APRS as a Remote Control, Step One

Recently there was an email sent in to the SOARA website asking for help. The writer was asking for their high school competing at the college level on a rocketry contest where one of the requirements is the rocket can receive and respond to APRS. I kept thinking about it and I think I can get this working.

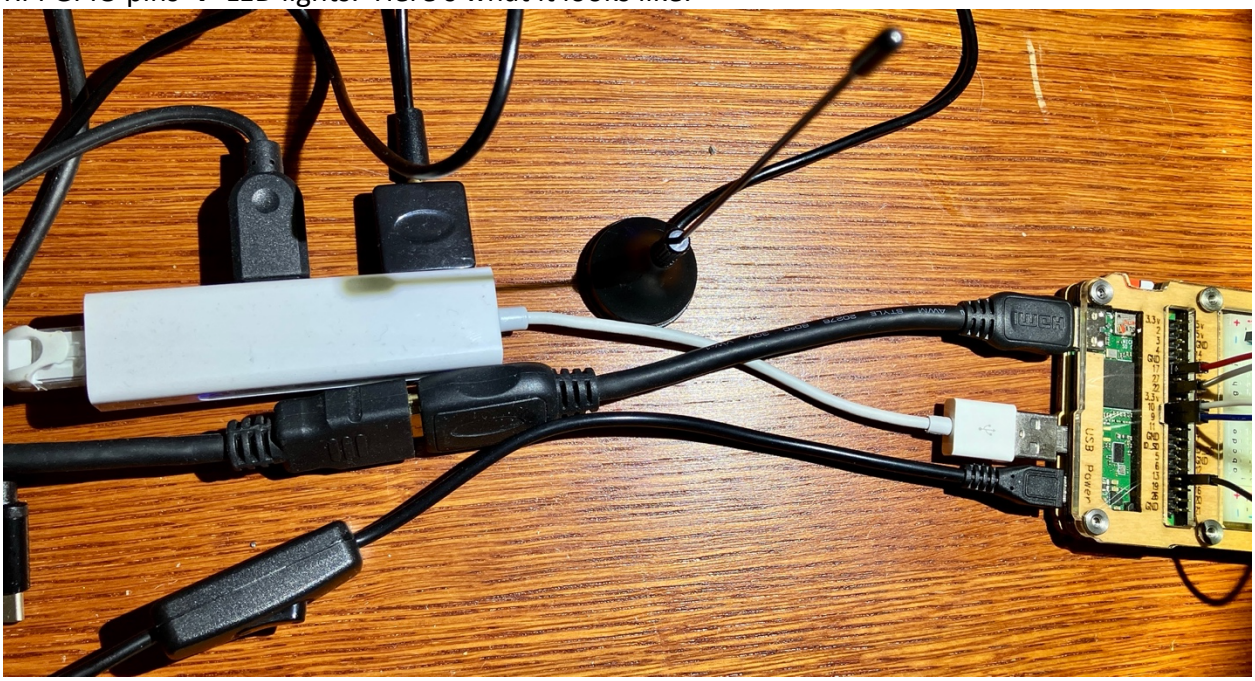
The way I see to start this prototype is to use a RTL-SDR receiver, that connects to a Raspberry Pi to light up LEDs. The software would be rtl_fm that feeds into direwolf software and that output is filtered by grep and piped into raspi-gpio command. The ultimate trick will be calibrating the tiny, cheap & low quality. This has been done before by using NOAA1 as a source. I don't know if *I* will figure it out in time to help this high school, but it's a fun project to leave on my desk to play with when I need a break.

For this article, I am just going to review the prototyping setup I put on my desk to manually turn LEDs on and off with the Raspberry Pi. In future articles, I will show how to use RTL-SDR software to listen to APRS, how to use direwolf software to decode APRS, how to redirect that decoded APRS into programs on a Raspberry Pi, how to fine tune RTL-SDR and last, how to optimized the RPI software and the hardware around it for lightness, speed and low power consumption.

This first article isn't very radio and assumes a knowledge and background with Raspberry Pi and Linux. I'm happy to discuss or review this at a SOARA Saturday or some other time.

