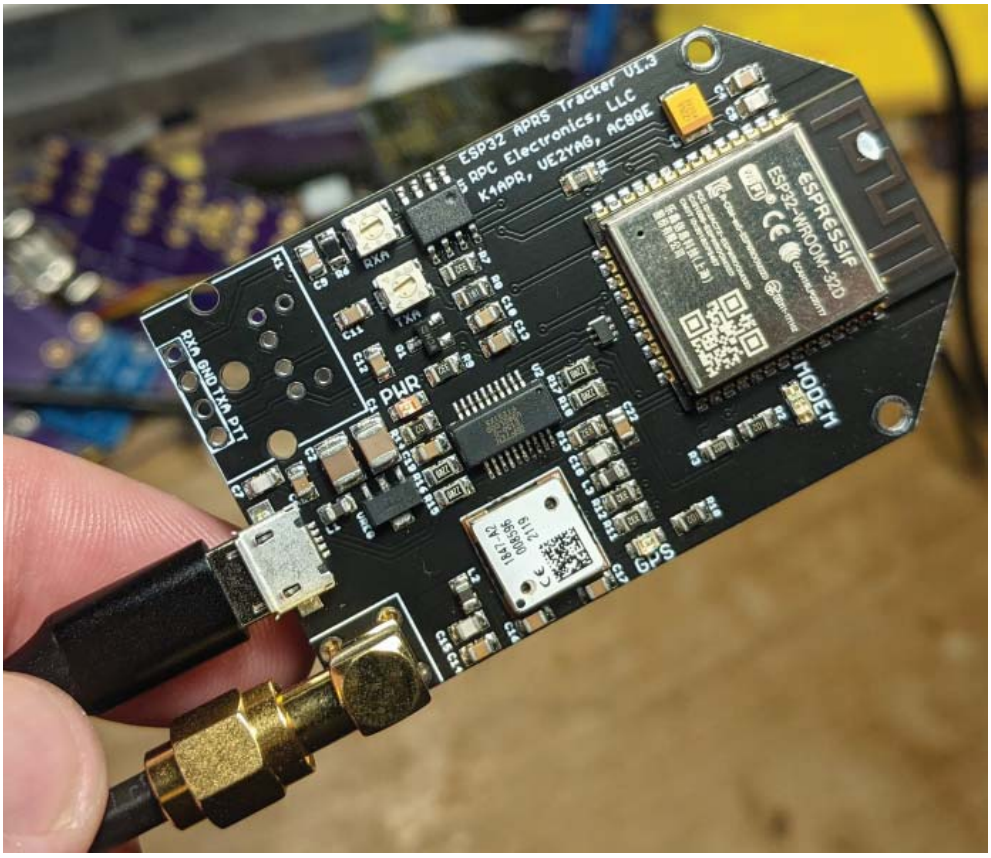
ESP32 Packet/APRS

Creating a Low Cost Tracker

Written By:
Jason Rausch K4APR
Remí Bilodeau VE2YAG



In Memory of Robert Bruninga WB4APR 1948-2022
"The Father of APRS"

Introduction

Nearly ten years ago, Remí VE2YAG and I crossed paths via a short YouTube clip I saw of a handmade APRS display device. I contacted the person who uploaded the video and told him I would love to work with him to create some kind of product around the idea. He contacted me back and said that the device was the work of his friend Remí and gave me an email address to contact him. I emailed Remí and we quickly became friends around our common love for packet and APRS. Fast forward, Remí and I have created several APRS related hardware devices. Some have become formal products that we sold and others have been experiments that either were held internally for our own use or we shelved because we got excited about another idea.

When the ESP32 came around, I was still playing with PIC's and had dabbled in Arduino. I should point out, I am by no stretch of the imagination an embedded programmer. My code is embarrassingly cobbled together from my own badly written routines that take me forever to write and snippets I find online. Remí, on the other hand, is a code wizard. I can throw out an idea and he'll have it written and ready for testing in no time. Anyways, the ESP32 came around and we immediately saw the potential to apply it to an amateur radio data application. Since we had previously focused on APRS, it was only natural that we do the same with the ESP32. This paper is based on that work, hopefully shows what we have accomplished and what we are still working on.

