ezDV: Low Cost Digital Voice on the ESP32

Mooneer Salem, K6AQ (mooneer@gmail.com)
August 2022

Abstract

FreeDV is a series of digital voice modes optimized for use on the HF portion of the radio spectrum that uses the Codec2 open source voice codec library. Traditionally, this has taken the form of a desktop application running on a Windows, Mac or Linux PC. However, there are challenges with this approach, especially for portable operation. This paper will discuss a low cost approach with high ease of use that eliminates the need for a computer for encoding and decoding digital voice.

Introduction

Codec2 is a library originally created by David Rowe (VK5DGR)[1] in 2009 to provide an open source, patent unencumbered digital voice codec for use on amateur radio. Since that time, the library has been extended to also support the various FreeDV modems required to use Codec2 on the air, as well as created a GUI based application[2] to enable their use.

While usage has gradually increased, there are several current pain points surrounding initial configuration as well as more practical concerns with the need for a PC to encode and decode FreeDV. Unlike with most popular ham radio applications (for example, WSJT-X), FreeDV requires two sound cards in order to be able to fully use the mode: one for the radio interface and the other for analog audio input and output. This is on top of the typical digital mode setup (e.g. adjusting transmit audio to ensure that ALC does not show any indication on the radio) and is different enough that it has caused confusion for some operators in the past.

In addition, unless one has a laptop that can be carried out into the field for portable use, the requirement that one runs FreeDV on a PC inherently limits it to stationary, home use. Even if one owns a laptop and is willing to take it portable, the decision to do so inherently results in having to rethink one's power budget and other constraints, which could result in having to make other changes to the portable station to stay within those constraints. Depending on one's priorities during portable operation, this could mean that FreeDV is left at home anyway.

---

[1] Rowe, David. "Open Source Low Rate Speech Codec Part 1." *Rowetel*, 26 Aug. 2010, https://www.rowetel.com/?p=128.
[2] "FreeDV: Open Source Amateur Digital Voice." *FreeDV*, https://freedv.org/.

An alternative to a full-featured laptop is an embedded device that can perform basic FreeDV modulation and demodulation. One example of such a device is the SM1000[3]. The SM1000 uses an STM32F4 microcontroller and is able to modulate and demodulate three FreeDV modes: 700D, 700E and 1600. However, due to the ongoing chip shortage, the SM1000 has been sold out with no ETA for return as of the time of this writing. The relatively high price prior to it selling out (~US$200) also made it less accessible to hams that could not easily afford to purchase the device without already being heavy users of FreeDV.

The ESP32 Microcontroller

The ESP32 was originally brought to market by a company called Espressif in 2013 in the form of the ESP8266[4]. This device was the first product to achieve wide adoption by the maker community due to its low cost and the inclusion of wireless functionality. Initially, documentation was solely in Chinese, but English language documentation along with a Software Development Kit (SDK) and libraries came about in short order. The wide adoption and wireless functionality made it a natural candidate for an embedded amateur radio device, especially as many newer radios are coming with network connectivity as a standard (or at least reasonably priced[5]) option.

As of this writing, there were a few variants[6] of the ESP32 that were found to possibly be suitable: the original ESP32 and the ESP32-S3. Unfortunately, the -C3 and -S2 variants are not suitable for FreeDV at this time due to the heavy usage of floating point operations in the library; previous tests by this author on the RP2040 (from the Raspberry Pi project)--which also has no floating point unit–resulted in performance significantly slower than real time[7]. Upon further investigation, it was found that the -S3 variant also has support for SIMD instructions[8] (similar to SSE on the Intel x86 architecture). There was previous successful experience optimizing portions of the Codec2 library to take advantage of those instructions on the PC[9], so the ESP32-S3 was ultimately chosen to enable this possibility during development.

---

[3] Rowe, David. “SM1000.” *Rowetel*, http://rowetel.com/sm1000.html.
[4] Cording, Stuart. “What Is the ESP32? Its Brief History and How to Get Started.” *Elektor*, 10 Mar. 2022, https://www.elektormagazine.com/articles/what-is-the-esp32.
[5] “Welcome to Wfview.org.” *Wfview*, https://wfview.org/.
[6] “Chip Series Comparison.” *ESP-IDF Programming Guide*, https://docs.espressif.com/projects/esp-idf/en/latest/esp32/hw-reference/chip-series-comparison.html.
[7] Salem, Mooneer. “Fixed Point/FPGA Implementation of Codec2 · Discussion #223 · drowe67/codec2.” *GitHub*, 31 Oct. 2021, https://github.com/drowe67/codec2/discussions/223#discussioncomment-1565289.
[8] *ESP32S3 Series - Espressif*. https://www.espressif.com/sites/default/files/documentation/esp32-s3_datasheet_en.pdf.
[9] Salem, Mooneer. “700C: Use Vector Ops for rx_filter_coh() When Possible. by Tmiw · Pull Request #200 · drowe67/codec2.” *GitHub*, 24 July 2021, https://github.com/drowe67/codec2/pull/200.

Porting to ESP32

The next step after deciding on the ESP32-S3 was to buy a development device to begin testing the FreeDV codebase. The nanoESP32-S3 was discovered during a search of Tindie[10] and quickly purchased. For futureproofing, the “N8R8” variant with 8MB of PSRAM and flash was selected.

Upon receipt of the nanoESP32-S3, work was undertaken to attempt to compile the Codec2 library using the ESP-IDF development environment. As the ESP-IDF development environment natively uses CMake (much like FreeDV and Codec2), integration with a sample project using the “__EMBEDDED__” compiler definition (which enables various memory and space optimizations to allow it to fit onto the STM32F4) was straightforward. However, upon initially attempting to compile the test application, errors related to the CMSIS library[11] appeared on the console.

Investigating further, it was discovered that the usage of CMSIS was limited to a single file called “ofdm.c” inside the Codec2 library. These references were to complex value and real valued dot product operations provided by CMSIS (namely arm_dot_prod_f32()[12] and arm_cmplx_dot_prod_f32()[13]). Fortunately, Espressif’s ESP-DSP library also contained similar functions for performing dot product operations[14] and used the ESP32 SMID instructions when available. An initial pass at updating the ofdm.c file to use the ESP-DSP functions instead of CMSIS proved to be successful.