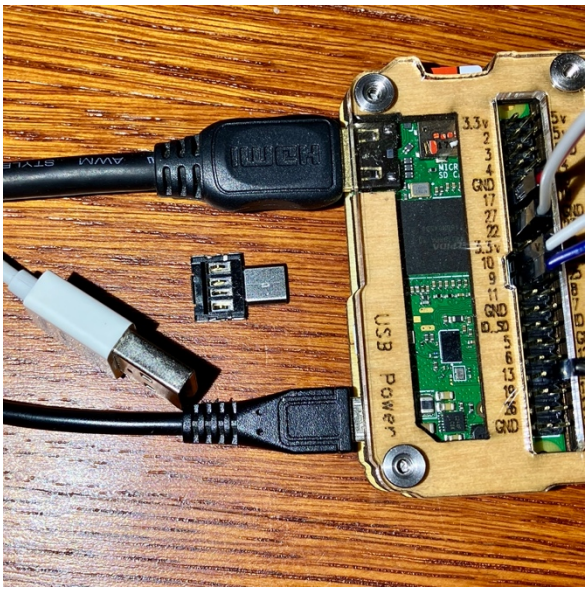
Hardware:

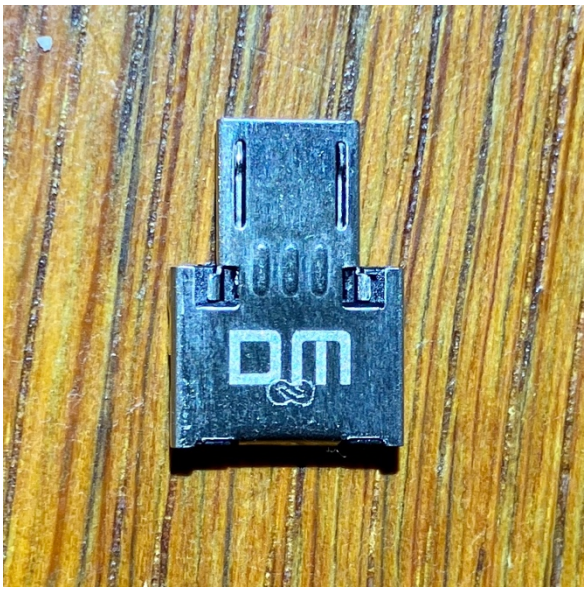
Here's that the hardware path I am going to use for prototyping:
APRS Antenna → RTL-SDR USB dongle → USB A to USB micro adapter → Raspberry Pi Zero → RPI GPIO pins → LED lights. Here's what it looks like:



The picture above is hard to interpret, so let me explain. The Raspberry Pi Zero W is on the right in a case with a breadboard. The top black cable connecting to the RPI is a HDMI to mini HDMI for a monitor. The bottom black cable is micro usb is for power, and if you follow it to the left, you'll even see a power switch. The middle white cable is a usb micro adapter connecting to a usb hub. Connected to that usb hub is ethernet for networking, a usb cable for keyboard and mouse and last is a very cheap RTL-SDR USB adapter and you can see a small antenna to its right.

If you unplug the usb hub, you can see I am using the cutest, tiniest usb adapter, about the size of a nickel – for those that remember coins.

