Dire Wolf Radio Interface Guide

First rough draft - Jan. 2021

More detailed information about gpiod for PTT control. - March 2024

> There is a surprising amount of confusion about interfacing Dire Wolf with radios. Perhaps it is because the documentation is scattered in different places. Here, I have attempted to gather it all together in one place, hopefully in an easy to use format.
>
> Please share what you have discovered so others can benefit from your experience. The best way would be to open an issue in the "direwolf-doc" project (not the normal "direwolf" project) so nothing slips through the cracks.
>
> I'm especially interested in what you found when using soundcards and USB PTT control built into some modern rigs.

Dire Wolf contains modems used to send Packet Radio & APRS digital data through ordinary voice transceivers.

Just for fun, you might want to try APRS with acoustic coupling. Place a receiver near your laptop computer and use the built-in microphone. Transceiver VOX could be used to transmit when the computer makes a sound. Yes, this actually works, just to prove that it can be done.

A much better solution is to use wires for more predictable results and immunity from ambient sounds.

Audio & PTT

There are numerous ways to achieve this. If you have been using your computer to run NBEMS (FLdigi, etc.), PSK-31, WSJT-X (FT8, etc.), RTTY, or SSTV, you probably have the necessary connection already.

This guide contains information on various types of possible interfaces, configuration examples, and troubleshooting.

Contents

1 The Easy Way

If you don't have a suitable interface already, the easiest way is to use a USB audio interface specifically designed for this purpose. It has audio in for receive, audio out for transmit, and PTT control to activate the transmitter.

NEED PICTURE

Some examples are:

- **DINAH** https://hamprojects.info/dinah/
- **PAUL** https://hamprojects.info/paul/
- **DRA-30** http://www.masterscommunications.com/products/radio-adapter/dra/dra30.html
- **RA-35** http://www.masterscommunications.com/products/radio-adapter/ra35.html
- **DMK URI** http://www.dmkeng.com/URI_Order_Page.htm
- **RB-USB RIM** http://www.repeater-builder.com/products/usb-rim-lite.html

I've only used DINAH, in the list above, so I can't vouch for proper operation of the others.

DINAH has a 6 pin mini DIN connector, same as the "data" (external modem) connector found on some Kenwood, Icom, and Yaesu transceivers. A suitable cable is included. This is ideal for 9600 bps operation.

☞ **Be sure to cut the JP-3 jumper so the COS signal, from the radio, does not shut off the transmit audio.**

PAUL has the same type of connector, and pinout, found on most traditional TNCs and trackers so you can use existing or pre made radio-specific TNC cables.

Many of the others were originally intended for ALLSTAR voice over IP and have different connector styles or pinouts. Check the pinouts carefully if using one of the others.

If you are using a Raspberry Pi, also consider the DRAWS. It sits right on top of the Raspberry Pi and has:

- Two radio interfaces.
- Voltage regulator so Raspberry Pi can be run from 12 volts.
- GPS for precise timing or building a tracker.
- Battery backed real time clock.

Visit http://nwdigitalradio.com/draws/ for more information.

2 Quick Start Guide for USB interfaces with PTT control

This will get you up and running ASAP for the most common case of a single USB audio adapter.

A later section will discuss using multiple audio interfaces so multiple radios can be used.

2.1 Linux

Run the **cm108** utility. You should see something like this:

```
    VID  PID   Product                      Sound                ADEVICE        ADEVICE          HID [ptt]
    ---  ---   -------                      -----                -------        -------          ---------
**  0d8c 0012  USB Audio Device             /dev/snd/pcmC2D0c    plughw:2,0     plughw:2,0       /dev/hidraw0
**  0d8c 0012  USB Audio Device             /dev/snd/pcmC2D0p    plughw:2,0     plughw:2,0       /dev/hidraw0
**  0d8c 0012  USB Audio Device             /dev/snd/controlC2                                   /dev/hidraw0

** = Can use Audio Adapter GPIO for PTT.
```

Devices marked with "**" can use CM108/119 GPIO pins for PTT control.

Edit the sample "direwolf.conf" configuration file.

a. Find the ADEVICE line and add the audio device name found above. Example:

    ADEVICE plughw:2,0

b. Find the PTT example section, for channel 0, and add this or uncomment an existing line:

    PTT CM108

    Do not add a specific device path. It will be determined automatically.