In Memory of Robert Bruninga WB4APR 1948-2022
"The Father of APRS"

Design Goals

We set out to build an APRS modem/tracker with the following design goals:

- Centered around the ESP32's rich feature set
- Integrated 1200 baud AX.25 modem
- USB Serial Interface (FTDI Preferred)
- KISS Support through USB and Bluetooth
- Integrated GPS Receiver
- Op-Amp Audio Design, Avoid Modem IC
- Operates as an APRS tracker autonomously and with a tethered device
- Small, Lightweight and low cost to produce
- Minimal LED Indication (reduce draw when battery operated)
- Easy audio adjustment
- Web Interface Configuration

Most of these design goals are well within reach and most have already been implemented. In this paper, I will do my best to point out where we have achieved a goal, which are yet to be implemented and those that are still being improved.

Choosing a Microprocessor

For years I have been working with PIC microprocessors. They were introduced to me by my involvement in the HamHUD APRS project and I had no idea there was such a powerful device available in such a tiny package. These days, there are many DSPic devices that with some add-on peripherals could do what we needed, but the parts count starts to really add up. Not to mention, debugging any of the sub-systems. Remi and I had used the CORTEX LM3S800 in the original YagTracker. Later we moved onto the LPC1343 and LPC1347 in the ExpressTracker models. While these ARM based processors all worked well at the time for our applications, they too were lacking something…wireless connectivity.

Enter the ESP32. It's truly a marvel of modern microprocessor technology. I'll spell out the full specifications in the next section, but the major advantage the ESP32 had over the other guys was the built in wireless connectivity options. Bluetooth, Bluetooth LE and WIFI 802.11b/g/n. This opens up the endless possibilities for connecting this device to other devices. We're not talking about concentration on internet, as many APRS users have gone to. We're talking about connecting to a modern device, while still using a real radio. Put the radio back into amateur radio.

Not only does the ESP32 have wireless connectivity, but it has a whole host of interfaces and supported protocols making adding on the hardware we need easy!

ESP32 Hardware Overview

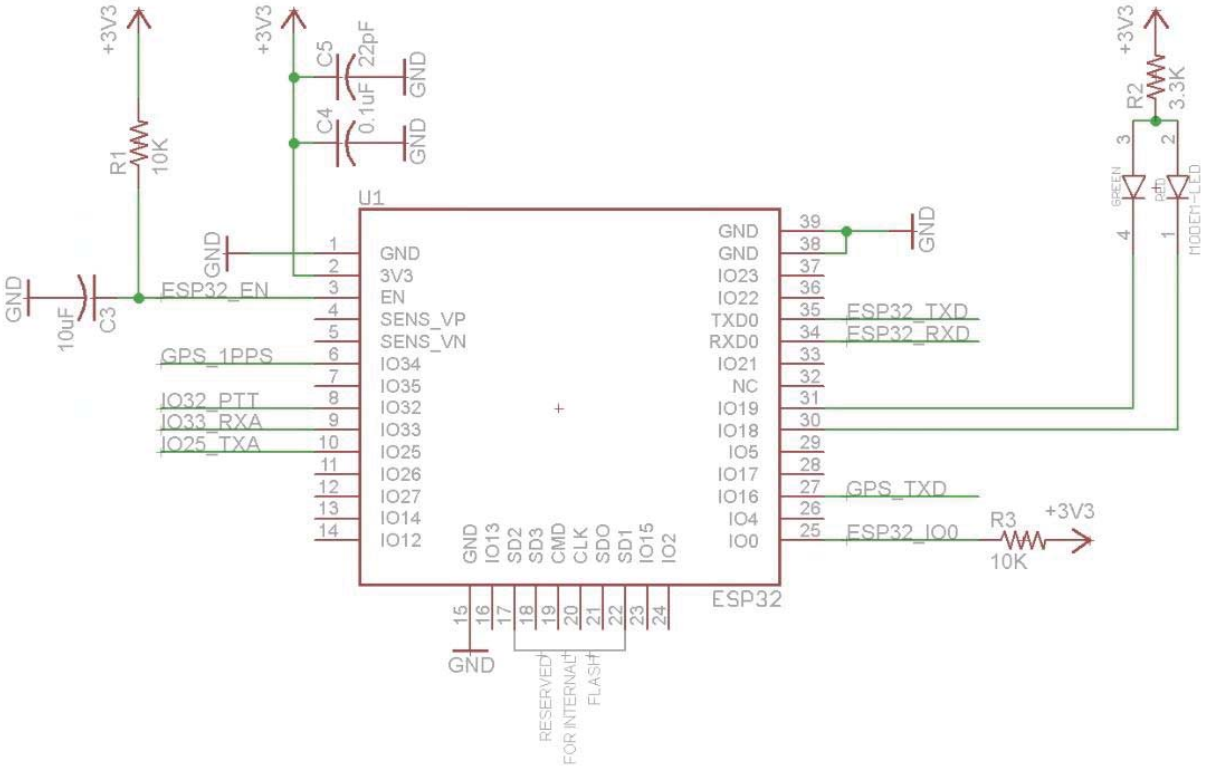
Let's get right into it and talk about the ESP32 hardware itself. The ESP32 is an incredible amount of features packed into a tiny package that averages less than $5 each, single quantity. Here are the specifications:

