EE 256 Final Project: Audio Amplifier Module

Sara Davidova and Dennis Woo

Fall 2022

Abstract

Audio amplifiers are ubiquitous devices that can be integrated with microcontrollers or other computers through proper communication protocols and data conversion. In this project, a digital to analog converter (DAC) and a class D audio amplifier were combined together into a single PCB board for a customized radio application, as seen on Figure 1. The device was laid out on a 4-layer PCB and controlled with a Feather M4 microcontroller operating on CircuitPython. The Feather M4 communicated mute, shutdown, and volume commands through I2C protocol. Audio data was inputted via the I2S protocol, and the amplifier powered two 20W speakers.

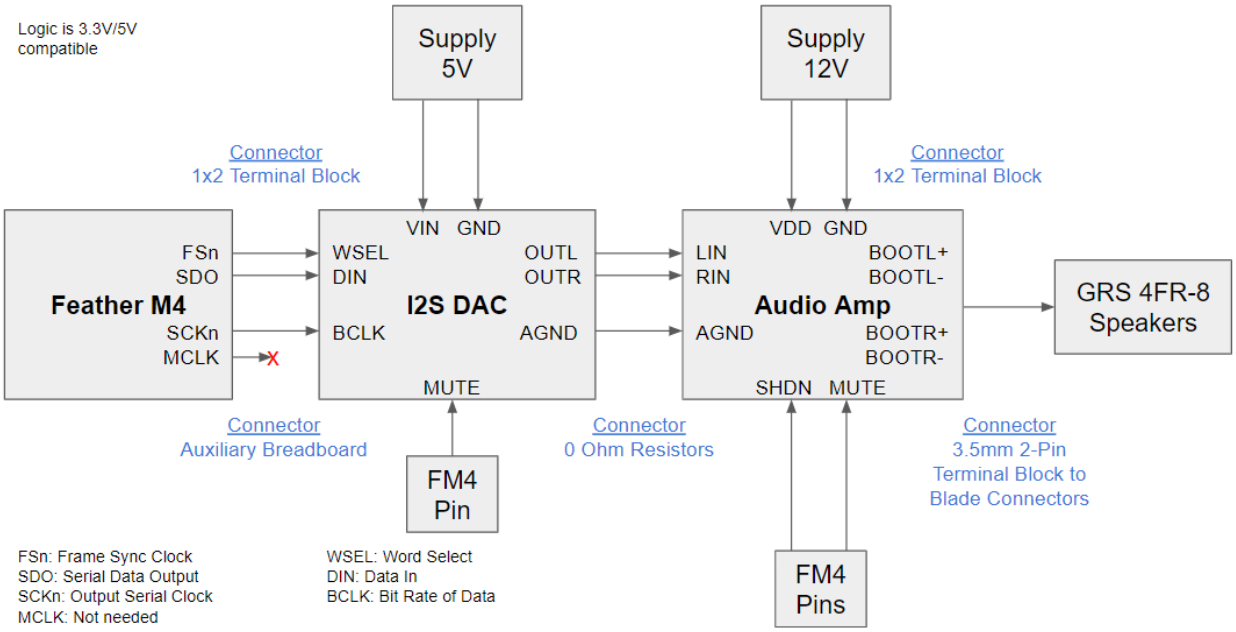

Figure 1: Block Diagram of the Audio Amplifier Module

Module Specifications

Power Rail Input for Audio Amplifier	12 V
Power Rail Input for DAC	5 V
Digital Communication Power Rail (VDDIO)	3.3 V/5 V compatible
Class D Amplifier Efficiency	93 %
Speaker Power Rating	20 W

Mute	VDD to mute / GND to play
Shutdown	VDD to mute / GND to play
Volume	6 Bit Value (0 - 63)

Table 1: Specifications

Component Choices

The design of the module (Figure 2) was inspired by two Adafruit boards that made use of the UDA1334A digital to analog converter and the MAX9744 amplifier. These two boards were combined into one, and the board was stripped of many unnecessary submodules and components that the original Adafruit boards included for customization, such as an I2C communication line for the DAC chip (pins associated with these functionalities were tied high or low as a result). Component values were matched to the Adafruit schematic, and all components were chosen to have a tolerance of 20% or better.