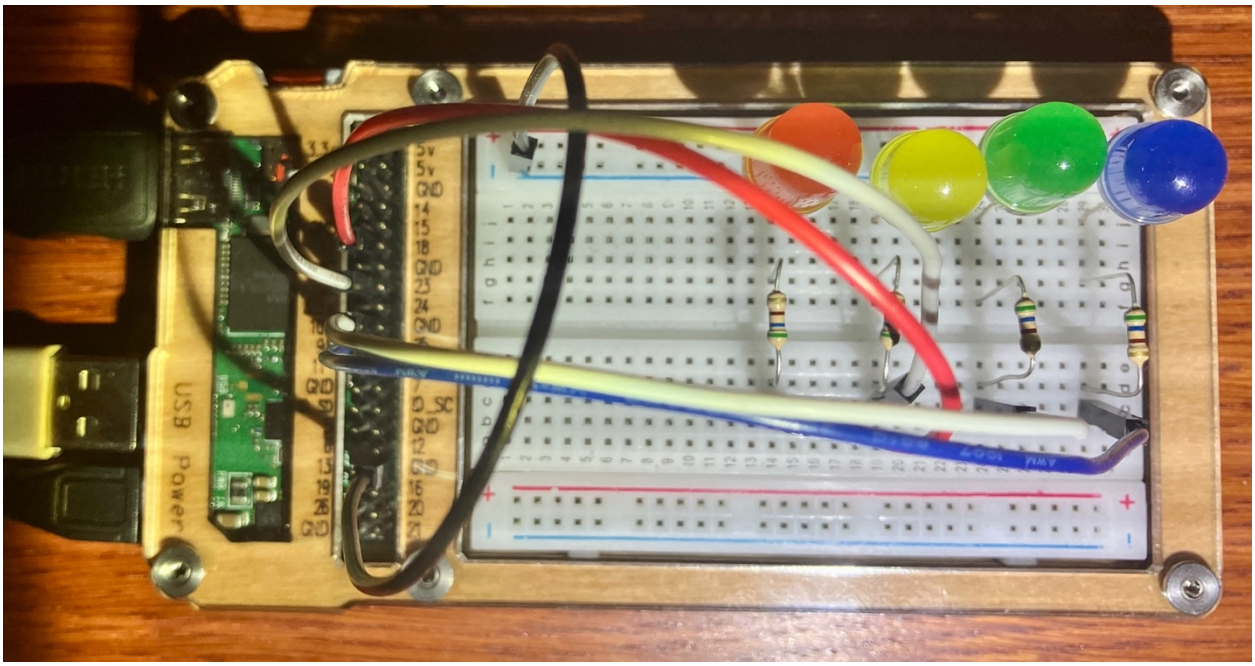




What is RTL-SDR? (taken from the footer on https://www.rtl-sdr.com/)

> *"The RTL-SDR is an ultra cheap Software Defined Radio (SDR) based on DVB-T TV tuners with RTL2832U chips. The RTL-SDR can be used as a wide band radio scanner. It may interest ham radio enthusiasts, hardware hackers, tinkerers and anyone interested in RF."*

The tiny black adapter and antenna in the pictures above, I think were a free gift with an order from somewhere. For the real project, you would want to use a better adapter, that is shielded and an appropriate antenna for 144.390MHz for north American APRS.



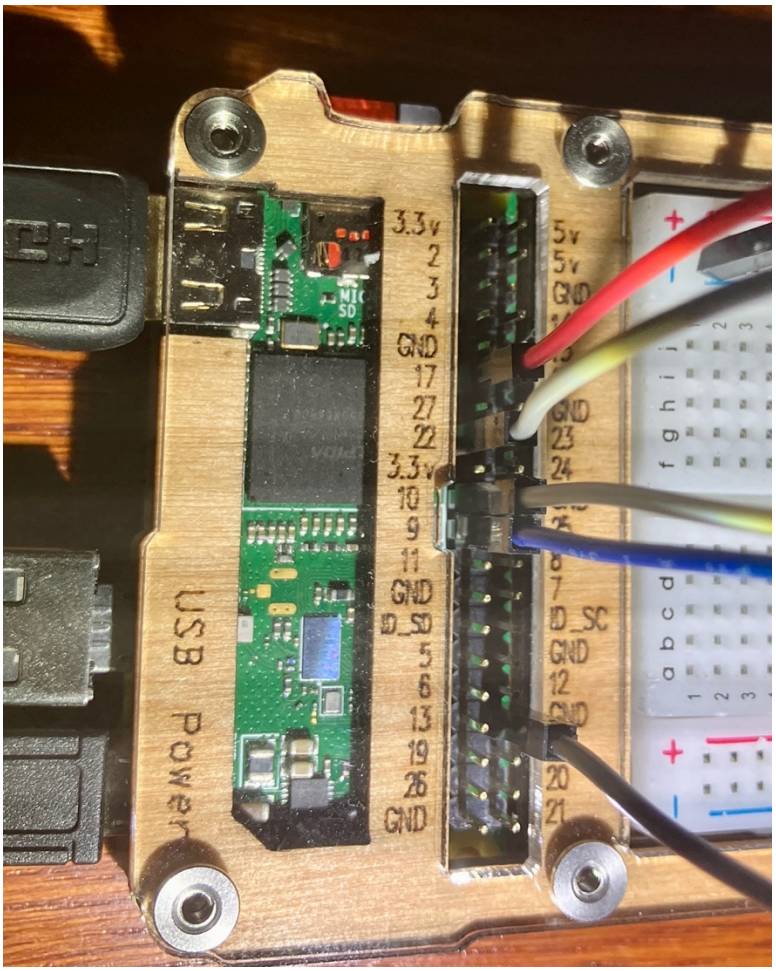
The Raspberry Pi Zero is in a case that holds a breadboard. I have female to male jumper wires to connect the GPIO pins for the RPI to the bread board. The nice thing about this case is it labels the GPIO pins for you as, they don't map as you might expect. If you don't have this, on Raspbian OS (more on this later), you can run the command `pinout` to show the pinout for your RPI.