- 32 Bit Microprocessor
- Two CPU Cores
- 80-240 MHz Clock Speed
- Three Hardware UARTs
- 12 Bit ADC, Up to Eighteen Channels
- Two 8 Bit DAC Channels
- I2C, SPI, SD Card Interface, PWM and SDIO
- 22 GPIO Pins
- 32 MB of On-Board SPI Flash
- Built-In Bluetooth and Bluetooth LE
- Built-In WIFI 802.11b/g/n
- +3.3V Operating Voltage
- Can be programmed using Arduino IDE and ESP32 Support Plugin

In Memory of Robert Bruninga WB4APR 1948-2022
"The Father of APRS"

ESP32 and FTDI USB Interface

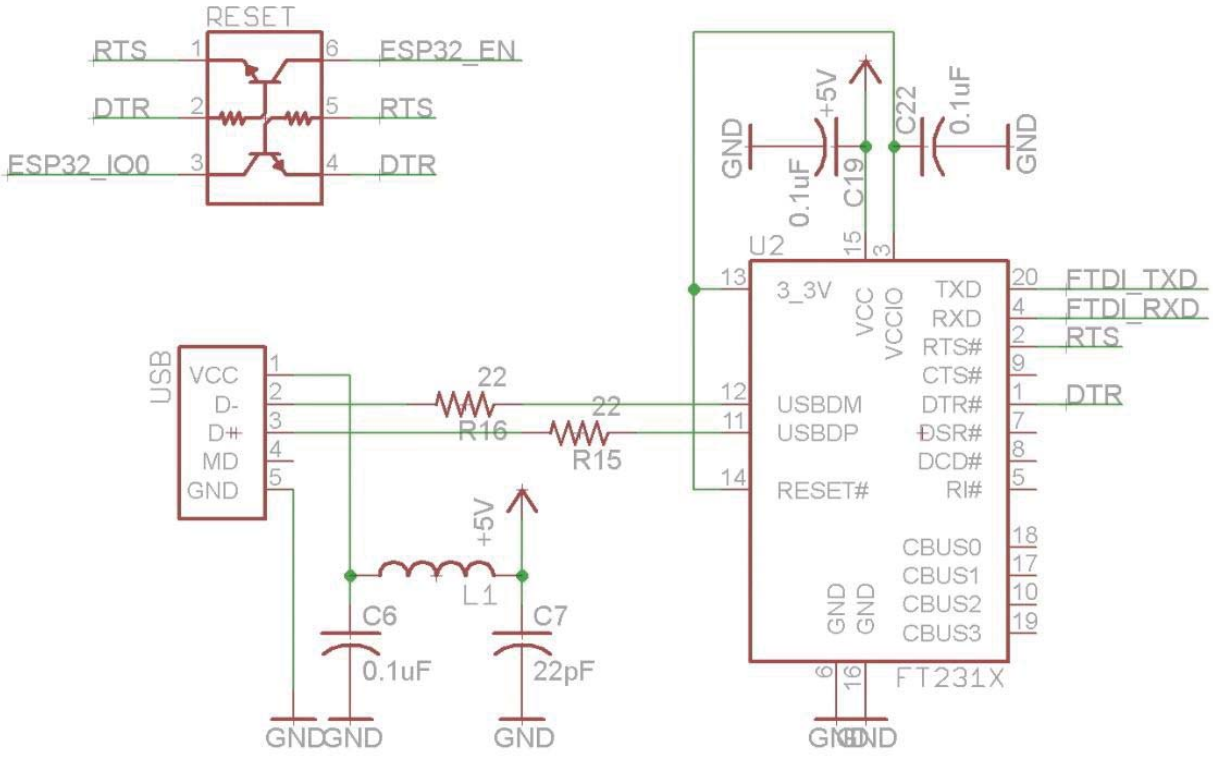
The ESP32 doesn't require a lot of external components to make it work. It is somewhat sensitive to power fluctuations and we found good power filtering with large electrolytic capacitors worked the best to prevent problems. Notably, the 10 uF capacitor on the EN (Enable) line of the ESP32. The 3V3 line also benefited from 0.1uF and 22pF capacitors to handle high frequency transient noise on the power rail.



