is effective because it allows songs to be dynamically allocated at runtime and reallocation of songs is more efficient than if the library was implemented using an array.

At runtime, the library is initialized with a call to `initialize_library`. This function uses FatFs utilities to locate songs withing the /library directory with a .wav extension and adds them to the library using the `add_song` function. The `add_song` function just allocates memory for the song using the `create_song` function and adds the song to the end of the linked list. These functions are shown below.

```
Song* initialize_library()
{
	/* FatFs private variables */
	FRESULT fres;
	static DIR dir;
	static FILINFO file_info;
	char temp[50];
//	static const char* path;

	static Song* library = NULL;

	fres = f_mkdir("library");
	fres = f_opendir(&dir, "library");
	sprintf(temp, "FRES opendir ERROR: %d in initialize_library()\r\n", fres);
	printf(temp);
	if (fres == FR_OK)
	{
		printf("Successfully opened Library!\r\n");
		while (1)
		{
			/* Read a directory item */
			fres = f_readdir(&dir, &file_info);
			if (fres != FR_OK || file_info.fname[0] == 0) break; // End of directory

			if (!(file_info.fattrib & AM_DIR))
			{
				// Check if the file has a ".wav" extension
				if (strstr(file_info.fname, ".wav") != NULL)
				{
					printf("Found a .wav\r\n");
					char path_str[100] = "library/";
					add_song(&library, strcat(path_str, file_info.fname));
				}
			}
		}
	}
```

```
        return library;
}

// Function to create a new song node
Song* create_song(const char* path)
{
    Song* newSong = (Song*)malloc(sizeof(Song));
    if (newSong == NULL) {
        exit(1);
    }
    strncpy(newSong->path, path, sizeof(newSong->path));
    newSong->next = NULL;
    newSong->prev = NULL;

    return newSong;
}

// Function to insert a song at the end of the list
void add_song(Song** head, const char* path)
{
    Song* newSong = create_song(path);
    if (*head == NULL)
    {
        *head = newSong;
    }
    else
    {
                Song* current = *head;
                while (current->next != NULL)
                {
                        current = current->next;
                }
        current->next = newSong;
        newSong->prev = current;
    }
}

// Function to free the memory of the double linked list
void free_songs(struct Song* head)
{
    Song* current = head;
    while (current != NULL)
    {
        Song* next = current->next;
        free(current);
        current = next;
    }
}
```

4.2.2.1. User Interface

The user interface is implemented using the TouchGFX Designer software- a graphical application that allows a UI designer to create a UI from a selection of configurable components such as buttons and widgets. This tool integrates seamlessly with the STM32CubeIDE previously mentioned in the introduction of this section; the UI files just need to be included in the project folder and then they build automatically.

Due to the breadth of the UI application, the user interface deserves its own section and is presented in section ___ of this document.

4.3. Subsystem Validation

The audio processing subsystem was validated using the requirements set forth in the *Validation Plan* in this document. Each validation test is meant to identify key system deficiencies and improve system performance and robustness. The following subsections present the results of each test.

4.3.1. Internal Communications

In order to validate that internal communications were working as expected, an oscilloscope was hooked up to each of the peripheral pins and the waveforms were evaluated. Additionally, several audio files were loaded into the SD card and played to determine if the internal communications were working properly.

4.3.1.1. I2C

I2C uses an open-drain bus which requires external pull-up resistors to passively pull the line high. Typical values range from 1 k Ω to 10 k Ω. The exact value of resistor selected effects the rise time of the signal. The DAC requires a rise time on the SDA and SCL lines of at most 1 μ s to function properly. The figure shown below demonstrates a SCL line that doesn't meet this requirement.

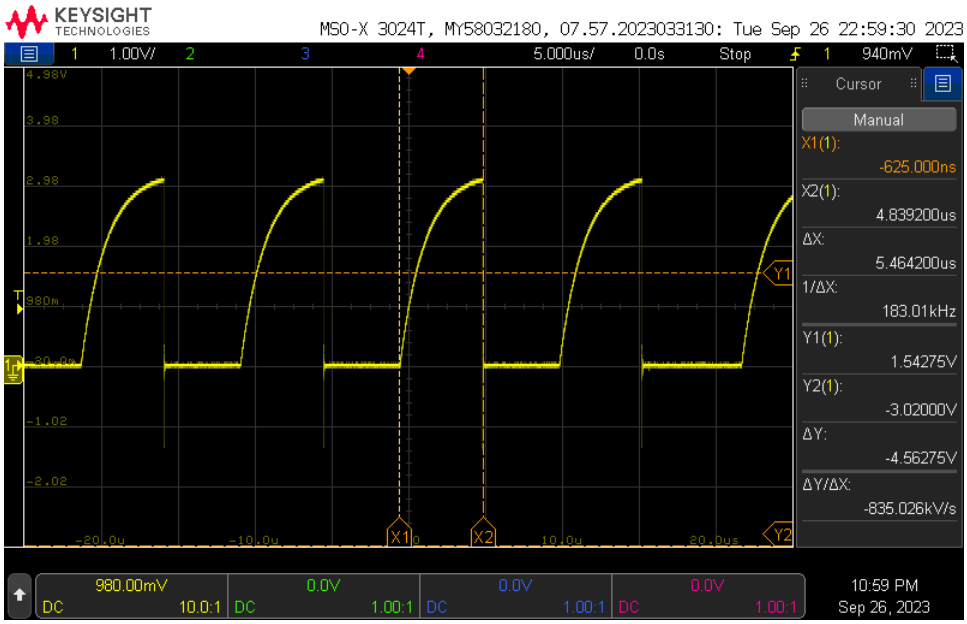


Figure 28: I2C SCL line With Slow Rise Time

By selecting a lower value (stronger pull-up) resistor, I was able to achieve the correct rise time that allowed I2C to properly write to the DAC control registers. Additional I2C validation results are presented in *Section 4.3.3.1 I2C Register Read/Write.*

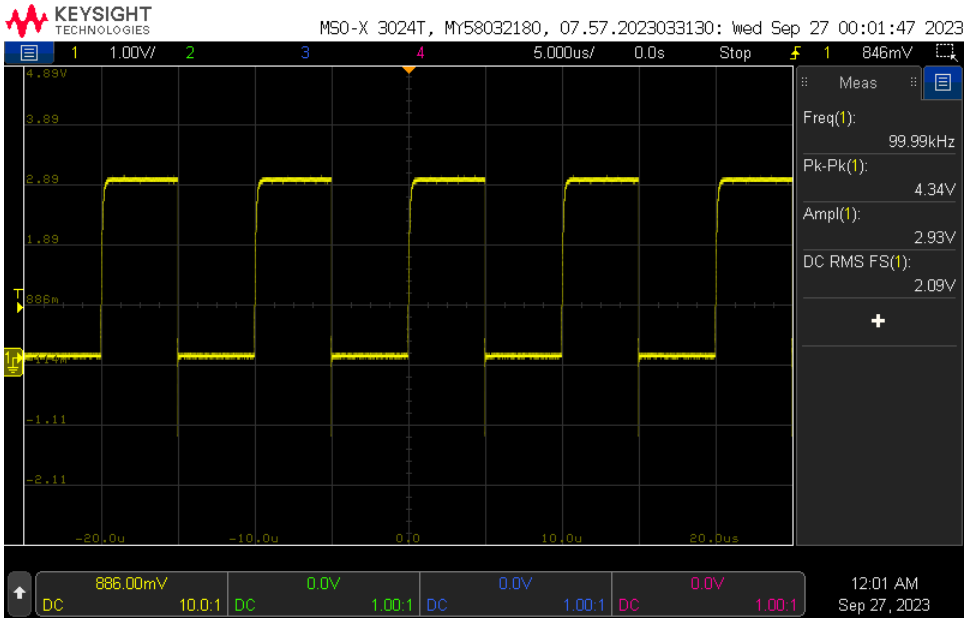


Figure 29: I2C SCL Line with Proper Rise Time

4.3.1.2. I2S

I2S is a 3-wire bus (sometimes 4-wire) that consists of LRCLK, SCK and SDATA clock and data lines. Given that our audio has a bit depth of 16 bits and is sampled at 44.1 kHz, the expected LRCLK and SCK is 44.1 kHz and 1.41 MHz respectively. LRCLK is 44.1 kHz because one left sample is clocked on the rising edge of the clock while the right sample is clocked on the falling edge. The equation for SCK frequency is shown below.

$$\text{SCK: } 16\ bits \times 2\ channels \times 44.1\ ksamples = 1.41\ MHz$$

The figure below shows the results of the LRCLK clock line. The achieved clock has a frequency 44.66 kHz which is close enough to the expected 44.1 kHz. The difference in frequency is likely attributed to the fact that an internal PLL is used to approximate this signal frequency.

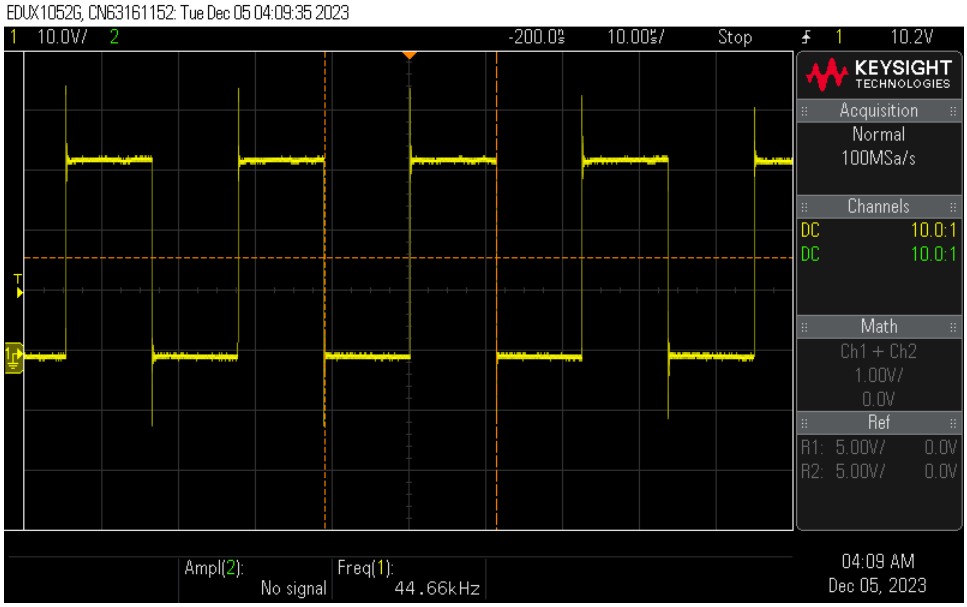


Figure 30: I2S LRCLK Clock

The next signal that was tested was the serial clock or SCK. This signal is used to clock each bit and should have a frequency of 1.41 MHz. The actual achieved frequency is 1.429 MHz which can again be attributed to the imperfect generation using a PLL to approximate this frequency.