Before submitting the pull request to the Codec2 project, the changes made to use ESP-DSP were generalized to support architectures other than ARM and ESP32. This was done through the use of a known wrapper interface that embedded device designers would use to implement the required dot product operations. The SM1000 code was also updated to implement these wrapper functions using the CMSIS library. After testing these changes, a pull request was created in the Codec2 project to merge them into the codebase[15].

---

[10] Wu, Johnny. “NanoESP32-S3 Development Board by MuseLab on Tindie.” *Tindie*, https://www.tindie.com/products/johnnywu/nanoesp32-s3-development-board/?pt=ac_prod_search.
[11] “CMSIS.” *Arm Developer*, Arm Ltd., https://developer.arm.com/tools-and-software/embedded/cmsis.
[12] “Vector Dot Product.” *CMSIS-DSP Software Library*, https://www.keil.com/pack/doc/CMSIS_Dev/DSP/html/group__BasicDotProd.html#gadf26f6bc62d6416528663ad3e46fbf67.
[13] “Complex Dot Product.” *CMSIS-DSP Software Library*, https://www.keil.com/pack/doc/CMSIS_Dev/DSP/html/group__cmplx__dot__prod.html#ga93796e73f02771cf6fe13de016e296ed.
[14] “Espressif DSP Library API Reference.” *Espressif DSP Library v1.2.0-3-g2415e61 Documentation*, https://docs.espressif.com/projects/esp-dsp/en/latest/esp-dsp-apis.html#dot-product.
[15] Salem, Mooneer. “Allow Use of __EMBEDDED__ on Non-ARM Systems. by Tmiw · Pull Request #317 · drowe67/codec2.” *GitHub*, 3 Apr. 2022, https://github.com/drowe67/codec2/pull/317.

Audio Input and Output

The next step after confirming that the FreeDV codebase was able to compile on the ESP32 was to set up audio I/O. Unfortunately, while the ESP32-S3 does have an analog to digital converter (ADC), it does not have the functionality to emit an analog signal again. Additionally, the ADC that did exist on the -S3 had 12 bits of resolution[16]. It was decided to investigate additional audio ADC and DAC chips that could be integrated onto the board.

One product that particularly stood out was the TLV320 series audio codec chip by Texas Instruments. In particular, the AIC3254 variant had two ADC and two DAC channels[17], which made it ideal for processing both the radio audio and the analog headset audio. The chip also supported various audio routing modes (for example, the left channel of the audio going to the ESP32 could be configured to be coming from input #1 while the right channel could be configured as coming from input #2). As FreeDV solely works with mono audio signals and ignores the second channel, this made it possible to only need one TLV320 on the board to handle all the audio requirements of the device. Its cost was also fairly low at around $10 each in small quantities[18].

To make sure that the TLV320 would be satisfactory for the device, investigation was done into purchasing an EVM (also known as a development board) containing the TLV320. However, the TLV320 EVM was US$250 when purchased directly from Texas Instruments[19], a significant cost for effort that may possibly need to be thrown away. Instead, it was determined that a first pass at building a development board containing the TLV320 would be done instead.

Building the Development Board

To build the development board, a new KiCad project was created to hold the schematic and PC board layout. Pin headers were then placed in the schematic to represent the nanoESP32-S3, followed by the TLV320 and its required passive components as per its datasheet and technical reference documentation. Two TRRS 3.5mm jacks were also added, one for radio audio and one for a wired headset. Buttons and LEDs were also added to represent the most common operations (PTT, volume up/down and selecting FreeDV modes) and system status (PTT on, network connection, sync and audio overload).

Once these components were routed on the PCB, some thought was taken as to how it would be assembled once the boards arrived. While this author does have some SMD soldering

---

[16] “Analog to Digital Converter (ADC).” *ESP-IDF Programming Guide*, https://docs.espressif.com/projects/esp-idf/en/v4.4/esp32s3/api-reference/peripherals/adc.html.
[17] “TLV320AIC3254.” *TLV320AIC3254 Data Sheet, Product Information and Support | TI.com*, Texas Instruments, https://www.ti.com/product/TLV320AIC3254.
[18] “TLV320AIC3254IRHBT.” *JLCPCB*, https://jlcpcb.com/partdetail/TexasInstruments-TLV320AIC3254IRHBT/C182339.
[19] “TLV320AIC3254EVM-K.” *TLV320AIC3254EVM-K Evaluation Board | TI.com*, Texas Instruments, https://www.ti.com/tool/TLV320AIC3254EVM-K#order-start-development.

experience, the experience was lacking for components as small as the TLV320. Thus, it was decided to take advantage of JLCPCB's SMD assembly service. This service was reasonably priced for prototyping (generally part cost plus a fee per part for what they call "extended parts"). After some time spent in associating parts on the schematic with parts that were in stock at JLCPCB, the Gerber files were submitted to them and payment was made.

While waiting for the initial boards to arrive, development of the firmware for the ESP32 was undertaken. This involved a first pass at writing the initialization and audio I/O code (using the I^2C and I^2S functions available as part of ESP-IDF) as well as a basic loop for sending audio through the Codec2 library. The initial effort completed in time for the arrival of the development boards.

Debugging the Firmware

Unfortunately, it was discovered that one of the pins on the nanoESP32-S3 that was initially intended for use with the TLV320 was shared by the multi-color LED. As a workaround, the pin on the board's pin header was cut and the female side bridged to a neighboring pin. The change was also reflected in the schematic and PCB for the next revision of the board.

Additionally, the initial pass had significant issues with configuring the TLV320. During development, it was decided to configure the TLV320 such that it used an 8 KHz sampling rate; this sample rate is the native sample rate as used in the FreeDV codebase. However, much of the sample code available for the TLV320 assumed a 48 KHz or 44.1 KHz sample rate, so a guess as to the data to send over the I^2C bus was made. Significant time was spent performing modification/flash/test cycles trying different commands and command orderings in an attempt to have working audio, but eventually a series of commands was found that properly configured the chip for 8 KHz audio.