The ESP32 is a bit finicky about how it is programmed. We used an FTDI FT231X USB-Serial IC that has a full UART, plus hardware handshaking pins. Most notably RTS and DTR lines needed for the self-resetting circuit. This not only prevents the user from having to manually reset the tracker after a firmware upload, but performs the somewhat complicated and time critical line pulsing "dance" it takes to put the ESP32 into a bootload mode for flashing. We started off by using a two discrete transistor setup for doing this, but found that it didn't always work and we would have to reset and start again. In comes the ROHM UMH3NTN single IC with dual NPN transistors and built in base resistors. By switching to this single part, all of the bootload/flashing issues were solved.

In Memory of Robert Bruninga WB4APR 1948-2022
"The Father of APRS"

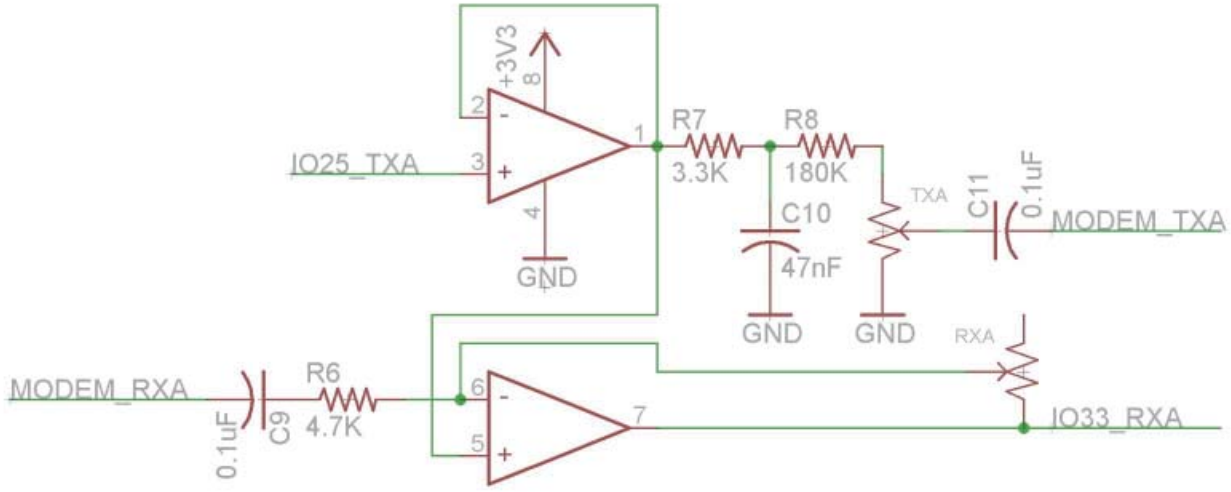
In this section of the schematic you can see the FTDI FT231X single IC solution to USB-Serial communications. The FT231X takes care of the USB heavy lifting, leaving you a TTL serial interface. FTDI drivers are included in just about every major OS now and you rarely have to actually go download them to get things talking. Just plug n' play!