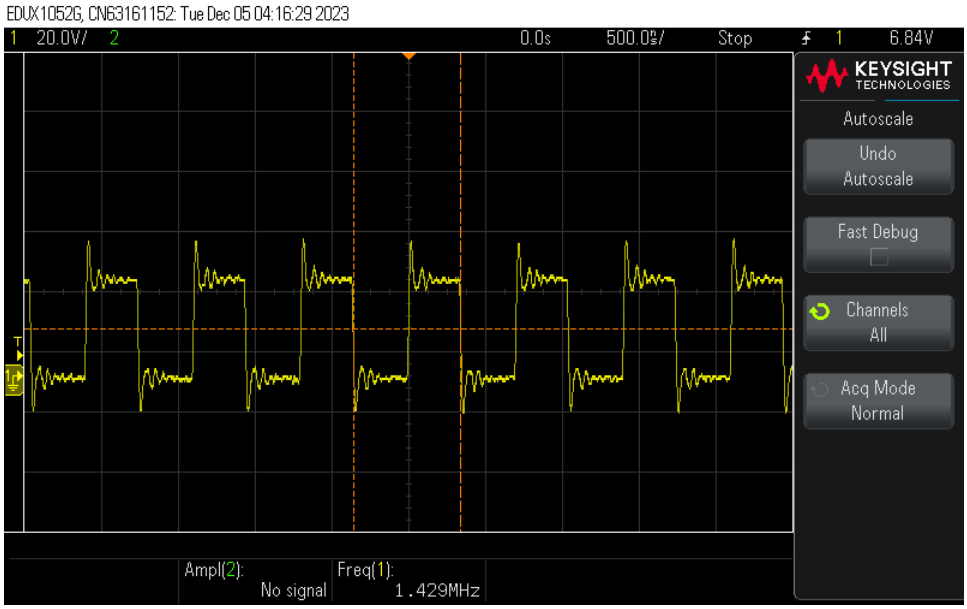


Figure 31: I2S SCK Clock

The last signal that needed to be validated was the serial data line or SDATA. This signal alternates between left and right 16-bit data samples. This signal was validated by using a logic analyzer and comparing the 16-bit samples with the hex code present in the wav file used for testing. Additional validation was done by examining the output of the DAC to see if the expected waveform was present. See *Section 4.3.3.2 DAC Sine Output.*

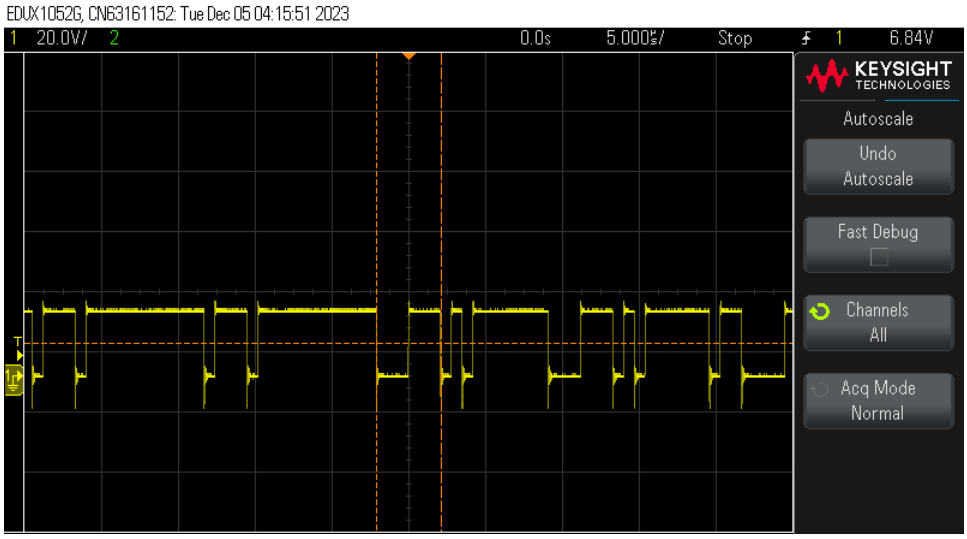


Figure 32: I2S SDATA Test

4.3.1.3. SPI

In the microcontroller subsystem, SPI is used to read data from the SD card. The SPI peripheral was configured to a baud rate of 3 MHz within STM32CubeIDE. The expected serial clock (SCLK) line should match this frequency. The resulting clock signal has a frequency of 3.1 MHz.The difference in frequency is likely attributed to the internal PLL used to approximate the desired SPI frequency.

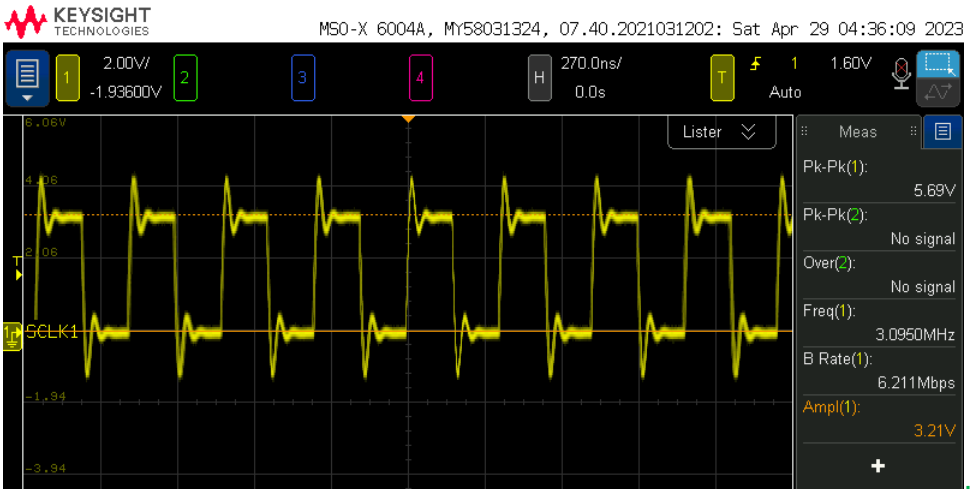


Figure 33: SPI SCLK Signal

In addition to the SCLK line, the MISO (Master-in Slave-out) line was measured. This signal was validated by using a logic analyzer and comparing the 16-bit samples with the hex code present in the wav file used for testing.

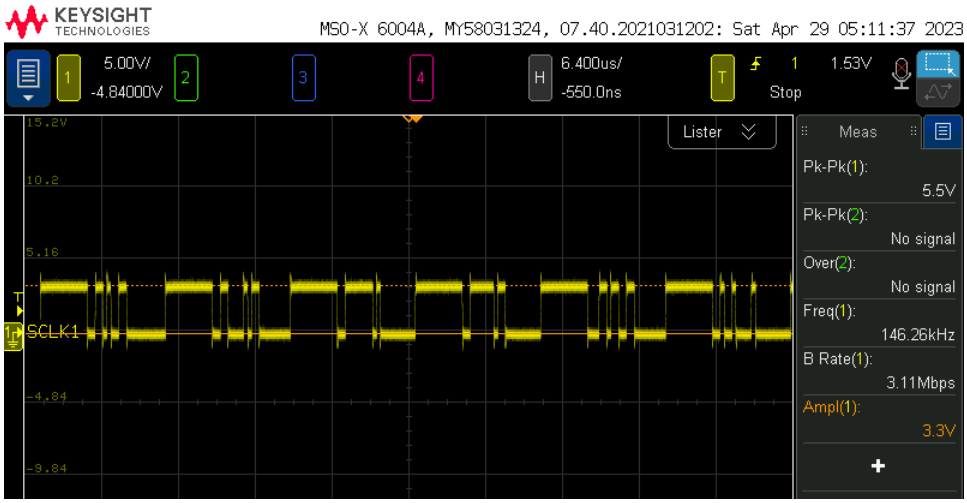


Figure 34: SPI MISO Signal

4.3.2. FatFs Read and Library Initialization

This section includes a unit test that was performed on that `initialize_library` and `player_song_select` functions. The objective of this test was to validate the files could be read and written using the FatFs API.