Because of the way that audio was being fed into the Codec2 library, however, part of one of the input audio channels did not work properly. Initially this was thought to be a hardware issue for two reasons: seeing pulsing on the offending channel with an oscilloscope and due to the issue only affecting one channel and not the other. A second revision of the board was spun that did the following:

- Avoided use of the GPIO pin that was connected to the multi-color LED on the nanoESP32-S3,
- Rerouted all traces to the TLV320 to shorten them as much as possible,
- Added additional vias connecting ground planes (especially surrounding the sensitive audio traces coming from the TLV320) to improve noise immunity,
- Tweaked the values of various components on the schematic/PCB, and
- Added mounting holes for future mounting in an enclosure.

These resulted in the following schematic and PCB design:

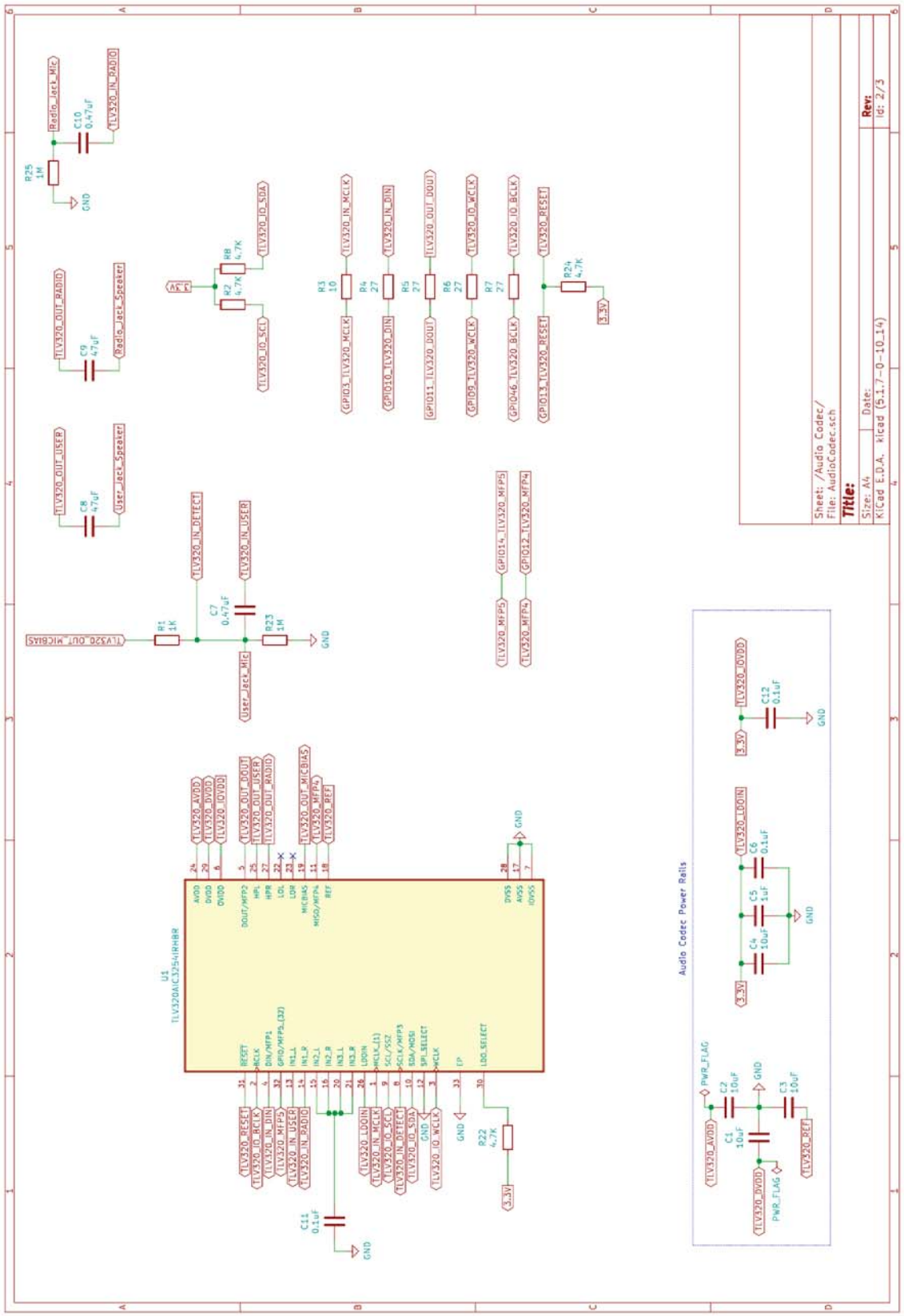

Figure 1: The TLV320 and supporting components on the ezDV schematic.