Op-Amp Audio Section

When I went to Remi about the idea to build this, he already had a working op-amp based modem design he was playing with on a couple of other projects. It seemed like the perfect fit for our new APRS tracker. The part we started off with was a Microchip MCP6002. With the global shortage, the MCP6002 soon became nearly impossible to obtain, so we had to look for a replacement. We found the Microchip MCP6402T was close to the specs of the MCP6002 and a perfect pin-for-pin replacement. The part is a dual op-amp, reducing down to a single part, with some additional caps and resistors to form the output low-pass filter on the transmit side and a simple DC blocking circuit on the receive side.

Two 10K single turn trim potentiometers are used to control the transmit level and limit overloading to the receive section. IO25_TXA is a direct connection from the ESP32 DAC to the op-amp. MODEM_RXA is a direct connection from the radio port to the op-amp.



This design along with Remi's method has yielded a phenomenal decode rate of even poorly encoded packets. Remi has four individual decoding algorithms operating simultaneously to potentially catch any scenario and hopefully pull out a CRC-passed packet. The first two algorithms are correlator methods and the second two are filter methods. The correlator algorithms can be described as self-multplicative correlator with a flat and high frequency boost filter. The filter algorithms use a comparison of sine, cosine/sine of the 1200 Hz and 2200 Hz tones to detect which is which.

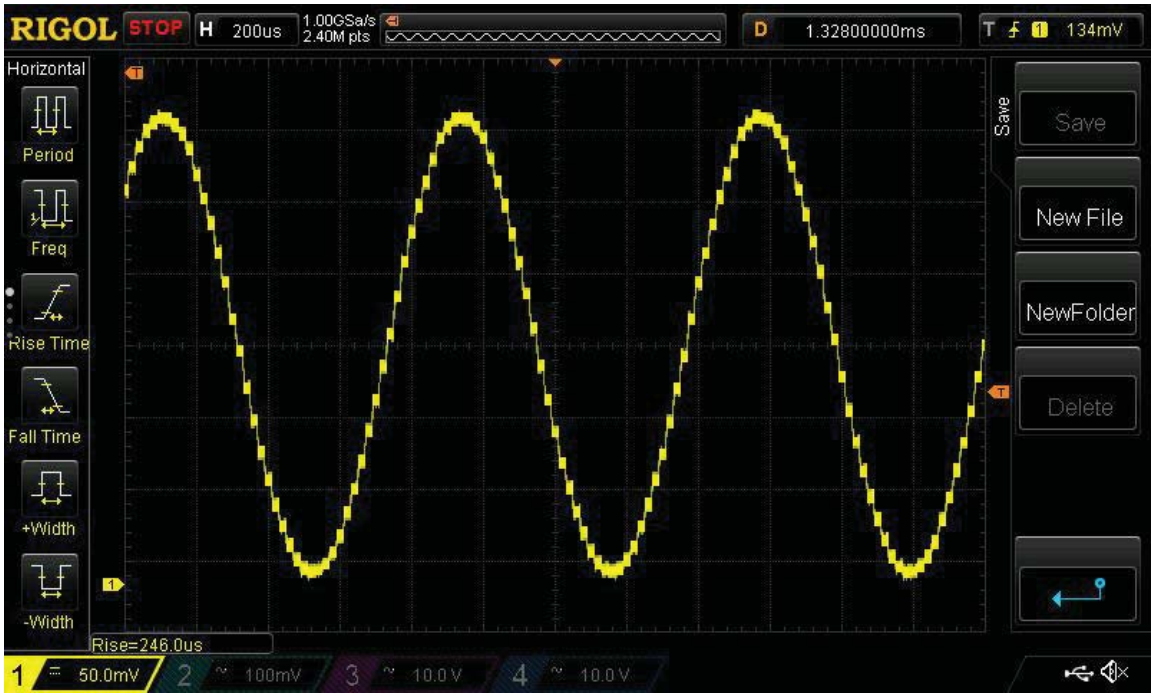
In Memory of Robert Bruninga WB4APR 1948-2022
"The Father of APRS"