The library is first initialized with a call to `initialize_library`. This function uses FatFs utilities to locate songs withing the /library directory with a .wav extension and adds them to the library using the `add_song` function. The `add_song` function just allocates memory for the song using the `create_song` function and adds the song to the end of the linked list. These functions are shown below.

```
Song* initialize_library()
{
        /* FatFs private variables */
        FRESULT fres;
        static DIR dir;
        static FILINFO file_info;
        char temp[50];
//      static const char* path;

        static Song* library = NULL;

        fres = f_mkdir("library");
        fres = f_opendir(&dir, "library");
        sprintf(temp, "FRES opendir ERROR: %d in initialize_library()\r\n", fres);
        printf(temp);
        if (fres == FR_OK)
        {
                printf("Successfully opened Library!\r\n");
                while (1)
```

```
                {
                        /* Read a directory item */
                        fres = f_readdir(&dir, &file_info);
                        if (fres != FR_OK || file_info.fname[0] == 0) break; // End of directory

                        if (!(file_info.fattrib & AM_DIR))
                        {
                                // Check if the file has a ".wav" extension
                                if (strstr(file_info.fname, ".wav") != NULL)
                                {
                                        printf("Found a .wav\r\n");
                                        char path_str[100] = "library/";
                                        add_song(&library, strcat(path_str, file_info.fname));
                                }
                        }
                }
        }
        return library;
}
```

Then `player_song_select` selects a song for playback and prints its path to the console.

```
void player_song_select(Song* song)
{
        WavHeaderTypeDef wav_header;
        uint32_t extracted_data;
    uint8_t* data_ptr = NULL;
    static char path[50];
    strcpy(path, song->path);
    printf("Path: %s\r\n", path);

//      assert(f_open(&wavfile, (const TCHAR*) song->path, FA_OPEN_EXISTING | FA_READ) == FR_OK
&& "Error opening file!\r\n");
//      assert(f_read(&wavfile, wav_header.wav_header_buf, 44, &bytes_read) == FR_OK && "Error
reading wav header!\r\n");
    fres = f_open(&wavfile, (const TCHAR*) path, FA_READ);
//    if (f_open(&wavfile, (const TCHAR*) path, FA_READ) != FR_OK) return;
    fres = f_read(&wavfile, wav_header.wav_header_buf, 44, &bytes_read);
//    if (f_read(&wavfile, wav_header.wav_header_buf, 44, &bytes_read) != FR_OK) return;
        data_ptr = &wav_header.wav_header_buf[4];
        memcpy(&extracted_data, data_ptr, sizeof(uint32_t));

        wav_header.file_size = extracted_data;
        file_length = wav_header.file_size;
        bytes_remaining = file_length - bytes_read;
}
```

The outputs of the `printf` statements were printed on the data console. The results indicate that the library can be initialized correctly and that wav files are able to be read from.

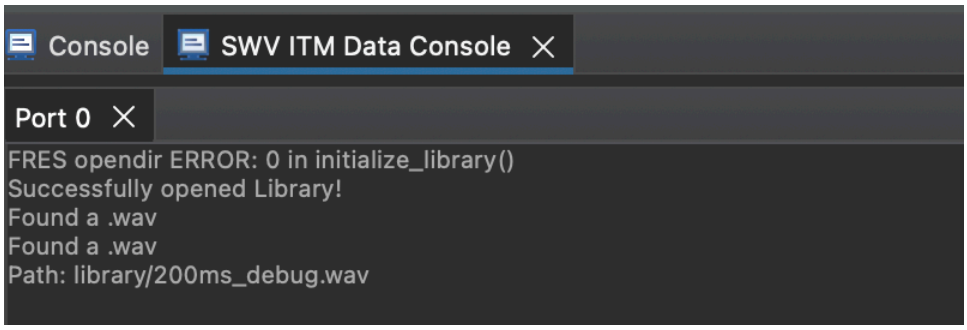


Figure 35: FatFs Read and Library Initialization Results

4.3.3. MCU/DAC Integration

This section demonstrates integration testing completed on the microcontroller and DAC. This test validates that wav files can be selected, read from, and buffered to the DAC. It also validates that the DAC can be configured properly by writing to its internal registers using I2C.

4.3.3.1. I2C Register Read/Write

The first module under test is the `CS4350_Init` function located in `CS4350.c`. This function initializes the DAC by writing to its internal registers using I2C. The code snippet below displays conditional statements that execute if I2C transfers are successful. The `printf` statements then display the value that was read from the internal registers.

```
/* Mode Control (pg.29)*/
	data = 0x95;
	status = CS4350_WriteRegister(codec, MODE_CTRL, &data);
	if (status != HAL_OK)
	{
		printf("Unable to write to register: MODE_CTRL!\r\n");
	}
	status = CS4350_ReadRegister(codec, MODE_CTRL, &read);
	if (status != HAL_OK)
	{
		printf("Unable to read register: MODE_CTRL!\r\n");
	}
	else
	{
		sprintf(temp, "[Test Passed] MODE_CTRL: %x\r\n", read);
		printf(temp);
	}

	/* Volume Mixing and Inversion Control */
	data = 0x89;
	status = CS4350_WriteRegister(codec, VOLUME_MIXING_CTRL, &data);
	if (status != HAL_OK)
	{
		printf("Unable to write to register: VOLUME_MIXING_CTR!\r\n");
	}
	status = CS4350_ReadRegister(codec, VOLUME_MIXING_CTRL, &read);
	if (status != HAL_OK)
	{
		printf("Unable to read register: VOLUME_MIXING_CTRL!\r\n");
	}
	else
	{
```

```
            sprintf(temp, "[Test Passed] VOLUME_MIXING_CTRL: %x\r\n", read);
            printf(temp);
    }

    /* Set volume */
    data = 240;
    status = CS4350_WriteRegister(codec, CHANNELA_VOL_CTRL, &data);
    if (status != HAL_OK)
    {
            printf("Unable to write to register: CHANNELA_VOL_CTRL!\r\n");
    }
    status = CS4350_ReadRegister(codec, CHANNELA_VOL_CTRL, &read);
    if (status != HAL_OK)
    {
            printf("Unable to read register: CHANNELA_VOL_CTRL!\r\n");
    }
    printf("[Test Passed] CHANNELA_VOL_CTRL: %d\r\n", read);

    status = CS4350_WriteRegister(codec, CHANNELB_VOL_CTRL, &data);
    if (status != HAL_OK)
    {
            printf("Unable to write to register: CHANNELB_VOL_CTRL!\r\n");
    }
    status = CS4350_ReadRegister(codec, CHANNELB_VOL_CTRL, &read);
    if (status != HAL_OK)
    {
            printf("Unable to read register: CHANNELB_VOL_CTRL!\r\n");
    }
```

This unit test displays that the DAC acknowledges the I2C transfers, indicating proper communication between the DAC and microcontroller.

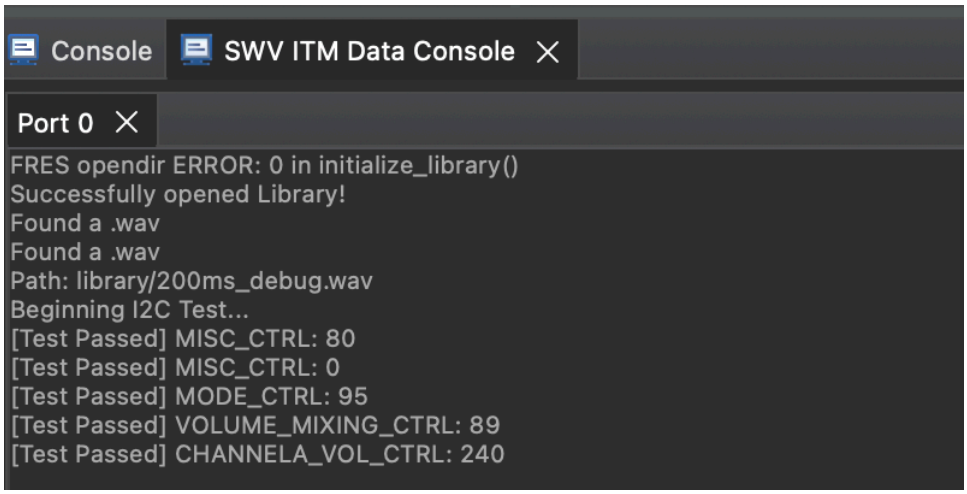


Figure 36: DAC Internal Register R/W Integration Test

4.4. Subsystem Conclusion

For development and validation, the STM32F4 Nucleo Development Board and DAC daughter board were used. A PCB will later replace the development board and integrate with the final power PCB and LCD display module. This implies that the Board Support Package (BSP) will have to be re-written later for integration with a custom PCB. Additionally, the display will need to be integrated with the microcontroller.

5. Mechanical Subsystem

5.1. Subsystem Introduction

The mechanical subsystem is responsible for designing and implementing the physical enclosure that will house all the electronic components. This enclosure not only provides protection to the components but also determines the overall appearance and feel of the device.

5.2. Subsystem Details

The housing for the digital audio player was developed using SolidWorks, a 3D CAD design software. The design process involved strategically placing ledges to provide stable platforms for the power and audio PCBs, ensuring stable connections to the enclosure without any risk of movement. Considerable attention was also given to the functionality of the enclosure, with plenty of space for wiring, easy SD card mount/dismount, and precise openings for the micro-USB, headphone jack, and display. Following the design phase, the housing was brought into physical form through 3D printing. Since the digital audio player is expected to be a portable device, the enclosure was made to the following dimensions: 4in x 6.5in x 1.5in. While these dimensions do not allow for the device to fit in a user's pocket, the device remains conveniently able to be handheld or easily placed in a purse or backpack.