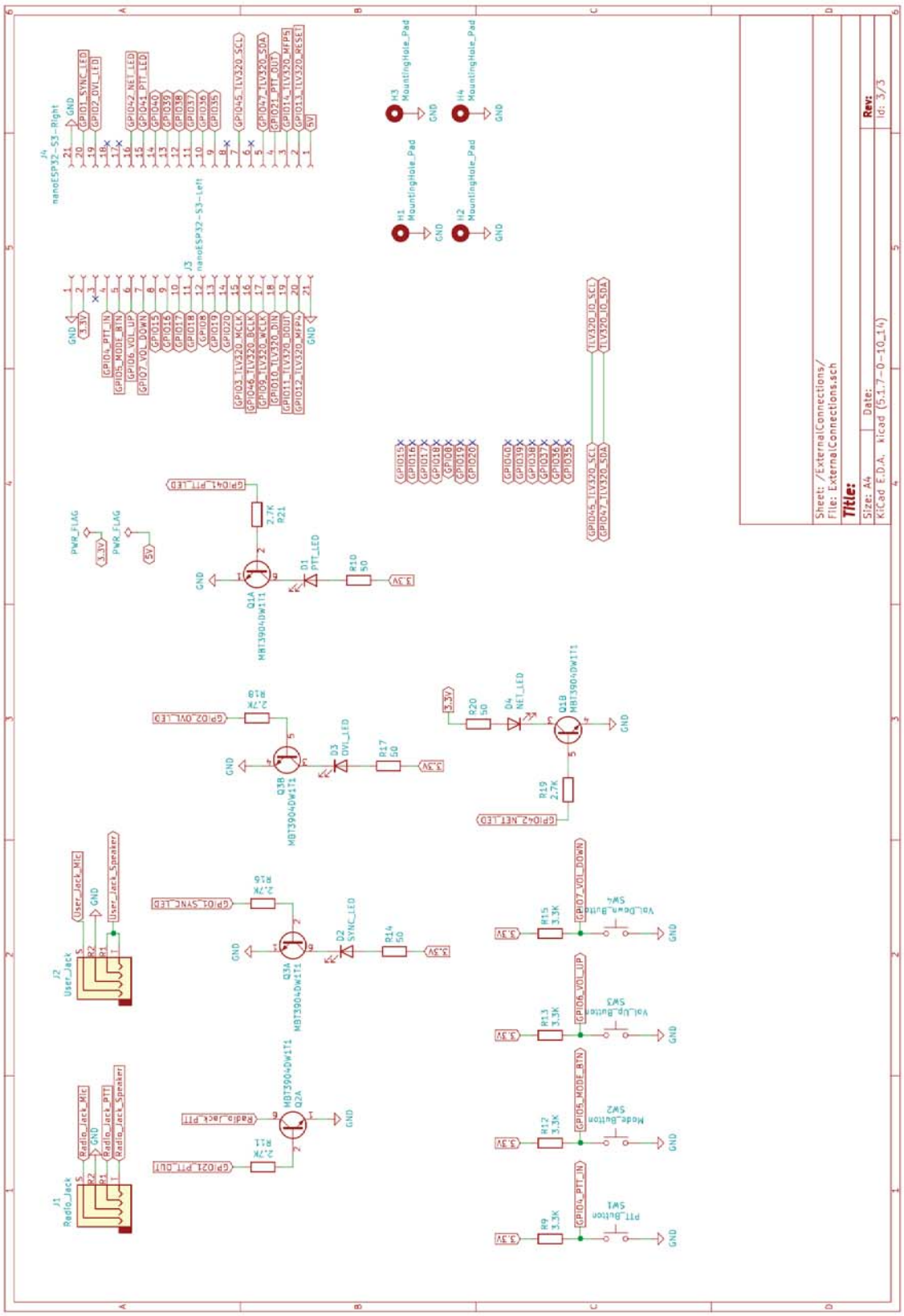

Figure 2: The LEDs and other external interface components on the ezDV schematic.

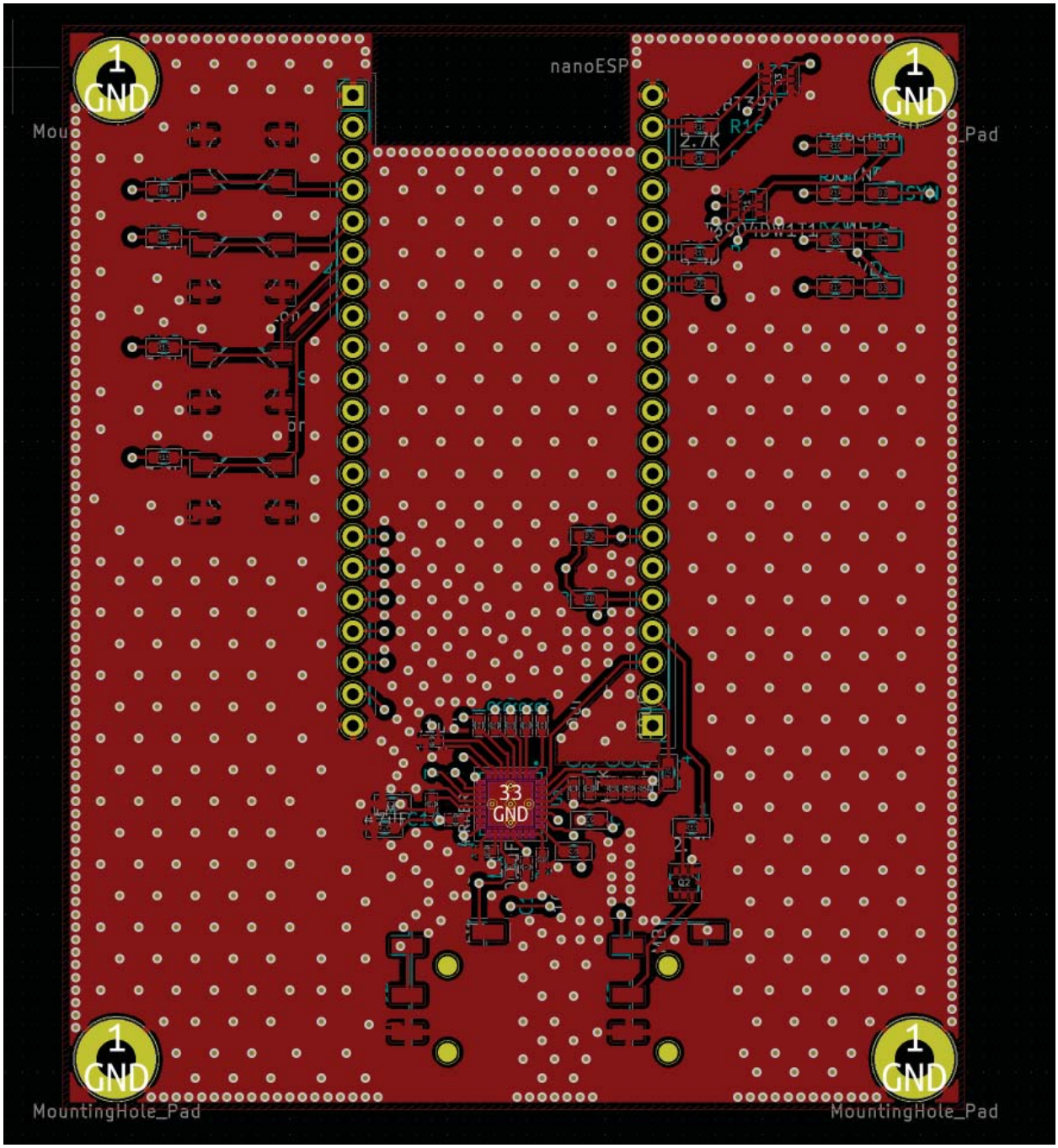

Figure 3: The top signal layer of the ezDV PCB including ground pours.