2.2 Windows

Run the cm108 utility. You should see something like this:

(need example)

Edit the sample "direwolf.conf" configuration file.

a. Find the ADEVICE line and add the audio device name found above. Example:

    ADEVICE USB

When direwolf starts up, it will produce a list of the audio devices available. Using the associated number is not recommended because the number can change as new devices are added or remove. It is best to use some unique substring of the name listed.

b. Find the PTT example section, for channel 0, and add this or uncomment an existing line:

```
PTT  CM108
```

Do not add a specific device path. It will be determined automatically if there is only one USB audio adapter. The multiple adapter case is explained in a later section.

2.3 Other operating systems

CM108/CM119 GPIO PTT is not available for other operating systems such as Mac OSX or BSD Unix.

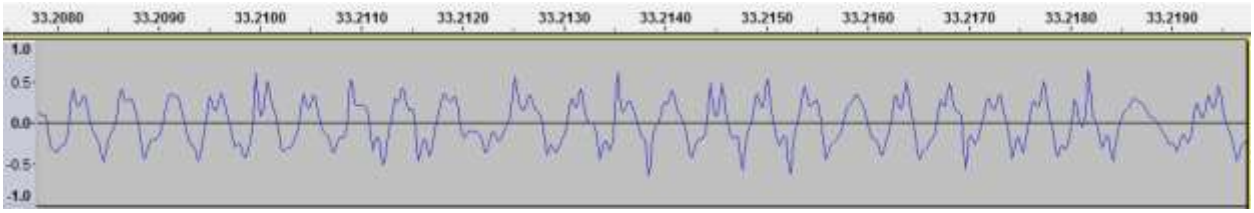
More complicated cases, such as multiple USB audio adapters, are explained later.

3 Setting proper transmit audio level

The optimal peak FM deviation is somewhere around 3 or 3.5 kHz. Obviously if it is too low, you won't be heard. If too loud, the audio signal will be distorted, making it less likely to be decoded properly.

You might expect it to clip and become more like a square wave but that is not what happens. Instead of being a sine wave, the top and bottom start to get jagged like this:

Here is a more extreme case of over-deviation:



Some interesting examples of really terrible signals can be found in ***A-Closer-Look-at-the-WA8LMF-TNC-Test-CD.pdf.*** You don't want to be included in my next collection of really bad signal examples. More importantly you probably want others to understand what you are sending.

Most of us don't have fancy radio test equipment, so how can the FM deviation be set properly?

You should be able to get a reasonably good approximation by ear. Adjust the transmit audio level up and down while listening on a different radio. It will stop getting louder and sound different at the point where it starts to distort. Turn it down a little so you can hear a decrease in volume. If you have an oscilloscope to look at the received audio, that would be more scientific. Notice where it starts to distort and decrease the amplitude to around ¾ of that.

After you have configured Dire Wolf for your soundcard and PTT method, you can get a continuous test tone by using "-x" followed by one of these letters on the command line:

- m for the mark tone, typically 1200 Hz.
- s for the space tone, typically 2200 Hz.
- a to alternate between them.

For a multi-radio setup, a radio channel, other than the first, can be specified by putting a number in there too.

4 Computer Audio

Back in the early days of the IBM style personal computer, sound was not a standard feature. It was later added with “sound cards” that were plugged into the motherboard. Now it is a standard feature integrated into the system board. You can use the existing internal sound system or add a dedicated interface for use by the radio. Most often this will be a USB device. The same “sound card” terminology lives on although it is not a “card” anymore.

Starting in the upper left is an old “sound card” that plugged into the motherboard.

Below that is a USB audio adapter that runs in the 5 to 10 dollar range. It has stereo audio output for headphones and mono audio input for a microphone. I added the wires sticking out. This is for PTT control which we will discuss later.

The SignaLink USB is an overgrown version of the USB audio adapter. It also has audio isolation transformers and a relay to break up the ground between the radio and computer. It also a VOX circuit that turns on the transmitter when transmit audio is present.

Finally, in the lower right we have a Raspberry Pi (RPi) with the UDRC audio board on top. Rather than using USB, it uses the special I^2S bus designed for digital audio. The round 6 pin connector matches the “data” connector found on many Kenwood, Icom, and Yaesu transceivers. A newer version of this is called DRAWS. It has two radio connectors, GPS, real time clock, and a voltage regulator so the RPi can be run from 12 volts.