We Are Having Some Problems

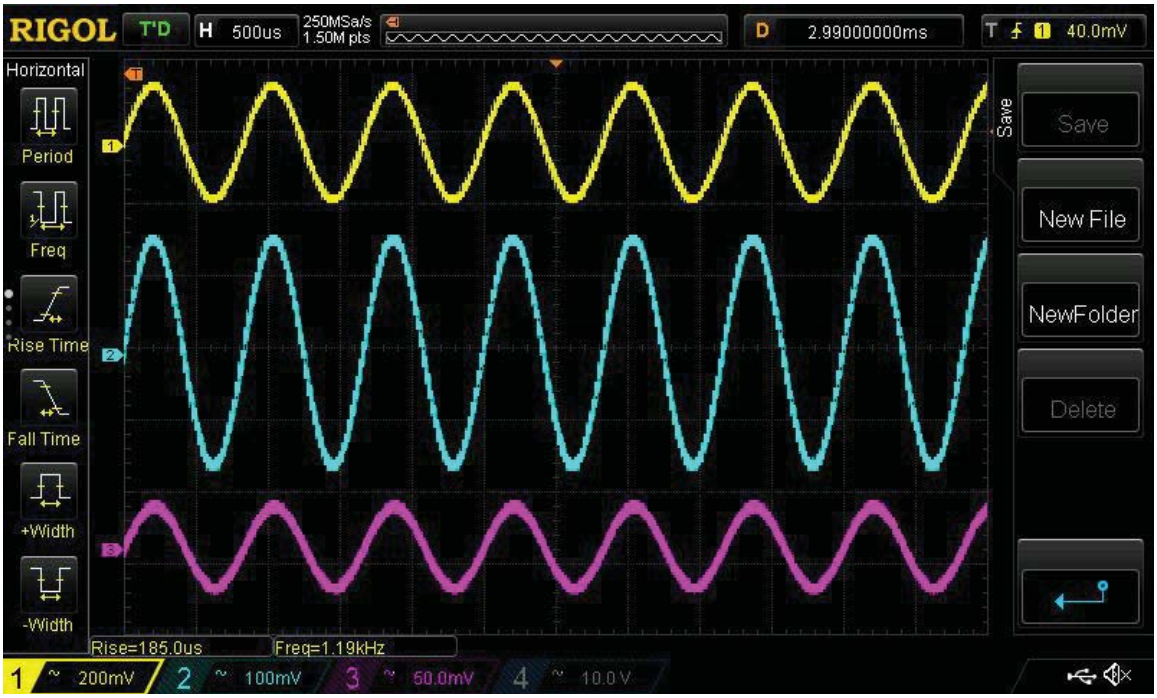
For the transmit side, we are having problems with the audio being decoded by just about any hardware I test with. This includes standards like the Kenwood D7/72/710 radios, Kantronics KPC-3, Argent Data Systems Tracker 2 and the NinoTNC. Some initial tests showed that the waveform might be the issue, having a rough “raspy” sound to it. Remi was able to tweak a few things in his code to clean this up, but yet decode of the transmitted packets have not been successful. I should note that Remi has a hand-built version of this circuit using through hole components and doesn't seem to have this problem. We can’t nail down if the issue is with hardware, firmware or a combination of the two.

This o-scope shot shows the audio stepping out of phase. It was determined that the DAC was inverting some of the waveform, causing this. Some quick changes to the code fixed the problem.

The fix resulted in this o-scope capture.



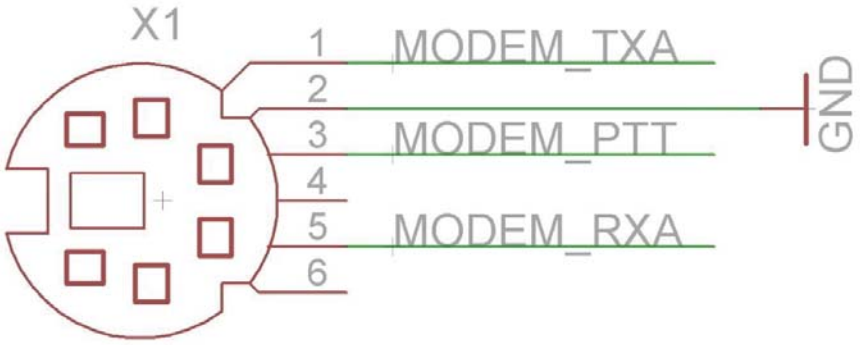
This is a comparison scope view of the raw output from the DAC, output of the op-amp section and just past the low-pass filter before going through the 10K transmit level potentiometer.