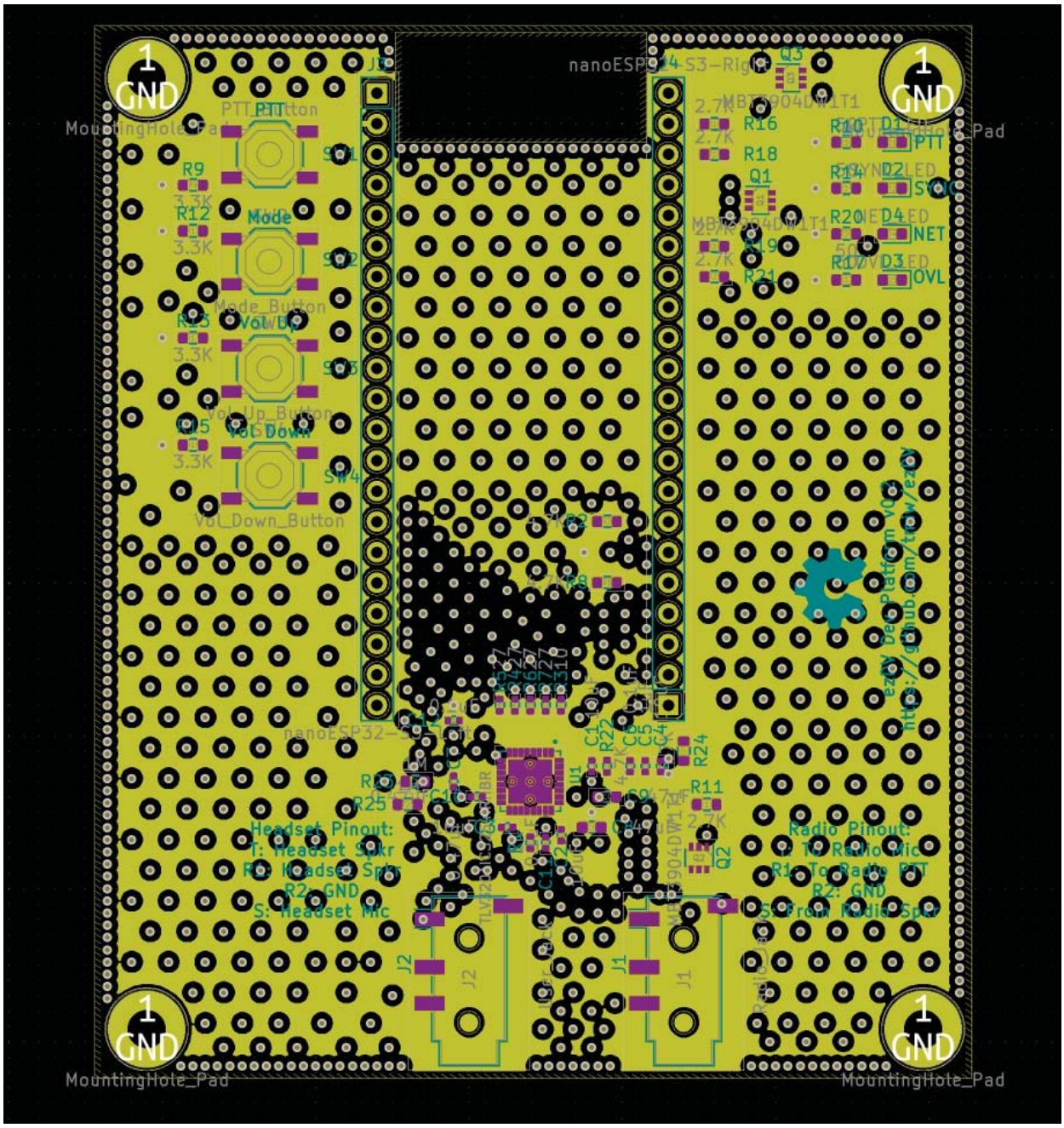

Figure 4: The 3.3V power layer of the ezDV PCB.