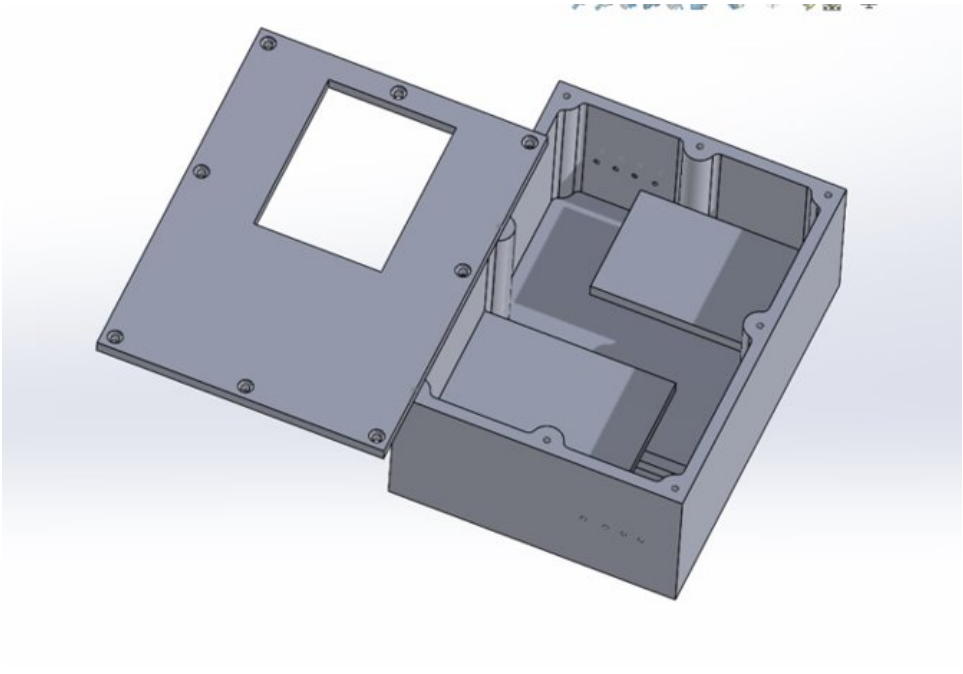


Figure 37: Example of Possible Device Enclosure

When designing the enclosure, three main factors must be considered: heat, moisture, and humidity. To address the heat generated by the device's components, the enclosure will be designed with small holes to allow for airflow. Since this device is a prototype, the device is not immune to moisture and humidity. The device will not be able to go in the water.

5.3. Subsystem Validation

The mechanical subsystem went through the validation process focused on portability. Through multiple iterations of the device enclosure, the final enclosure design allows for user to carry and transport the device using a purse or backpack. This approach ensured that the mechanical subsystem meets the validation expectations, offering a user-friendly and portable digital audio player.

5.4. Subsystem Conclusion

Overall, the mechanical subsystem is responsible for housing all the components associated with the digital audio player. This subsystem is also tasked with being the final stage of subsystem integration. It acts as the centerpiece for the complete working device.

6. User Interface Subsystem

6.1. Subsystem Introduction

This subsystem operates as the interface between the electronic device and the product user. This subsystem was designed using principles of abstraction to simplify its use for nontechnical customers. This system controls the device's operation by interacting, primarily, with the audio processing subsystem. This subsystem displays our product logo, basic audio control buttons such as play, pause, skip, forward, and volume control on a thin film transistor liquid crystal display. The touching of these buttons causes audio processing functions to execute.

6.2. Subsystem Details

6.2.1. Hardware

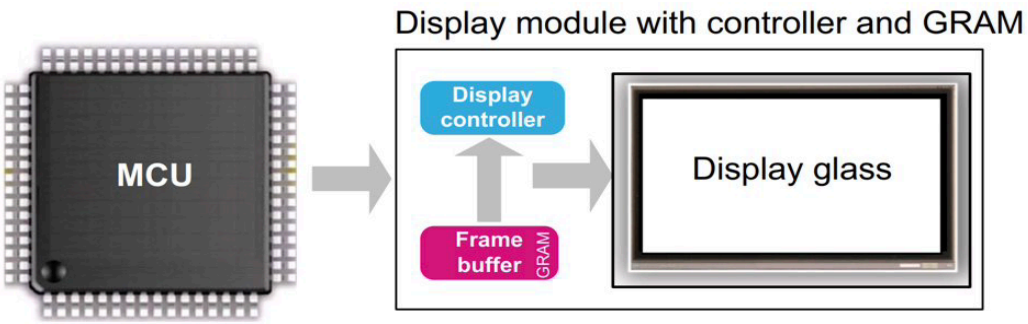


Figure 38: MCU Display Interaction

Our user interface is controlled by a STM32 Nucleo development kit. At the heart of this set up is the STM32F4 microcontroller. This design allows for two way communication from peripheral (LCD) to controller and vice versa. We chose an Adafruit 2.8 inch 240 x 320 TFT liquid crystal display to implement the user interface on. The image display on this LCD is controlled by the ILI9341 display driver and the touch control hardware used in this LCD is the FT62606 touch controller. The image display and transfer is done by direct memory access and peripheral data transfer through the serial peripheral interface (SPI) communication protocol while the touch information is transmitted with the inter-integrated circuit (I2C) communication protocol.

6.2.2. Software

The graphics of our user interface were developed with the STM supplied graphical user interface application: TouchGFX. An ili9341 device driver was used to implement the interfacing of the STM development board with the LCD. This driver source file required

alterations, including the SPI port, to work. The touch control is managed by a similar device driver source file.

```
// The variable for Callback is open. User should set by himself
void ILI9341_DrawBitmap(uint16_t w, uint16_t h, uint8_t *s)
{
        // Enable to access GRAM
        LCD_WR_REG(0x2c);

        DC_H();
#if 0
        __HAL_SPI_DISABLE(&hspi1);
        hspi1.Instance->CR2 |= SPI_DATASIZE_16BIT; // Set 16 bit
        __HAL_SPI_ENABLE(&hspi1);
#endif
        ConvHL(s, (int32_t)w*h*2);
        HAL_SPI_Transmit_DMA(&hspi1, (uint8_t*)s, w * h *2);
#if 0
        __HAL_SPI_DISABLE(&hspi1);
        hspi1.Instance->CR2 &= ~(SPI_DATASIZE_16BIT); // Set 8 b
        __HAL_SPI_ENABLE(&hspi1);
#endif
}

// User should call it at callback
void ILI9341_EndOfDrawBitmap(void)
{
#if 0
        __HAL_SPI_DISABLE(&hspi1);
```

Figure 39: Display Device Driver

The display functions by writing a bitmap from memory to the LCD and communicating in reverse when a particular section of the screen is touched. This touch causes some action, implemented in software, to be executed. The primary challenge from a software perspective was dealing with image alterations and glitching on the display screen. To resolve this issue, I calculated the time that it takes to send one frame buffer from memory to the display. This time was computed from the total pixels, the color scheme (RGB 565 2 bytes per pixel), and the SPI baud rate. This time is approximately 50 milliseconds. This data transfer time was used to select a screen refresh rate at slightly under 20 Hertz. It was discovered that a refresh rate of 17 Hertz was optimal. This control was set by implementing an internal timer. The timer was adjusted by scaling and setting the clocking period. The

other software task in this subsystem involved integration with the audio processing microcontroller. To integrate the two subsystems, we developed a basic parallel data transfer communication protocol. We configured GPIO pins on the audio processing microcontroller as digital inputs while setting corresponding digital output pins on the display microcontroller. On the display side, we programmed the microcontroller to send a digital pulse when a certain button is pressed. The audio processing microcontroller searches for this pulse and calls the corresponding audio function when it determines that a certain button has been pressed via the pulse. The pulse was implemented by global interrupts.

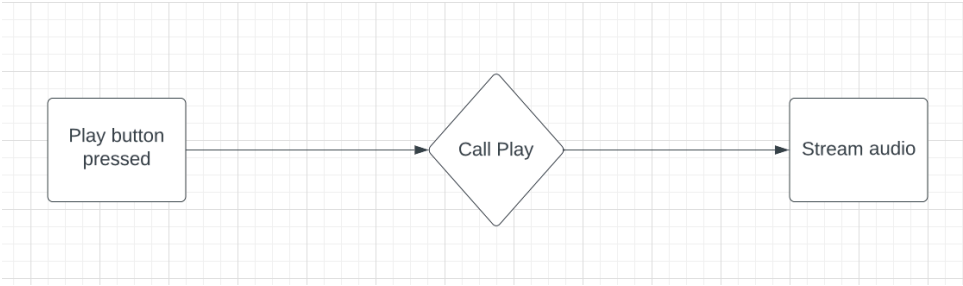


Figure 40: Flowchart of Communication

6.3. Subsystem Validation

This subsystem is primarily visually, and action based. Therefore, validation was tricky as there were few precise measurements to make and test. Validation of this system was primarily completed by observation. For instance, I made sure that the desired buttons and animations appeared, and remained present without glitching, where they were expected to be.

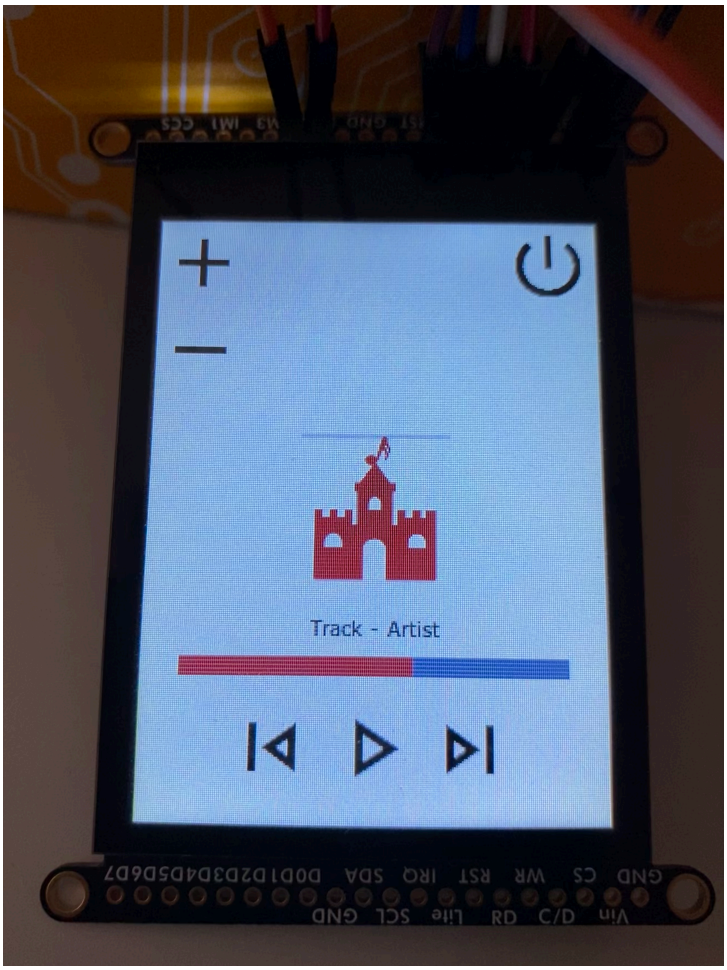


Figure 41: User Interface

Button functionality was validated by setting a GPIO pin to logic “1” and measuring the value with a multimeter. This logic “1” was used to call a function on a separate microcontroller to validate our communication method.