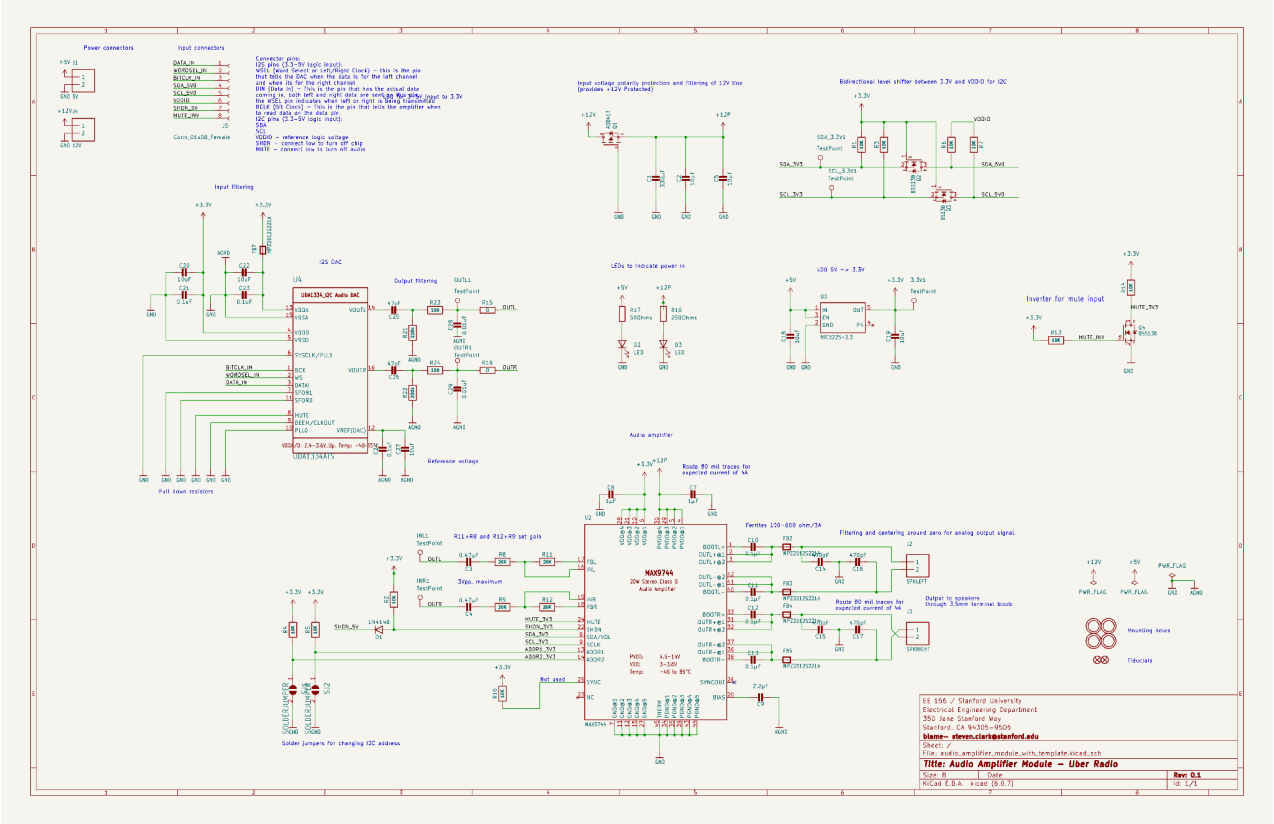

Figure 2: The Schematic Including the DAC and the Amplifier

Due to unavailability of the Adafruit-recommended ferrite beads filtering the DAC's power line and amplifier's output, two alternate beads were chosen from Murata Electronics: the

BLM31KN271SH1L and BLM31KN471SH1L. The impedance characteristic in the former (270 Ω at 100 MHz) was chosen to roughly match that recommended by Adafruit (220 Ω at 100 MHz). The impedance in the latter is significantly higher (470 Ω at 100 MHz) and was selected as an option if the power line needed heavier filtering than expected.