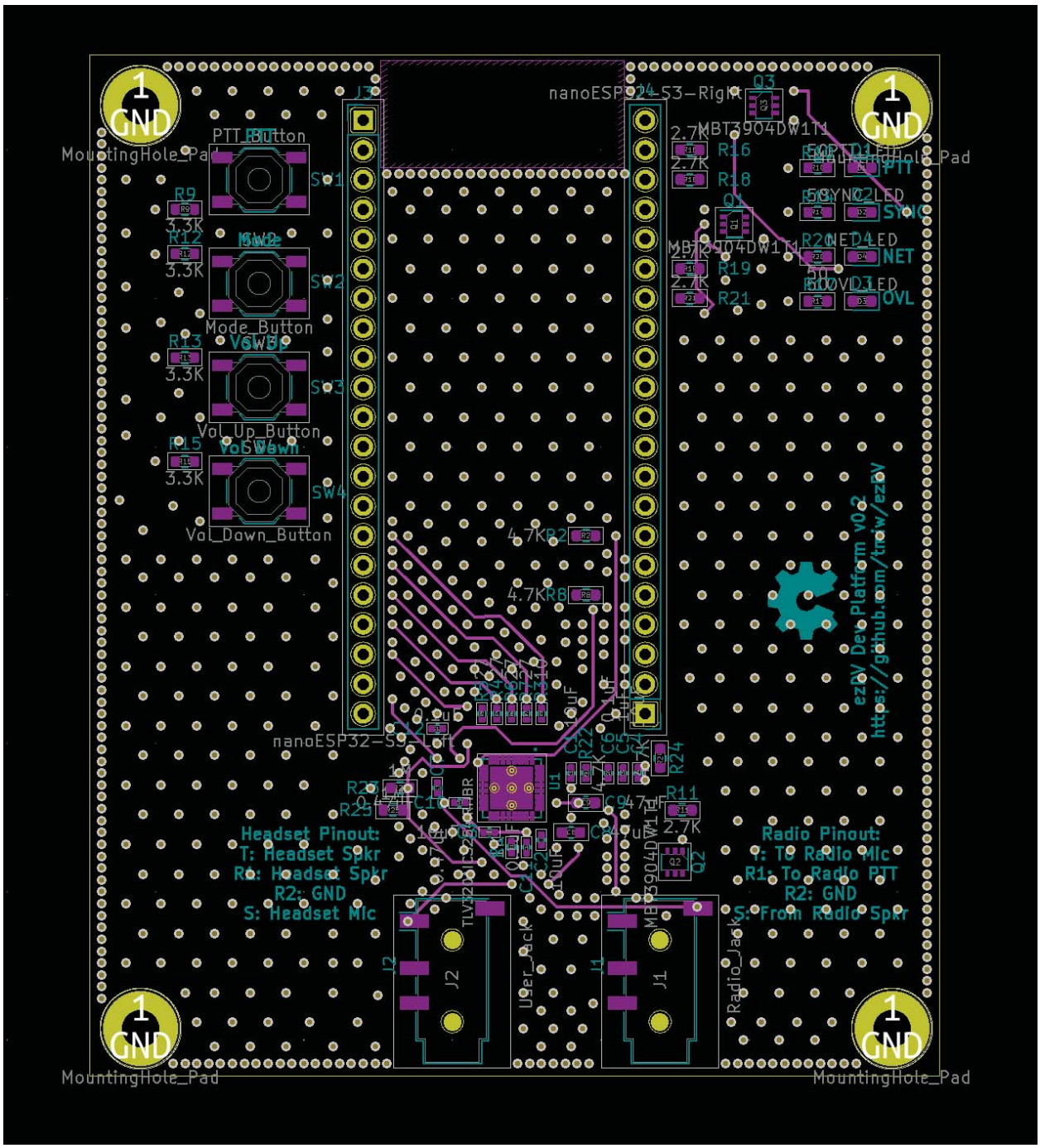

Figure 5: The bottom signal layer of the ezDV PCB.

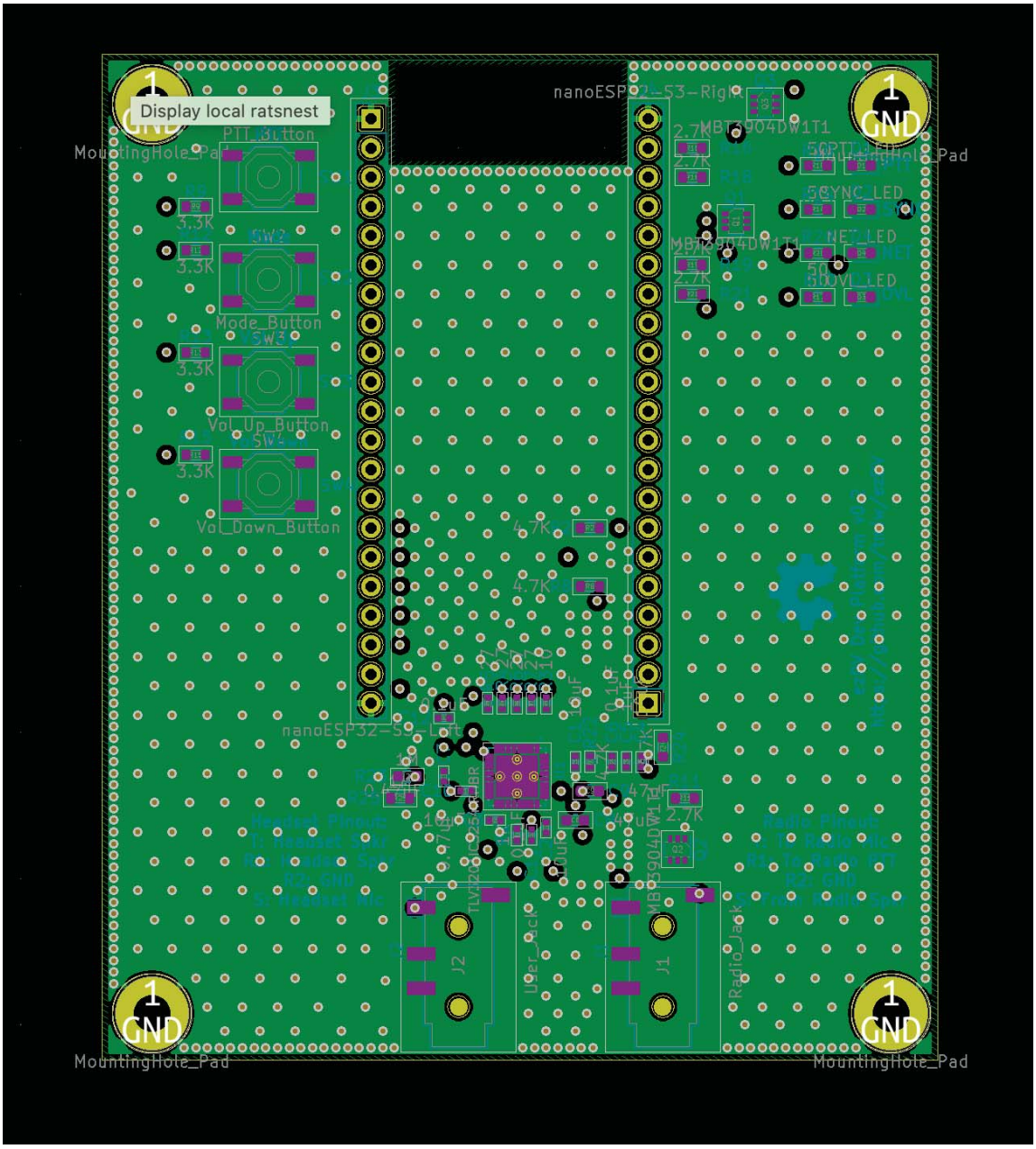

Figure 6: The bottom ground layer of the ezDV PCB.