I am using pin 34 as a common ground (black wire) and all the led cathodes (short legs) are connected to this ground. I then have jumpers for the different color LEDs. There isn't much rhyme or reason for my pin choices.

- Pin 11 = GPIO17 = Red
- Pin 22 = GPIO22 = Yellow
- Pin 19 = GPIO10 = Green
- Pin 21 = GPIO9 = Blue

Each GPIO pin is then connected to a 560 ohm resistor. For these 10mm ¼W LEDs, that's overkill for 3.3V. 50 or 100 ohms resistors would be better, but I think I used all my smaller resistors on another project, and that's what I had on hand. ☺

Here is a better picture of the jumpers connection to the RPI.



Software

The big picture of this project will be to chain existing software together to make all of this happen. At the high level, it will look like:
RTL-SDR to get NOAA1 output, and write the settings for fine tuning → RLD-SDR to listen to APRS → direwolf to decode audio into text → Pipe to output to a script or write our own program to take actions based on APRS output.

To setup the Raspberry Pi, I used the Raspberry Pi Imager software to install the latest Raspbian OS (version 11, "bullseye") on a 32GB SD card. The software also let's you configure your wifi access and enable ssh. After booting it up and logging in, here's the software I installed and the command to do it:

```
sudo app-get update ; sudo apt-get install vim git mlocate rtl-sdr direwolf
```

This will do an update of the list of available software, then it will install vim (text editor), git (version control), mlocate (locate files on the system), rtl-sdr (the radio receiver) & direwolf (APRS decoder).

While we have the software we need to start, let's just start very simple to make sure our hardware setup works with basic commands given to us with the operating system.

For testing the LEDs, the software is already installed with Raspbian. You can use the `raspi-gpio` command for testing.

First, you need to set the GPIO pin for doing output. Here's an example of how to do set the red led (GPIO 17) to do output:

```
/usr/bin/raspi-gpio set 17 op
```

After running this, you can check on the setting for red led with this command:

```
/usr/bin/raspi-gpio get 17
```

That should now output:

```
GPIO 17: level=0 fsel=1 func=OUTPUT
```

With that in place, you are now ready to turn on the led, wait 5 seconds and then the next command is to turn the same led off. We do this by setting the pin "high" (dh) and then setting it "low" (dl).

```
/usr/bin/raspi-gpio set 17 dh ; sleep 5 ; /usr/bin/raspi-gpio
17 dl
```

If this doesn't work, and there were no errors shown on the screen, it's probably time to check your wiring. After this, it's also time to check any other LEDs you set up, using the same commands, but different GPIO numbers. If you happen to use the same LED colors and GPIO pins, you can use this quick shell script I put together at https://github.com/TechTamer/RandomTips-n-Scripts/blob/main/led-functions.sh

(I should make this it's own repo)
Git clone <repo>
Cd <repo>
Bash led-test.sh