All the submodules necessary for the implementation are included in the schematic on Figure 2. Among these are 1. Power connectors and digital input pins; 2. Input voltage polarity protection; 3. Level shifter to digital logic input; 4. LDO; 5. Inverter for mute command; 6. Power input LEDs; 7. DAC IC; 8. Audio amplifier IC; 9. Fiducials, mounting holes. For more details of the schematic, see attached KiCAD files.

Layout

The circuitry was laid out on a 4-layer board of dimensions 90mm by 52mm, with the top and bottom planes accommodating traces and the middle two planes accommodating the 3.3V power rail and GND, respectively (Figure 3). Apart from functionality, the primary focus of the layout was to ensure ease of debugging. Seven looped test points were added to power planes and the inputs and outputs of each half of the system to allow for quick measurement with oscilloscopes. In addition, the DAC and audio amplifier were connected by a 0 Ω resistor—a component that could be easily removed to test the two parts of the module separately. Where possible, components with footprints smaller than an 0402 were avoided. Blue LEDs were added in series with the 12V and 3.3V power lines to indicate whether current was flowing properly. A header-pin connector for all inputs was added to allow the board to be connected to a breadboard. Finally, for ease of populating the board with components, two fiducials were placed on the board.

An ímportant layout concern was making sure each trace was compatible with the amount of current flowing through it, which differed vastly between the low power digital circuitry and the high power amplification circuitry. Traces of the amplification circuitry were sized for 4A of current, and traces of steadily increasing width were used where traces left the amplifier IC.

One interesting consideration—and also the source of a layout mistake—was the consideration of ground connections. The Adafruit schematic listed both analog and digital grounds for the DAC and amplifier, likely to reduce noise in the digital signals from traversing into the analog circuitry. To simplify the circuit, these grounds were shorted together in the final schematic (both types of ground were connected by vias to the ground plane). However, no header pin was added to connect the ground of the Feather M4 to the PCB module, and to effectively operate the board during testing, a jumper wire was soldered to provide this ground connection.