Upon receiving the new revision boards, it was discovered that performing these changes did not resolve the audio problem. After further research into the TLV320 technical documentation, it was discovered that it had a loopback mode that bypassed the firmware entirely. Enabling this mode demonstrated that audio was being decoded and re-encoded successfully on both channels, indicating a problem in the firmware. Additional debugging of the firmware then found the issue causing the audio failures and thus allowed audio input and output on both jacks to work properly.

Building the Enclosure

The next step after producing a basic working prototype of the firmware and hardware was to build an enclosure to protect it during transport and use. This author purchased a Creality Ender 2 Pro 3D printer to print out prototypes of the enclosure due to its small size (which was good for storage and use in an apartment) and low cost. After some time learning how to use the printer for pre-made designs on websites such as Thingiverse, it was time to learn how to create 3D models. OpenSCAD was especially ideal for someone with a software development background as it allowed designs to be represented in the form of code[20].

One challenge was in determining how to produce buttons that stay with the enclosure (that is, those that don't easily fall out). Building the enclosure such that the buttons were larger on the top and bottom and had a narrow area to allow movement seemed like the obvious choice to enable this to happen (Figures 7 and 8). However, it was difficult to adjust the sizing properly to allow the buttons to freely move yet not fall out.

[20] "OpenSCAD CheatSheet." *OpenSCAD*, https://openscad.org/cheatsheet/.

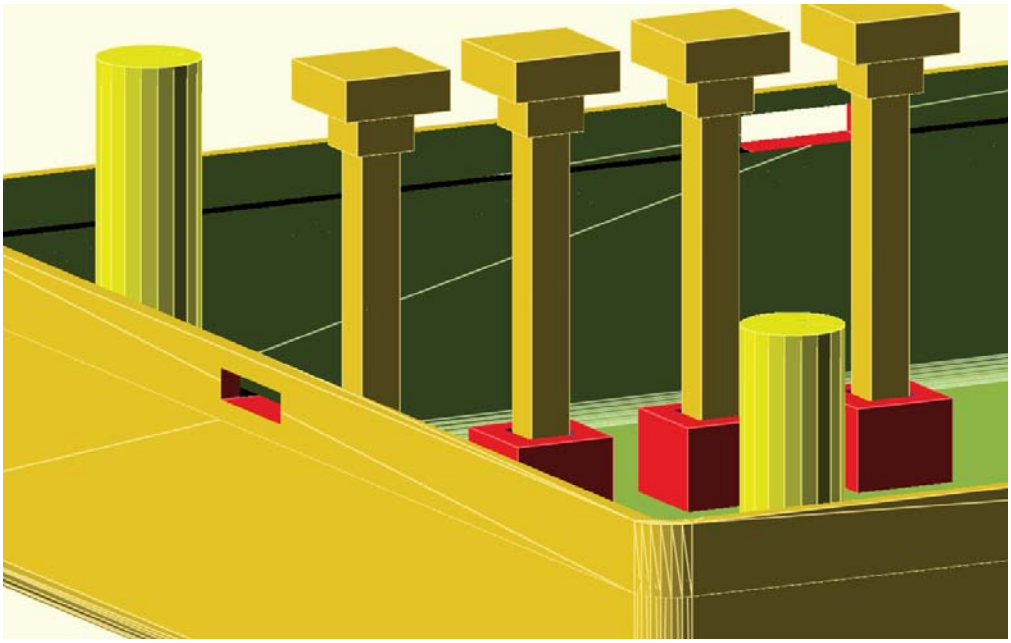

Figure 7: The ezDV enclosure buttons viewed from inside the enclosure.

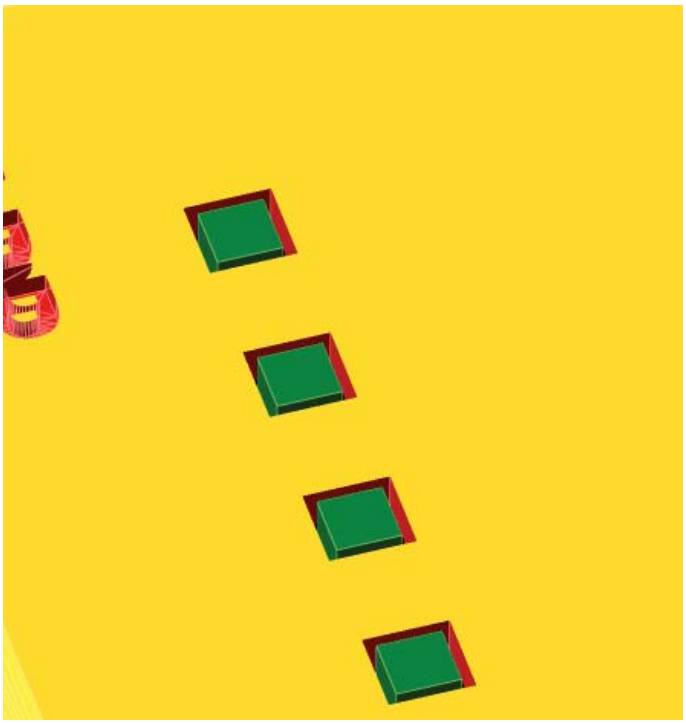

Figure 8: ezDV enclosure buttons viewed from the top of the enclosure.