Figure 42: Button Test

Final validation consisted of verifying that the correct audio functions were called, and that the power button controlled the LCD backlight, as desired by observation. The artist's name and current song title display is the only design requirement for this subsystem that was not completed and validated.

6.4. Subsystem Conclusion

The graphical user interface subsystem is at the core of the user experience. It interacts directly with the user to allow control of the overall electronic device. This subsystem passed all validation phases except for displaying the current song title and artist information and functions in the way it was designed to.

Portable Hi-Res Digital Audio Player

Rammi Hameed

Jake Flores

Seth Pregler

Patrick Westmoreland

INTEGRATED SYSTEM REPORT

REVISION – 1.0

02 December 2023

SYSTEM REPORT

FOR

Portable High-Resolution Digital Audio Player

TEAM 25

APPROVED BY:

Patrick Westmoreland

_____

Project Leader Date

_____

Dr. Kevin Nowka Date

_____

T/A Date

Change Record

Rev.	Date	Originator	Approvals	Description
1.0	12/02/2023	Rammi Hameed Jake Flores Seth Pregler PatrickWestmoreland		Original Release

Table of Contents

List of Tables

List of Figures

1. Overview

This documentation presents a detailed account of system integration and validation for the second semester capstone project titled, *Portable High-Resolution Digital Audio Player*. The focus was shifted from individual subsystem design/validation to that of the whole system. Due care was taken to meet the specifications outlined in the *Function System Requirements* document included in this report. The subsequent sections describe major project changes, a project execution plan, system integration, system validation and concluding remarks.

2. Project Changes

Major project changes include the addition of Patrick Westmoreland. Patrick Westmoreland implemented the user interface and worked with Seth Pregler integrating it with the microcontroller. Additionally, other changes include modifications in the scope of the project including, the removal of equalization capabilities.

3. Project Execution

The second semester of senior capstone was divided into three phases, subsystem finalization, system integration, and system validation. To keep track of team progress, a grant chart was created and followed closely.

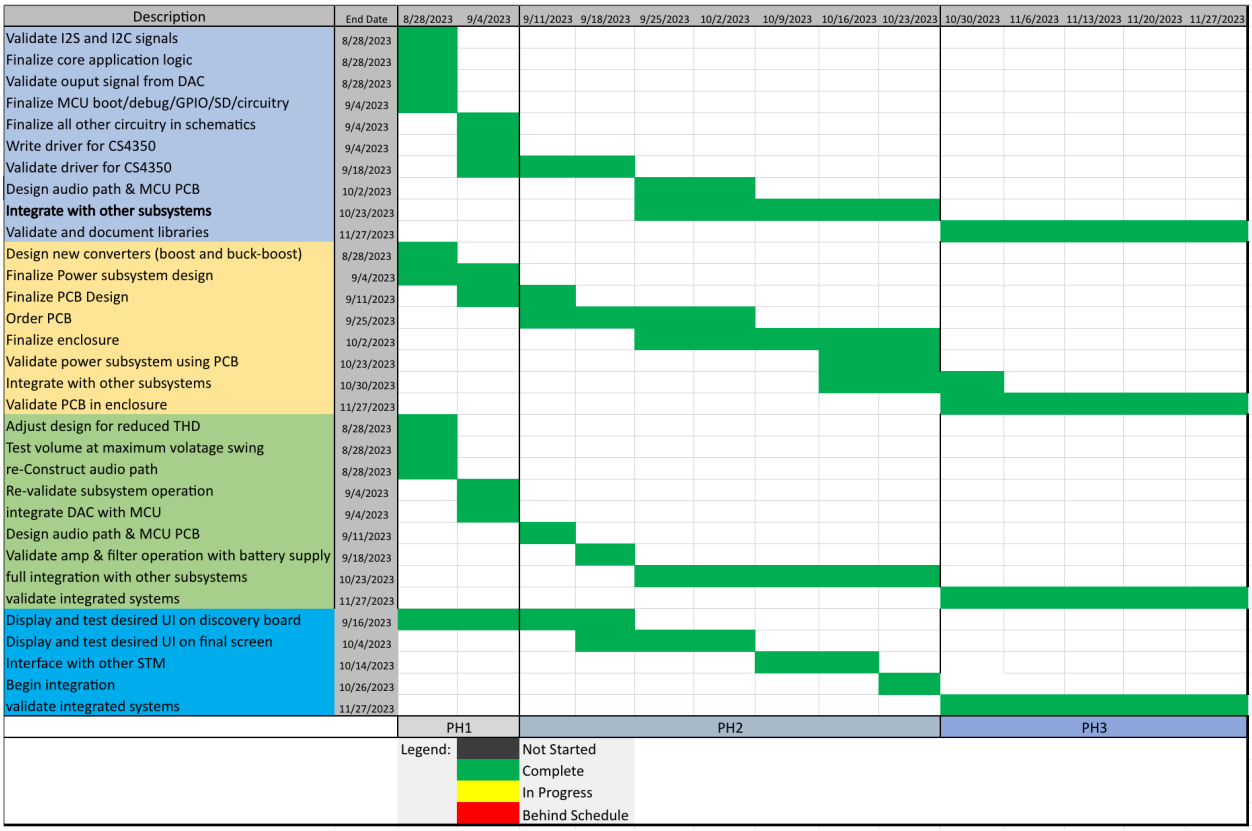


Figure 1: Second Semester Execution Plan

4. System Integration

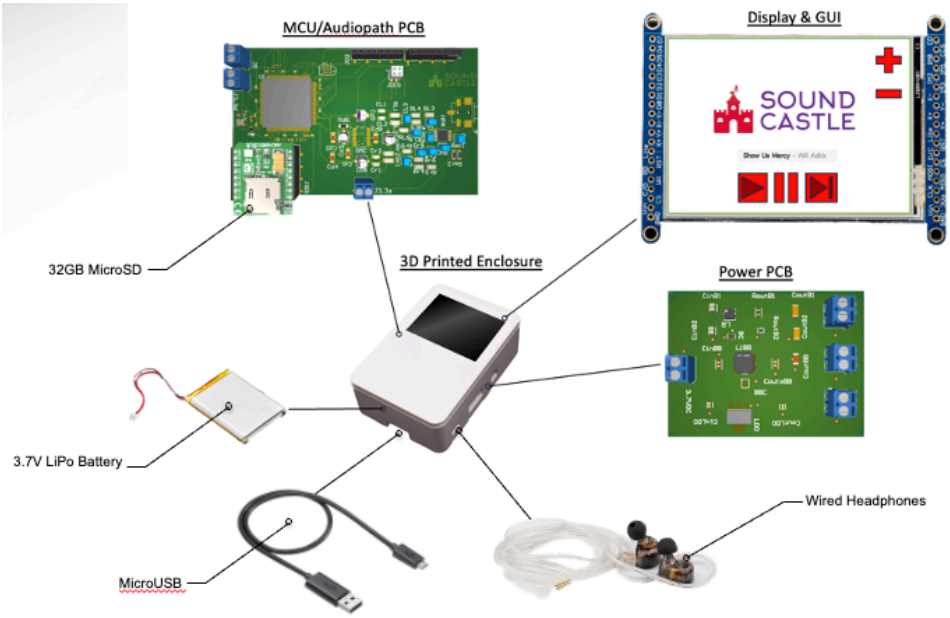


Figure 2: Integrated System Diagram

4.1. Main System PCB

The main system PCB includes all MCU and audiopath circuitry and ensures a reliable and functional system. The PCB also provides an interface for integration with the power PCB and LCD display. The board was designed using Altium Designer and manufactured by Advanced Circuits. Assembly and board bring-up was completed soon after design.