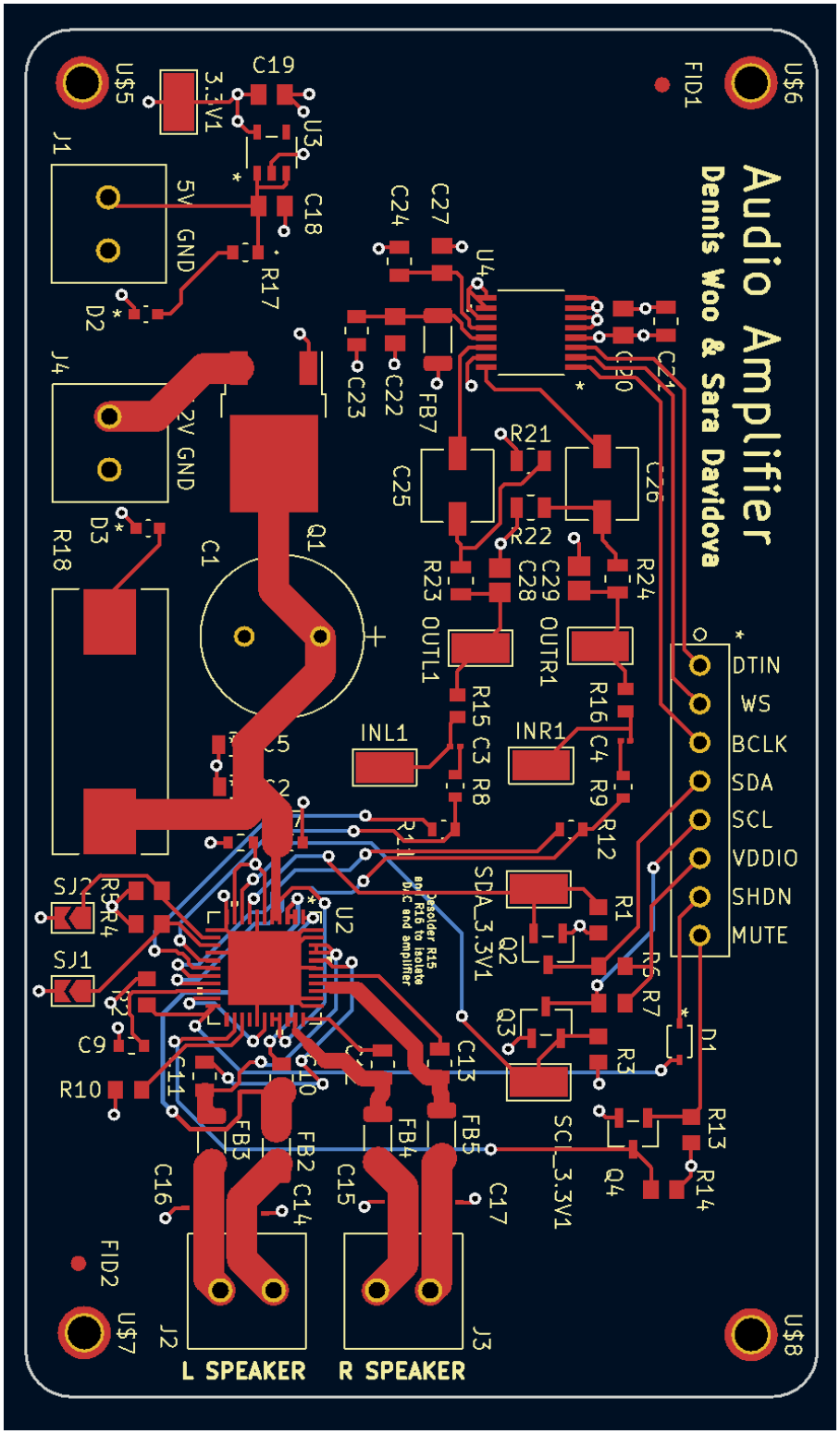

Figure 3: Final Layout
(the header pins labeled WS and DTIN were originally reversed, a bug that was caught during test and verification)

Test and Verification

Testing was conducted using CircuitPython and Adafruit Libraries operated on the Mu interface. When assembling the boards, the 0Ω resistors connecting the DAC and the amplifier were left unpopulated so that each module could be tested separately. To test the DAC, a sine wave was generated in CircuitPython, and Adafruit libraries were used to communicate it through the Feather M4 via I2S to the DAC. The output from the DAC was measured at the test points using an AnalogDiscovery 2 oscilloscope. This test uncovered a layout mistake. The WS and DTIN traces were rerouted and their connector pins switched to enable parallel routing. However, the silkscreen labels next to the connector pins were unchanged. When flipping the connections, the DAC was confirmed to output a zero order hold sine wave. The 0Ω resistors were then put in place and the amplifier was tested on a .wav file of a song that was converted to audio samples and fed through the DAC and into the audio amplifier. Finally, the volume, mute, and shutdown commands were verified. See the appended code for detail.

Conclusions

Our layout design decisions truly made the board bring-up and testing very simple. The multiple test points, LEDs, and 0Ω resistors were optimized for the planned testing procedure, and made it quick, safe, and informative. Probing was mostly soldering-free, we could quickly check things like power rails without risk of shorting the board, and we were able to test the two main ICs separately. Similarly successful was the code setup for the testing procedure. Using Adafruit libraries for the I2C and I2S communication protocols was extremely simple and did not even require the setting of the operating frequencies, etc. We were surprised by the ease with which data was communicated and the system was debugged thanks to the software abstractions implemented and tested by Adafruit.