After updating and printing many iterations of the design (to test various measurements for the buttons), a working version of the ezDV enclosure that allowed the buttons to move up and down and “click” on the PCB buttons resulted. This enclosure was printed using a transparent PETG filament (for added strength) and is shown below:

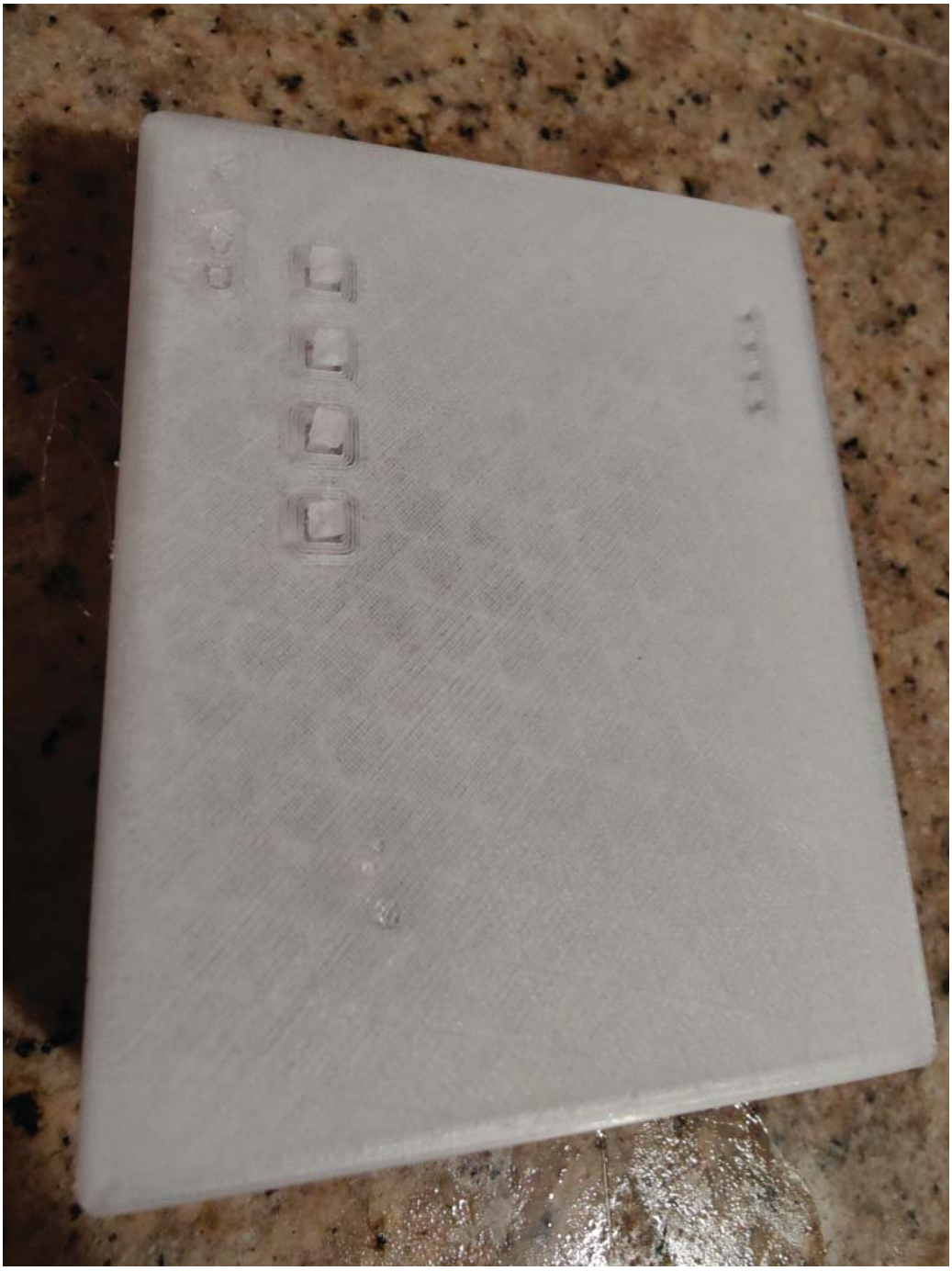

Figure 9: Top view of the 3D printed ezDV enclosure.

Lessons Learned and Next Steps

During the initial development of ezDV, a few things were learned:

1. Hardware development is more accessible than it has been, especially with online PCB assembly houses (such as JLCPCB and PCBWay) that only need a credit card and your Gerber files to produce small runs of boards. In the past, PCB assembly houses only worked with developers if they were willing to produce large runs, which would be problematic if an issue was found in the PCB design after the fact.
2. It's important to definitively rule out firmware bugs before jumping to hardware related issues. While having one audio channel with problems was suggestive of a hardware problem, it wasn't guaranteed to be one in retrospect, especially if the firmware does additional processing of the input or output.
3. The Codec2 library (and FreeDV more generally) are well suited for cross platform use due to the relatively low usage of platform specific code. Even on the PC, audio I/O is done through third party libraries (such as PortAudio and PulseAudio/pipewire) instead of being done directly.

There are also some things that would be good to add in future iterations:

1. Removing the dependency on the nanoESP32-S3 by integrating the ESP32-S3 directly onto the board. Doing so would result in a decrease in board size and cost as well as enable other functionality to be included in the space that was saved (for example, a built-in speaker/mic for analog audio or battery charging).
2. Perform testing using the original ESP32. If FreeDV performance is still acceptable on that part, that would enable use of standard Bluetooth instead of requiring a wired headset, resulting in only needing power to run the board (if also used with a radio that supports wireless connectivity such as the Icom IC-705).
3. Enable wireless connectivity. Some basic code to communicate with the IC-705 was prototyped, but it didn't have adequate reliability. Part of this effort would be to improve said reliability as well as possibly lower memory consumption (as it currently requires use of the PSRAM to run without crashing the ESP32). A Web based configuration interface could also be created to configure various components in the firmware, such as Wi-Fi network information and callsign (for PSK Reporter support).

Additional Information and Resources

ezDV is open source and available on GitHub at https://github.com/tmiw/ezDV for those interested in having their own embedded FreeDV solution. Pull requests are always welcome there, as well as the main FreeDV projects (https://github.com/drowe67/freedv-gui and https://github.com/drowe67/codec2).