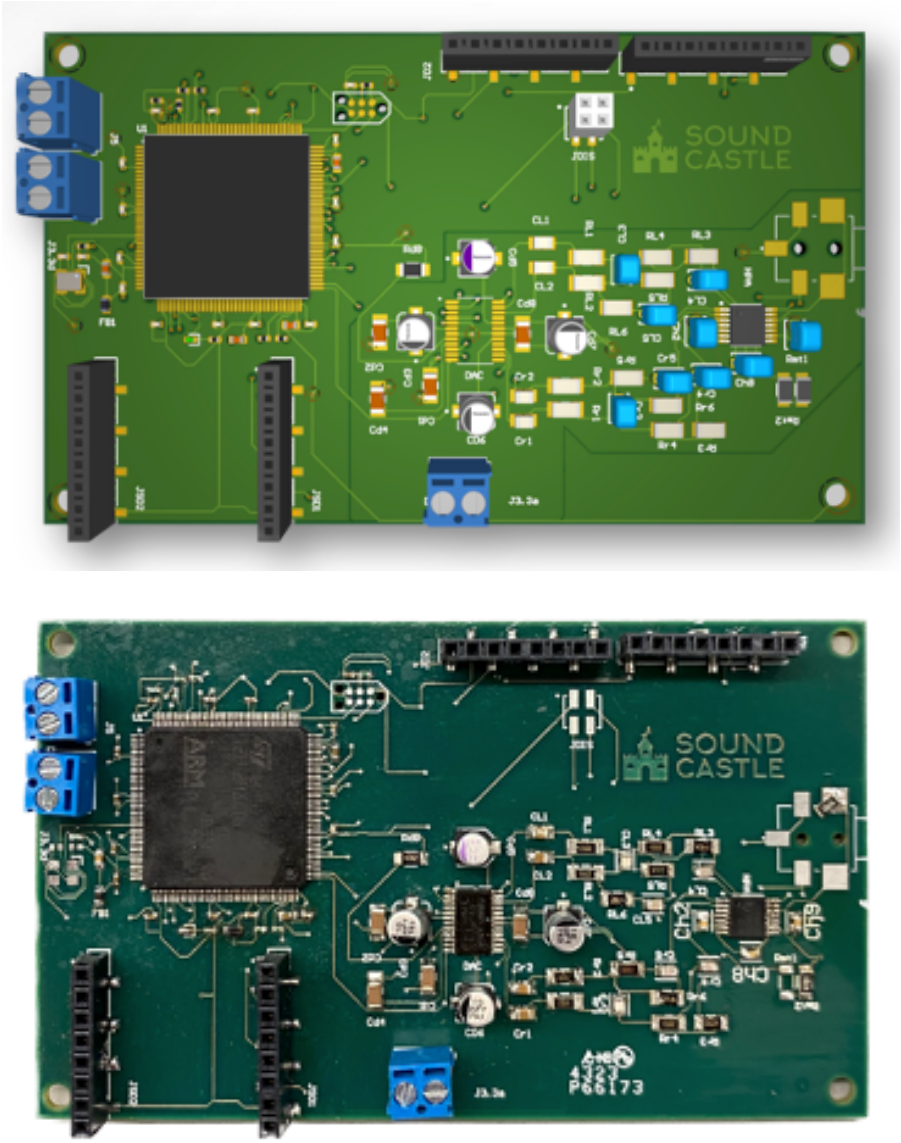


Figure 3: Main System PCB

Since our main goal with this digital audio player was to have the best audio we possibly could have; many considerations had to be made when designing the PCB. There are usually a few failure points for audio PCB designs, namely: magnetic coupling, through hole components, digital noise in analog circuits, and power supply noise.

In order to mitigate magnetic coupling, the number and length of parallel traces were kept to a minimum, especially in the analog portion of the PCB. If parallel traces run alongside one another for a long enough distance, magnetic coupling occurs and signals bleed over occurs, increasing THD and channel to channel bleed over, killing signal integrity.

Another way signal bleed over could occur is through vias and other exposed metal, hence making through hole components a terrible choice for audio circuits. There are no through hole components on our board. Through the strict use of surface mount components, the surface area of exposed metal was minimized, mitigating ambient noise being picked up alongside the chance of magnetic coupling.

The single biggest culprit of failed audio PCBs is the noise of Digital clocks and signals affecting the integrity of analog signals. In order to mitigate the effects, digital traces were kept as far as possible from the analog portion of the PCB, and separate ground planes were implemented for each signal type keeping the voltage references separate, further isolating digital and analog signals.

Power supply choices are very important in maintaining audio integrity. In order to power the analog audio circuits, an LDO was implemented to keep the power supply clean and maintain signal integrity. Since the digital circuitry doesn't use an LDO, power planes were made for the digital and analog supplies in order to keep them as separate as possible.

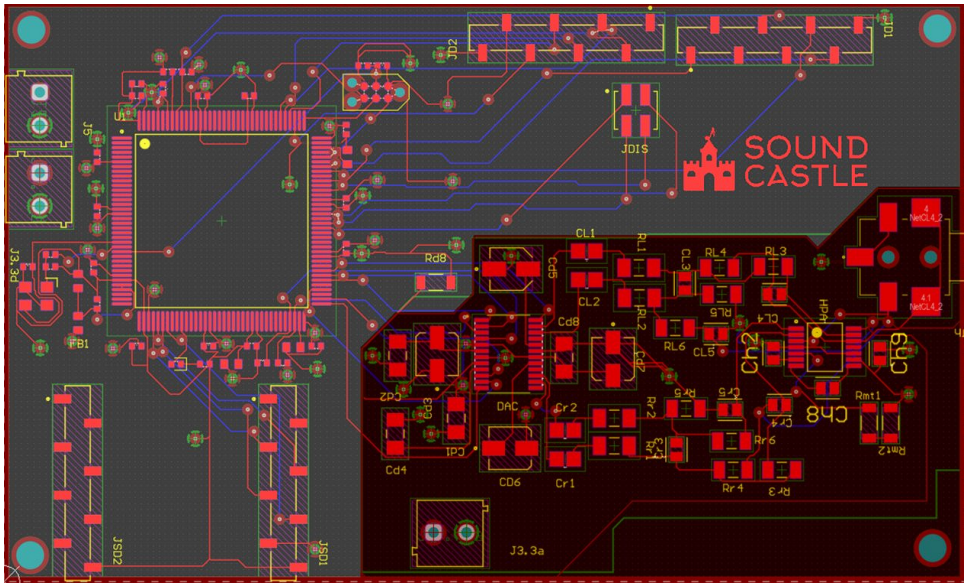


Figure 4: Main System PCB (2)

To ensure stable and reliable operation of the microcontroller, decoupling capacitors were placed near the supply pins to mitigate voltage fluctuations and improve power stability.

Clock management was implemented using a Pierce Oscillator consisting of an 8 MHz crystal and two capacitors forming a bandpass pi network. Capacitor values were selected to match the load capacitance of the crystal while accounting for the stray capacitance across the microcontroller.

Debugging and programming was implemented using a standard Host/Target interface along with serial wire (SW). The debug tool used to program the system was an ST-LINK/V2 with a JTAG/SW adaptor. Using this setup along with the STM32CubeIDE, we were able to use the standard GNU Debugger (GDB). Moreover, `printf` was redirected from standard out to the Instrumentation Trace Macrocell (ITM) data console. This greatly simplified board bring-up while saving GPIO that would have otherwise been used for UART.

Additionally, headers were added for connecting to the SD card and LCD display modules.

4.2. Power PCB

The power PCB includes all the power circuitry that allows for the digital audio player to be powered. The PCB is equipped with a boost converter circuit (3.7V-5V), a buck-boost converter circuit (3.7V-3.3V), and an LDO circuit (3.7V-3.3V). It is also equipped with several connections to connect to the battery/charging chip, digital components, and analog components. This PCB configuration enables efficient power management, facilitating the transition and regulation of voltage levels as needed.

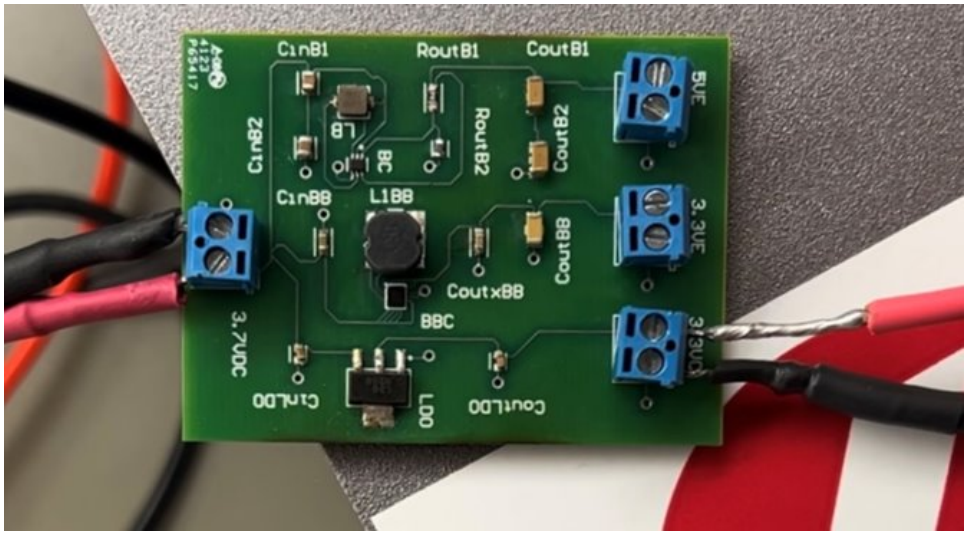


Figure 5: Power PCB

When designing the power PCB, the main goal was finding a way to provide separate signals for the analog and digital components. To accomplish this task, the buck-boost converter circuit and the LDO circuit were created. The buck-boost converter circuit powers all the digital components such as the MCU because the digital components need to be efficient, and the possible noisy signal of the converter does not hinder the digital components. The LDO circuit is used to power all the analog components (audio path) because all the audio path components need a clean voltage signal to ensure a clean listening experience for users from a power standpoint.

4.3. Integration with Display

Our original project plan involved using the STM32F4 chip on the system PCB to handle both the audio processing functions and the user interface control. During integration, we realized that having one microcontroller could not handle both tasks concurrently using a bare-metal implementation. After discussion with our instructor and sponsor, we decided to use a separate microcontroller for each task. An STM32 Nucleo development kit is used to control the display while the MCU chip is used for audio processing.

These two subsystems were integrated through a simple parallel communication protocol. We assigned each button a digital output pin on the display controller and formatted corresponding digital input pins on the other microcontroller. When a button is pressed, a digital pulse is sent from the display MCU to the audio MCU triggers an interrupt. The audio MCU handles this interrupt by calling the function associated with the specific button press. See user interface subsystem report for further details and some validation.