Unfortunately, several issues and imperfections were discovered during the testing and verification. Firstly, the improper ground connection described above had to be fixed by adding an additional jumper to connect the Feather M4 ground to the circuit (and power supply) ground. Secondly, an issue with the flipped silkscreen labels had to be debugged and - although easily solved by rewiring - would need to be corrected in the next iteration of the design. Additionally, although the functionality was not compromised, two changes would make the design more robust. Firstly, while the traces were sized appropriately for current, the vias were kept at a default diameter, which implies higher power dissipation in heat per area, as well as an impedance mismatch which might be a problem for high frequency signals. Secondly, some 0201 capacitors were selected by accident, making debugging near these components as well as soldering of these components extremely cumbersome. These four issues would however be easily fixed in a next board spin.

Appended Code:

```
import board
import busio
import adafruit_max9744
import digitalio
import time
import array
import math
import audiocore
import audiobusio

# Select Test. Do not select both tests at the same time.
DAC_test_en = 0
AMP_test_en = 1

# Tone Setup
tone_volume = 1  # Increase this to increase the volume of the tone.
frequency = 800  # Set this to the Hz of the tone you want to generate.
length = 40000 // frequency
sine_wave = array.array("h", [0] * length)
for i in range(length):
   sine_wave[i] = int((math.sin(math.pi * 2 * i / length)) * tone_volume * (2 ** 15
-1))

# For Feather M4 Express
wave_file = open("StreetChicken.wav", "rb")
wave = audiocore.WaveFile(wave_file)

#                          bclk   WS           data
audio = audiobusio.I2SOut(board.D1, board.D10, board.D11)
sine_wave_sample = audiocore.RawSample(sine_wave)

# VDDIO
vddio = digitalio.DigitalInOut(board.D9)
vddio.direction = digitalio.Direction.OUTPUT
vddio.value = True         # Provide Logic High

print("All setup completed")

# DAC TESTING
if DAC_test_en:
   while True:
       audio.play(wave, loop=True)
       time.sleep(1)
       #audio.stop()
       time.sleep(1)

elif AMP_test_en:
   print("Testing amplifier")
   # AMPLIFIER TESTING
   # Set up mute button output
   mute = digitalio.DigitalInOut(board.D12)
   mute.direction = digitalio.Direction.OUTPUT
```

```
mute.value = False;
print("mute initiated")

# Set up shutdown output
shutdown = digitalio.DigitalInOut(board.D13)
shutdown.direction = digitalio.Direction.OUTPUT
shutdown.value = False;
print("shutdown initiated")

# Initialize I2C bus.
i2c = busio.I2C(board.SCL, board.SDA)
print("i2c Initiated")

# Initialize amplifier.
amp = adafruit_max9744.MAX9744(i2c)
print("amplifier object initiated")

# Setting the volume is as easy as writing to the volume property
amp.volume = 25  # Volume is a value from 0 to 63 where 0 is muted/off and
print("volume set")

while True:
    audio.play(sine_wave_sample, loop=True)

    # Shutdown and mute sequence to test out
    time.sleep(1)
    mute.value = True          # Mute!
    print("Muting")
    time.sleep(1)
    mute.value = False         # Unmute!
    print("Unmuting")

    time.sleep(1)
    shutdown.value = True      # Shutdown!
    print("Shutting down")
    time.sleep(1)
    shutdown.value = False   # Turn on again?
    print("Turning back on")

    # In addition you can call a function to instruct the amp to move up or down
    # a single volume level.  This is handy if you just have up/down buttons in
    # your project for volume:
    #amp.volume_up(20)  # Increase volume by one level.
    #print("Volume up")
    #time.sleep(1)
    #amp.volume_down(20)  # Decrease volume by one level.
    #print("Volume down")
    #time.sleep(1)

    audio.stop()
    time.sleep(1)
```