DRAWS, and the earlier UDRC (pictured here) are high performance soundcards that fit on top of the Raspberry Pi. They can use higher audio sampling rates making them ideal for 9600 baud and above.

The 6 pin mini-DIN radio connector matches the "data" connectors on many transceivers, making hook up very easy. For details, see http://nwdigitalradio.com/draws/ and https://nw-digital-radio.groups.io/g/udrc/wiki/UDRC%E2%84%A2-and-Direwolf-Packet-Modem#Basic-Configuration

A simpler audio board (not shown) with an I^2C / SPI interface so it doesn't tie up a USB port. This would be especially beneficial with the Pi zero which has only a single USB port that you might want to use for something else. https://wb7fhc.com/about-the-fe-pi.html

The Raspberry Pi Foundation even produces one of their own.
https://www.raspberrypi.org/products/iqaudio-codec-zero/

4.1 Special notes for USB Audio Adapters

These all have two 3.5mm TRS (tip, ring, sleeve) connectors marked headphone and mic. The headphone jack is stereo so you will want to use a stereo (3 conductor) cable. If you use a mono (2 conductor) cable, you would be shorting one of the outputs to ground.

The microphone connector is a little less obvious.

Here is a sample diagram for what you might find inside of one of these USB audio adapters: http://www.qsl.net/om3cph/sb/CM108_DataSheet_v1.6.pdf See second to last page.

It shows the microphone jack tip capacitively coupled to the CM108 chip with 1 µF. The ring has a microphone bias voltage.

Here is another that is just the opposite: http://www.hardwaresecrets.com/datasheets/CM109.pdf
The tip is a DC bias voltage and the audio input is on the ring.

This http://www.repeater-builder.com/voip/pdf/cm119-datasheet.pdf has both on the ring and the tip is not connected.

I guess you just need to try both and see what works. If you are not getting any audio input, or it is extremely weak – only from crosstalk – that might be the problem. The printed circuit board, mentioned a couple pages later, has a jumper so you can easily try either one.

Both the Syba product mentioned above, and a different one from Adafruit, have the tip and ring connected together. They have an open circuit voltage of 4.51 volts which drops down to about half that when connected to ground through a 1.5 k resistor.

☞ **The lesson, here, is that you should use a 3.5 mm stereo (TRS) plug, not a mono (TR) plug for the microphone input. If you use a mono cable,**

the longer sleeve on the plug might short the microphone input to ground.

Finally, you might want to stick a 1 µF capacitor between the receive audio and the microphone input, due to the DC bias, but I never found it to be necessary.

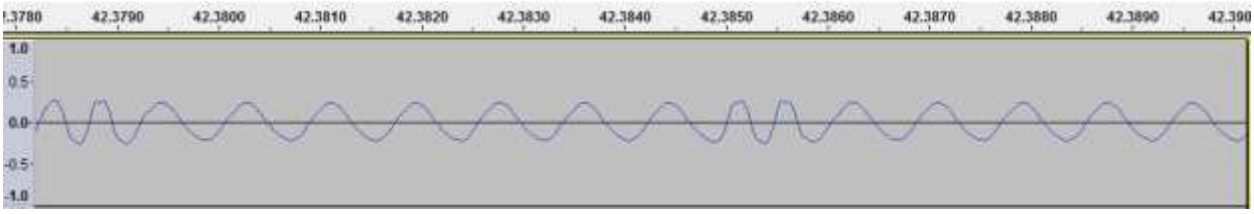
5 Connection to Radio

The major Japanese Ham Radio manufacturers standardized on the same 6 pin mini DIN connector for use with external modems such as a packet radio TNC. It's usually labeled "data" when it is really audio. I would have called it "external modem" but they didn't ask me for my advice.

Alinco has the equivalent but it uses a different connector style.

Use this if it is available. The microphone and speaker connections are optimized for voice operation and intentionally introduce distortion which makes it harder for the modem to operate effectively.

VHF/UHF FM transmitters reshape the audio with "pre-emphasis." This boosts the higher frequencies at +6 dB per octave. Suppose your modem was switching between 1200 and 2200 Hz tones of the same amplitude.



The pre-emphasis increases the amplitude of the higher tone by almost a factor of 2.