5. Results and System Validation

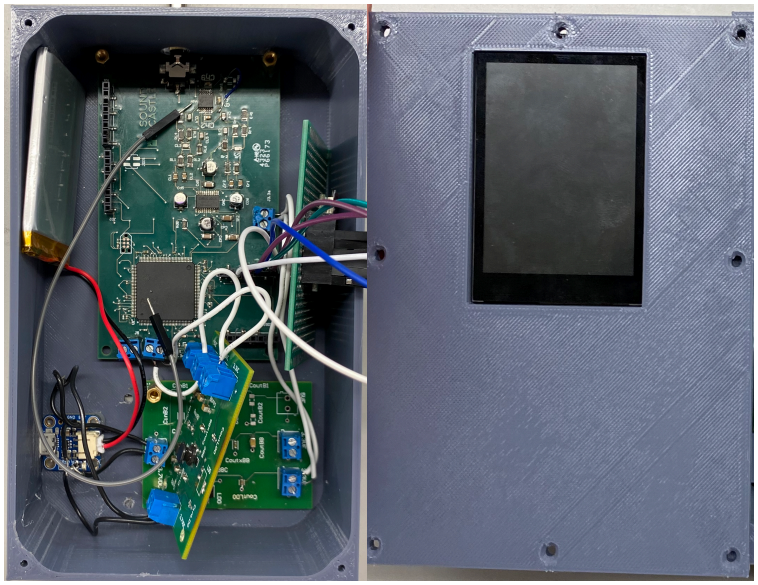


Figure 6: Integrated System

5.1. Crystal Oscillator

As mentioned previously, a Pierce Oscillator was designed for clocking the microcontroller. The crystal oscillator chosen was an 8 MHz crystal. The expected output of the crystal was a sine wave with a frequency of 8 Mhz. The figure below demonstrates the resulting signal.

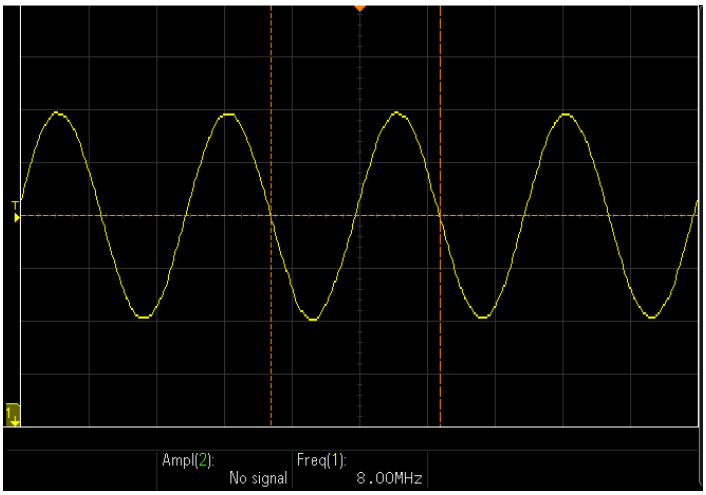


Figure 7: Crystal Oscillator Output

5.2. Buck-Boost Converter (Digital)

The system validation for the buck-boost converter was completed by using an E-load to run line and load tests on the circuit. These tests are designed to test the circuit integrity using various load sizes.

Table 1: Buck-Boost Converter Validation

Buck-Boost Converter E-Load Testing								
Load Test:			Line Test: (Load)			Line Test: (No Load)		
Vin	Vout	I	Vin	Vout	I	Vin	Vout	I
3.7	3.276	0	3	3.279	0.148	3	3.325	0
3.7	3.271	0.038	3.2	3.235	0.148	3.2	3.326	0
3.7	3.266	0.092	3.4	3.237	0.148	3.4	3.326	0
3.7	3.262	0.141	3.6	3.238	0.148	3.6	3.327	0
3.7	3.256	0.198	3.8	3.243	0.148	3.8	3.329	0
3.7	3.251	0.237	4	3.245	0.148	4	3.329	0
3.7	3.247	0.291	4.2	3.246	0.148	4.2	3.3	0
3.7	3.243	0.35						
3.7	3.239	0.407						
3.7	3.238	0.443						
3.7	3.236	0.512						

5.3. Voltage Regulator (Analog)

The validation for the LDO circuit was completed by using an E-load to conduct line and load tests on the circuit.

Table 2: LDO Validation

LDO E-Load Testing								
Load Test:			Line Test: (No Load)			Line Test: (Load)		
Vin	Vout	I	Vin	Vout	I	Vin	Vout	I
3.7	3.661	0	3	2.966	0	3	2.903	0.04
3.7	3.61	0.022	3.2	3.163	0	3.2	3.102	0.04
3.7	3.601	0.033	3.4	3.363	0	3.4	3.301	0.04
3.7	3.595	0.041	3.6	3.561	0	3.6	3.5	0.04
3.7	3.587	0.055	3.8	3.761	0	3.8	3.699	0.04
3.7	3.585	0.063	4	3.96	0	4	3.898	0.04
3.7	3.583	0.071	4.2	4.159	0	4.2	4.098	0.04

5.4. Charge/Discharge

The validation for the battery was conducted by running discharge and charge tests. For the discharge test, the battery was connected to the IMAX B6AC Dual Power Charge and discharged down to 3.2V at a rate of 0.5A.

Figure 8: Discharge Test

For the charging test, the battery was connected to the charging chip and was charged for 3 hours.

Table 3: Charging Test

Charging Test													
Time	0:00	0:15	0:30	0:45	1:00	1:15	1:30	1:45	2:00	2:15	2:30	2:45	3:00
Volts	3.29	3.46	3.53	3.54	3.55	3.56	3.58	3.6	3.62	3.63	3.65	3.66	3.67

5.5. Audio Characteristics

The requirements for the audio are as follows:

Pass band: **10 Hz - 30 kHz**

Voltage gain: **1.1 V/V**

THD: **-40 dB**

Voltage swing at full volume: **5.5 V**

Unclipped Voltage swing: **6 V**

Load capability: **60 ohms**

Dynamic range: **96 dB**

To validate the passband and voltage gain, a test signal was passed at the input of the headphone amplifier and the bode plot was generated at the output using an oscilloscope. The specifications were met almost exactly, the passband was found to be 8 Hz – 31 kHz and a voltage gain of 1.1 V/V. The full Volume voltage swing can also be seen here, alongside loading capabilities. The voltage swing was measured at 5.5 V with a 60-ohm load connected at the output. Since the output of the DAC is 5 V peak-to-peak, that confirms the gain to be 1.1 V/V.

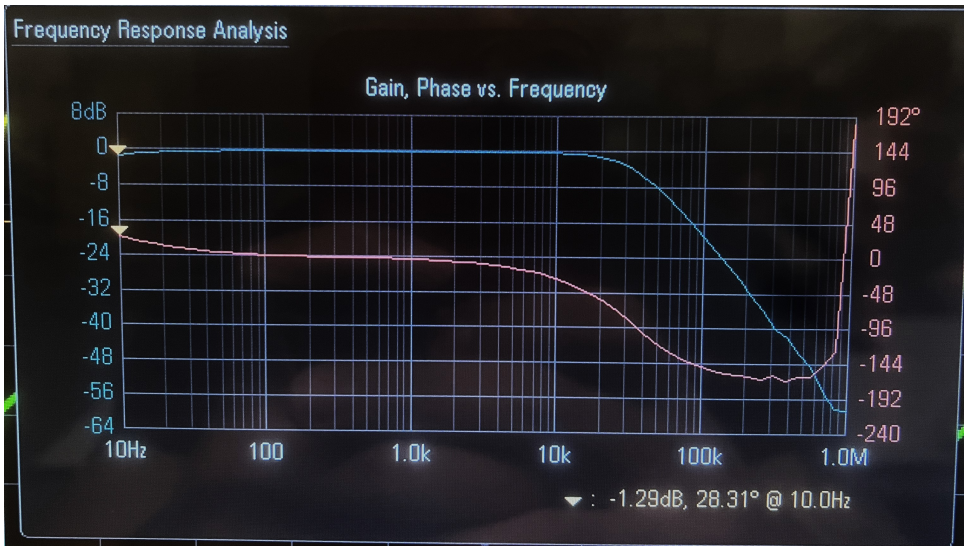


Figure 9: Headphone amplifier lower cutoff frequency

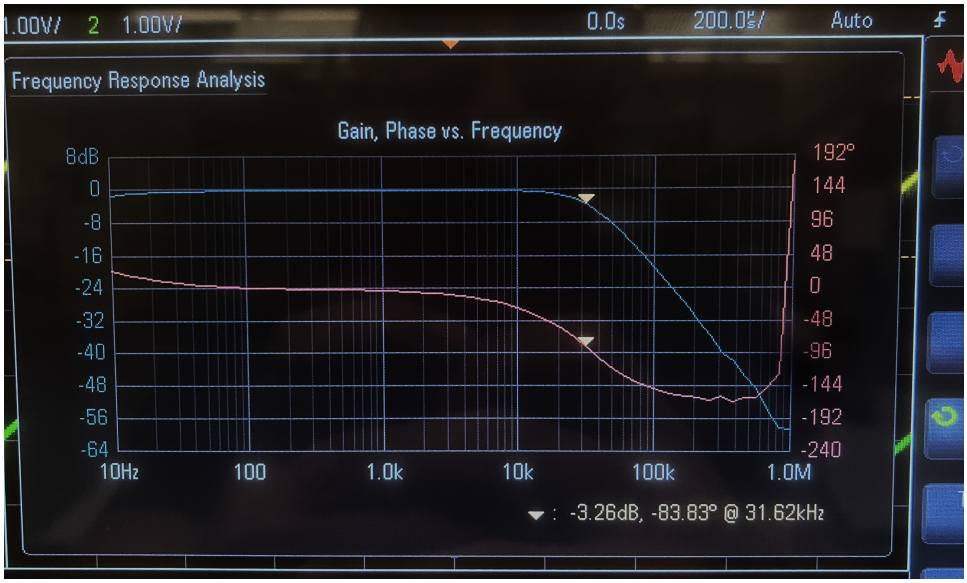


Figure 10: Headphone Amplifier Upper Cutoff Frequency

Figure 11: Output Swing at Full Volume

The TDH was validated by passing a 1 kHz pure sin wave through both the DAC and headphone amplifier, resulting in a THD of –41 dB, satisfying the specifications.