In Memory of Robert Bruninga WB4APR 1948-2022
"The Father of APRS"

Radio Interface

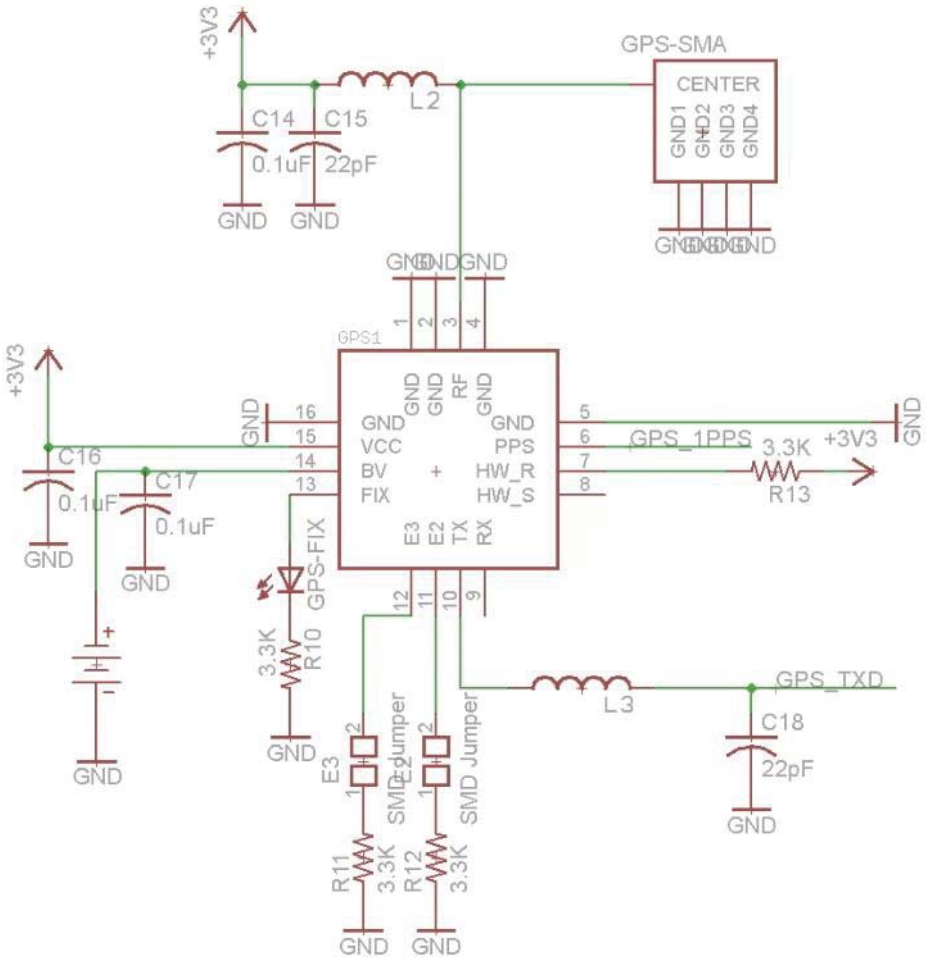
We employed the Mini DIN interface standard as the interface to the radio. There is also a four pin header on the PCB, for applications where the Mini DIN might not be optimal.



On-Board GPS Receiver

When looking for a low-cost, high performance GPS for the tracker, there were several options. While not cheap, the Trimble Copernicus II was an excellent option and proven. I also had a lot of experience with this GPS chipset when I put it in my RTrak all-in-one APRS tracker. It was a great option, but just too expensive. Another popular option is the UBlox line of receivers. Low cost, decent performance, but sometimes hard to source.

No, we needed a better option. I decided to fall back to a gem I had found several years ago, but had mixed results with. The Antenova M10578-A3. At just over $20 USD, it's an excellent performing chipset, small, 3.3V power capable and perfect for our needs. The only real configuration is making sure the E2 and E3 lines are at the correct levels to set the baud rate to 4800 baud. In the future, we could switch up to 9600 baud. A simple change to the solder jumpers on the back of the board and we're talking faster!