In the receiver, de-emphasis decreases the amplitude of higher frequencies, again by 6 dB per octave. In theory they should balance out

Frequencies below 300 Hz or so, are eliminated so you don't hear the CTCSS tones (commonly known as PL). This is all fine for voice but makes the modem's job more challenging. Dire Wolf goes to a lot of effort to compensate for this imbalance for superior decoding performance.

More details in ***A-Better-APRS-Packet-Demodulator-Part-1-1200-baud.pdf***

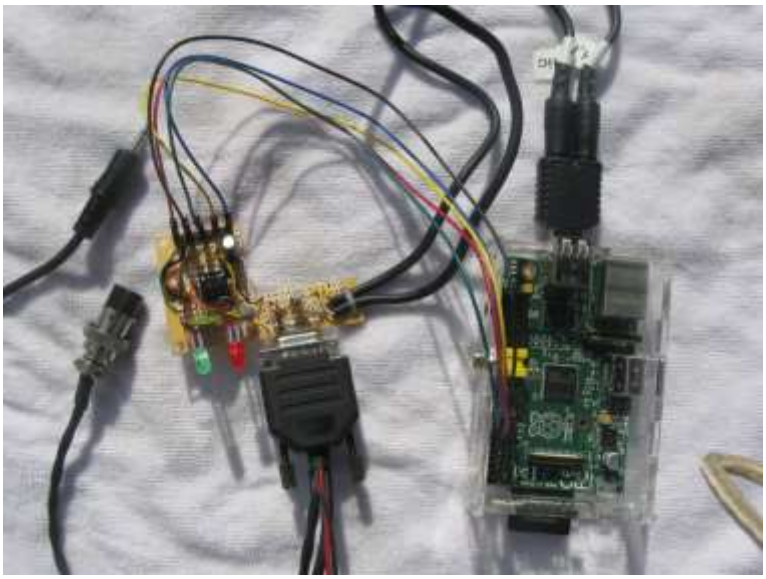
The most common 1200 bps (bits per second) speed is somewhat tolerant of this distortion but for 9600 you absolutely need to use the special connector which bypasses the pre-emphasis and de-emphasis to provide a flat wide audio bandwidth.

Trying to use the microphone and speaker connections, for 9600 bps, will be an exercise in futility.

6 What goes in between?

There is a wide range of options from air to more involved designs with audio isolation transformers and an optoisolator for PTT. Others have covered this topic extensively so I will just provide some examples and links.

This is what I'm using for my digipeater / IGate.



Left: Connections to speaker and Microphone jack of transceiver.

Middle: Interface circuit with a timer to limit transmission time.

It uses the standard 9 pin connector found on most TNCs and trackers so the same radio-specific cables can be used.

The two LEDs are for Data Carrier Detect (DCD) and the Push to Talk (PTT) signals.

The shape of the board was not intentional. It was just a scrap piece of perfboard left over from another project.

Right: Raspberry Pi. A model 1 is more than adequate.

At the top is a USB audio adapter. The current software version can handle 3 audio interfaces at the same time.

Lower left are GPIO connections for PTT and the DCD LED.

I usually take the receive audio and connect it directly to the audio input on the computer.

For transmit, I use a couple resistors, such as 10k and 1k, to

TO BE CONTINUED

7 PTT

Like other many other Ham Radio "soundcard" applications, Dire Wolf provides a wide variety of methods to activate your transmitter. It sounds simple but there are many different options and some require detailed explanation to use them most effectively.

- Serial Port

  Back in the previous Century, personal computers had RS-232 serial "com" ports. Using one of the control lines such as RTS (conveniently called Request to Send) was a popular method. Serial ports are now about as scarce as hen's teeth. You can still use this method by using a USB to serial converter cable. You can also find interfaces which have a USB-to-serial chip used only to generate the PTT signal.

- GPIO or GPIOD

  If you have a single board computer, such as a Raspberry Pi, one of the General Purpose Input Output (GPIO) pins can be used.

- GPIO pins inside a USB Audio adapter

  The C-Media CM108 and CM119 chips are very popular for USB to Audio adapters. They have GPIO pins that we can use for the PTT signal. This is a very tidy solution because everything goes through a single USB cable.

- Hamlib - CAT (Computer Aided Transceiver)

  Most ham transceivers made in the past 30 years or so, have the ability to be controlled from a computer. Originally these used RS-232 serial ports but now USB is the norm. The commands are not standardized so "hamlib" is generally used to deal with all the different possibilities for various brands and models.