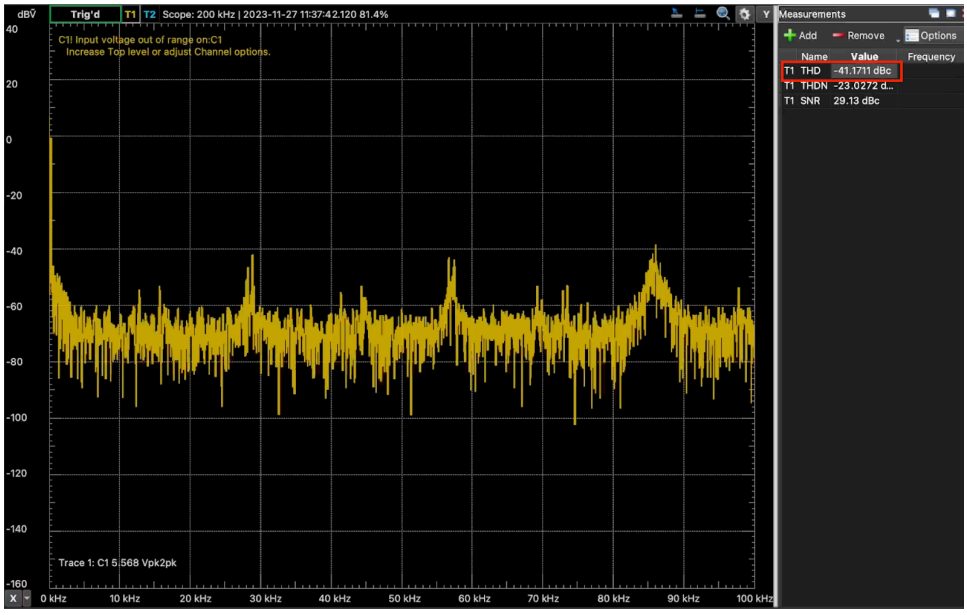


Figure 12: Full System THD

The unclipped voltage swing came in at 6.9 V due to the facts that the power supply outputs a voltage slightly higher than 3.3 V at full charge.

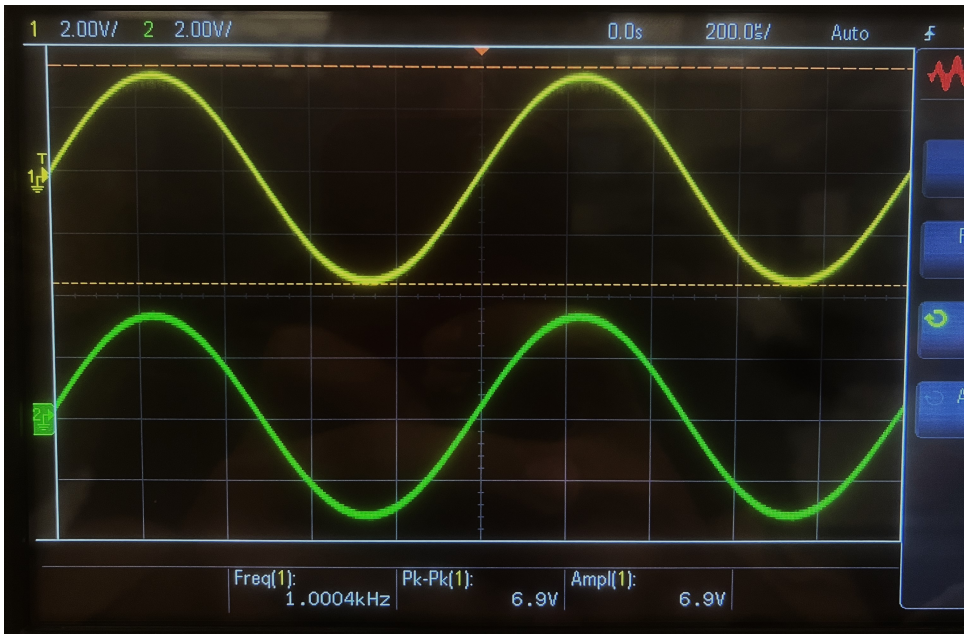


Figure 13: Full Unclipped Output Waveform

The dynamic range was untested. In order to test dynamic range, a full voltage signal must be passed, and the amplitude is measured. Then the input signal was to be attenuated by –60 dB and then the fundamental harmonic was to be removed from the output and the remaining amplitude be measured. Then the gain between the maximum swing and the noise was to be the dynamic range. The attenuated input was impossible with the MCU and therefore dynamic range was untested.

5.6. MCU – Audiopath Integration Test

The MCU – Audiopath integration test tests that the MCU and Audiopath subsystems interface correctly. It tests all functions and communications protocols used involved in reading wav data from a file located on an SD card, configuring the DAC, and the I2S buffer implementation using DMA and interrupts.

A test signal was generated using Audacity software. This test waveform included two sine waves with fundamental frequencies 150Hz and 300Hz. The two sine waves have different frequencies to ensure that the signal is not being repeated. Each sin wave also has a header and footer ramp. Each section is 200ms long which is 8820 samples with 44.1kHz sampling. Using this information, I configured my I2S buffer to have length 8820*2. This is

because I have a half-complete and full-complete DMA transfer callback functions that are called whenever the first/second half of the buffer is filled.

This is intentional because issues are most likely to occur at the edges of sections. If wave deformation occurs only in these regions than that would indicate a problem with my buffer scheme.

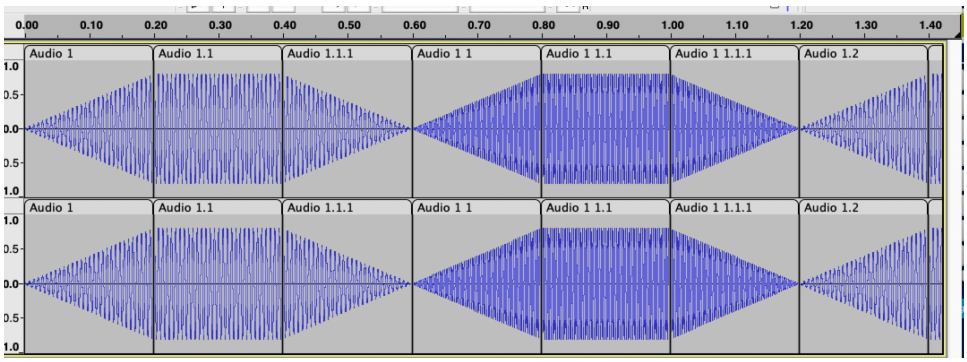


Figure 14: Subsystem Validation Test Signal

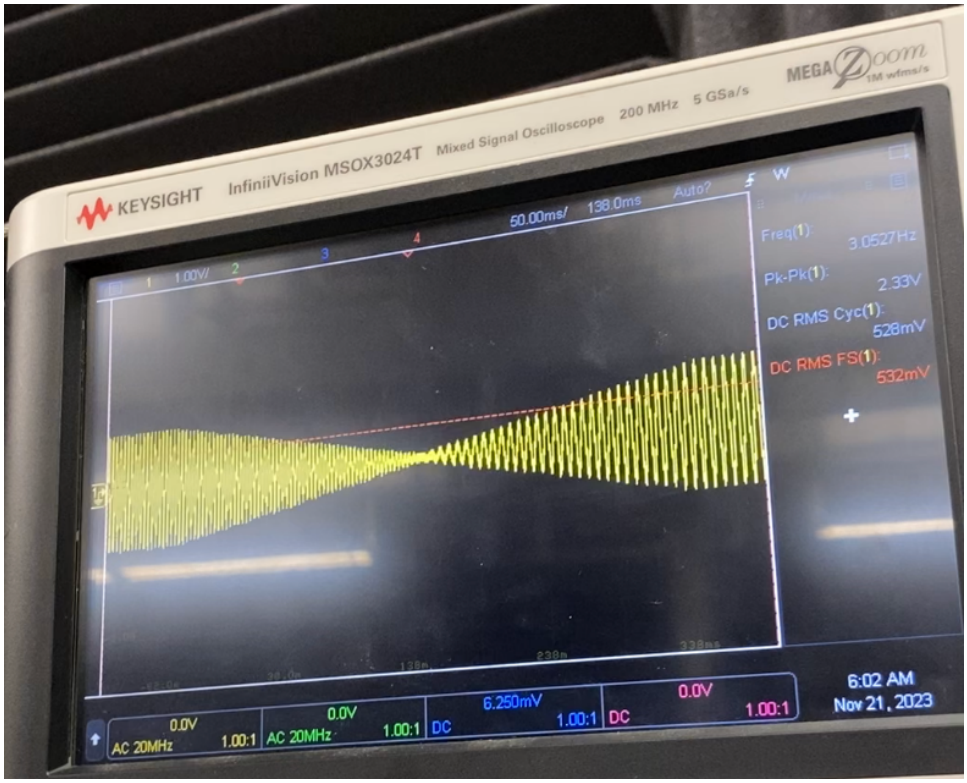


Figure 15: Analog Signal Measured from DAC Output

The results of the test indicate that all communication protocols function properly, that there are no issues in reading from the SD card, that the DAC is configured properly, and that the buffer scheme is correct. This measurement was taken from one of the output channels of the DAC.

For further validation results regarding each subsystem, refer to that subsystem's corresponding section in the *Subsystem Report*.

6. Conclusion

This report concludes our work completed over a twelve-month period for our senior capstone design. All subsystems went through major revisions after the conclusion of the first semester. At the onset of the second semester, the audio processing system was restarted from scratch, and all source code, re-created.

System integration was at the forefront for the second semester. The focus was shifted from individual subsystems to a unified system. System validation was also a major focus of the second semester; the system was rigorously tested to ensure that it met the project specifications outlined in the *Functional System Requirements* included in this document.

The resulting system met these specifications. It was demonstrated in front of university faculty and evaluated for its completeness and our application of fundamental engineering principles.

Future iterations of this project would increase the bit depth from 16 to 32 bits. This could be implemented by selecting a DAC that supported a higher bit depth or even using two DACs. Moreover, a real-time operating system (RTOS) would enable greater concurrency and lead to a more performant system.

All source code and design documents are included in the project repository.