In Memory of Robert Bruninga WB4APR 1948-2022
"The Father of APRS"

Configuration

As of the time writing this, configuration is done in one of two ways:

Command Line

Much like a DOS or Linux terminal interface. Simple text commands that return current values or when accompanied by arguments can commit changes to settings.

Text Configuration File

Remi has implemented an in-terminal text editor that allows you to create a complete configuration file that is loaded on power-up/boot. The structure is much like a DOS batch file with section [xxx] headers and simple x=y settings under the header.

Example (not complete):

```
[tnc]
callsign=k4apr
tz=0
verbose=0

[gps]
baud=4800

[kiss]
enable=1
persist=63
slottime=1
txdelay=35
dac_zero=128
```

Web Configuration

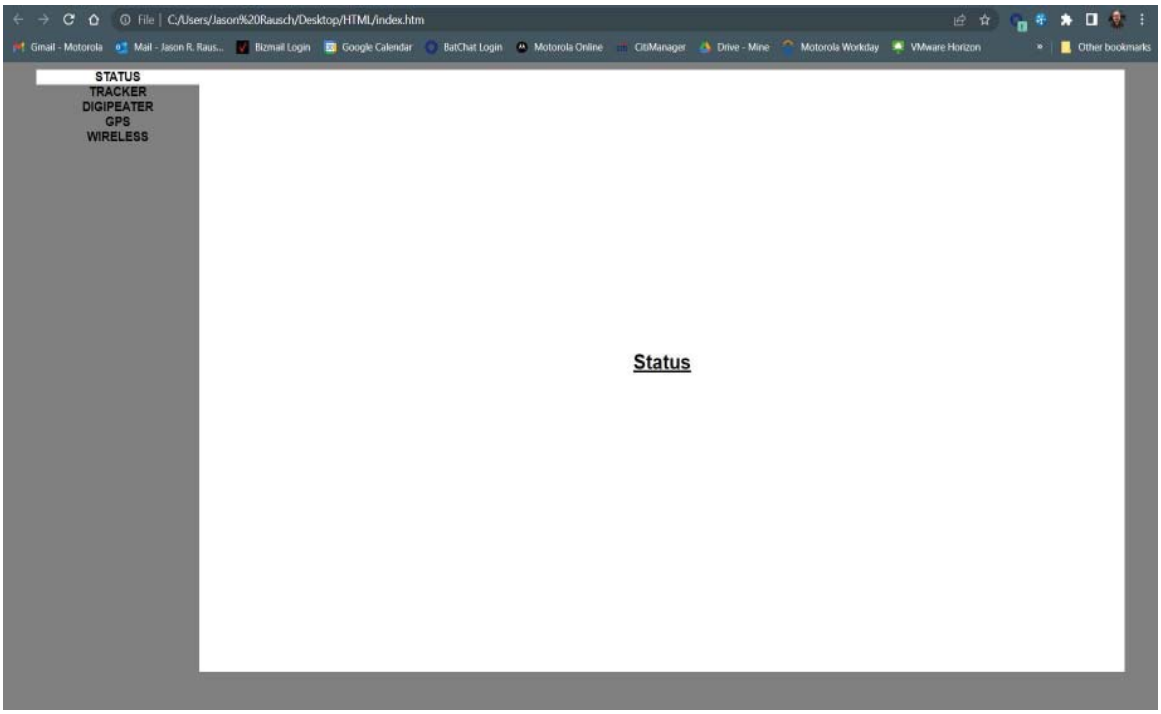
A future option will be the ability to connect to the device via WIFI and using a web browser navigate to 192.168.10.1 to be presented with a web interface for configuration of all settings.

What's Left to Do?

1. We have to get to the bottom of the transmit audio issue. We have been fighting this for a while. Numerous hardware revisions, hacking up older hardware to try "simple fixes". We're open to any suggestions!

2. Web Interface for configuration. We know that if this is going to be a product that people buy, they are going to want an easy way to make configuration changes, especially when in the field/on the go. The web interface is really the only solution for this.

   This is a quick mock-up site written in HTML.



Conclusion

We have made a lot of progress, but there is still work to do. Some might ask "why?" when APRS seems to be waning. Our argument is, APRS is coming back. We're seeing an increase in activity all over the place and we think there is a need for a product like this.

In Memory of Robert Bruninga WB4APR 1948-2022
"The Father of APRS"