- VOX

  The transmitter can be activated automatically when transmit audio is present. Use of the VOX, built in to some transceivers, is generally a bad idea because it is designed for voice and keeps the transmitter on much too long after the transmit audio has stopped.

This ***Dire Wolf Radio Interface Guide***, new for release 1.7, gathers scattered information into one place, goes into more detail than the ***User Guide***, and should make finding answers easier.

7.1 Serial Port (not recommended)

Back in the previous Century, personal computers had RS-232 serial "COM" ports. Using one of the control lines such as RTS (conveniently called Request to Send) was a popular method. Serial ports are now about as scarce as hen's teeth. One possible method is to use a USB to serial adapter.

> **This is NOT a very good approach and not recommended. When a computer is rebooted, the default state for the RTS status line is usually on. So, if your computer is unexpectedly rebooted, the RTS control line will cause your transmitter to stay on, annoying other people and possibly damaging your transmitter.**
>
> **If using this approach, you should ensure that the transmitter does not stay on for too long. Many transceivers have a transmit time out option to limit the length of a transmission. You could also use a hardware timer. An example is shown in the GPIO section.**

If you do anything with RS-232 devices, you should get a cable like this with indicator lights built in. It will save a lot of time and aggravation when troubleshooting. https://www.gearmo.com/shop/usb2-0-rs-232-serial-adapter-led-indicators/

Don't connect the RS-232 signal to your transceiver!!! You will need:

- A resistor of about 10k.
- General purpose diode (1N4148, 1N914, etc.) to prevent the base from going too far negative and damaging the transistor.
- General purpose NPN transistor (2N3904, 2N2222, or whatever you have in your junk box).

Some example circuits can be found here:

- https://www.qsl.net/wm2u/interface.html
- http://zs1i.blogspot.com/2010/02/zs1i-soundcard-interface-ii-project.html
- https://kb3kai.com/tnc/soft-tnc.pdf
- http://www.dunmire.org/projects/DigitalCommCenter/soundmodem/mySoundCardInterface.png

To use a serial port (either built-in or a USB to RS232 adapter cable) for PTT control, use an option of this form in your configuration file, in the appropriate CHANNEL section:

PTT *device-name* [-]*rts-or-dtr* *[[-]rts-or-dtr]*

(The [ ] indicate something is optional.)

For Windows the device name would be COM1, COM2, etc.

For Linux, the device name would probably be something like

- /dev/ttyS0 or /dev/ttyS1 for a COM port on the PC motherboard, or
- /dev/ttyUSB0 or /dev/ttyUSB1 for a USB-to-RS232 adapter. You would also see this for transceivers, like the IC-7100, that have a USB-to-serial converter built in.

You can also use the Windows format on Linux. COM1 is converted to /dev/ttyS0, COM1 is converted to /dev/ttyS1, and so on. Remember this would apply to a COM port on the motherboard or in a PCI slot. If USB is involved, the names would be different.

Examples:

PTT COM1 RTS

PTT COM1 -DTR

PTT /dev/ttyUSB0 RTS

Normally the higher voltage is used for transmit. Prefix the control line name with "-" to get the opposite polarity. Some interfaces want RTS and DTR to be driven with opposite polarity to minimize chances of transmitting at the wrong time. You can specify two control lines with opposite polarity. Example:

```
PTT   COM1   -RTS   DTR
```

```
PTT  COM1  RTS  -DTR
```

For the Easy Digi Complete interface, with built-in USB to serial adapter for PTT control, you should drive both control lines with the same polarity. These are both equivalent:

```
PTT  COM1  RTS  DTR

PTT  COM1  DTR  RTS
```

Alternatively, the RTS and DTR signals from one serial port could control two transmitters. E.g.

```
CHANNEL  0

PTT  COM1  RTS

CHANNEL  1

PTT  COM1  DTR
```

7.2 GPIO

On Linux you can use General Purpose I/O (GPIO) pins if available. This is mostly applicable to a single board, such as a Raspberry Pi, BeagleBone, Orange Pi, etc., not a general purpose PC.

> ☞ CAUTION! The general purpose input output (GPIO) pins are connected directly to the CPU chip. There is no buffering or other protection. The interface uses 3.3 volts and will not tolerate 5 volt signals. Static discharge, from careless handing, could destroy your Raspberry Pi.

There are many GPIO pins. How would you choose an appropriate one? I would try to avoid those with special functions such as UART, SPI, PWM, or I^2C. These are my suggestions for the best choices. Note that the physical pin number on the connector, and the GPIO number (used by the software), are not the same.

For the original RPi model 1 (before the "plus" versions):

- P1-11 GPIO 17
- P1-15 GPIO 22
- P1-16 GPIO 23
- P1-18 GPIO 24
- P1-22 GPIO 25

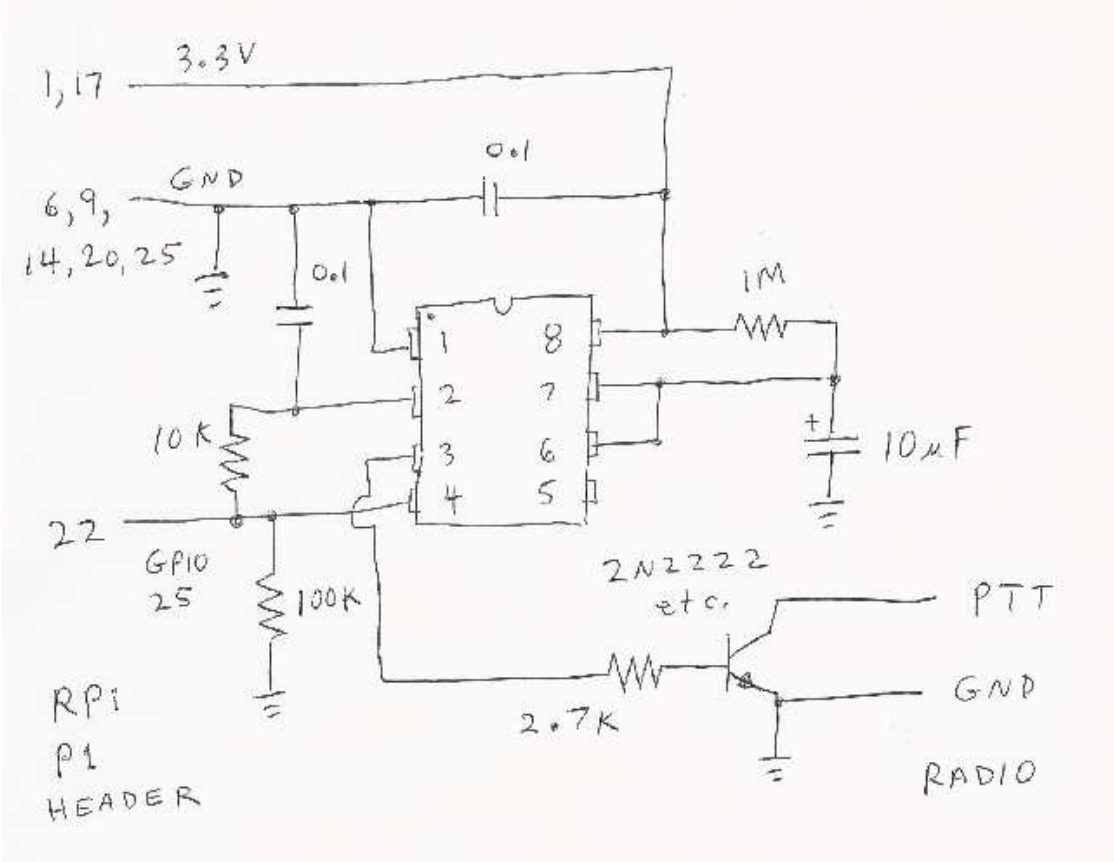
The later models (A+, B+, 2, 3, 4) have a larger connector with additional GPIO pins. Any of the pins listed above or these new ones would be suitable.

- P1-29 GPIO 5
- P1-31 GPIO 6
- P1-32 GPIO 12
- P1-33 GPIO 13
- P1-35 GPIO 19
- P1-36 GPIO 16
- P1-37 GPIO 26
- P1-38 GPIO 20
- P1-40 GPIO 21

You could simply use a resistor and transistor to control PTT on your radio. Most modern rigs have a Transmit Time Out setting to set the maximum transmit time. Use the lowest setting available but at least 15 seconds. If something goes wrong, you won't jam other people and possibly damage your rig.

If that feature is not available, you might want to add a protective circuit such as this one. It uses a **CMOS** 555 timer (LMC555, TLC555, ICM7555, etc.) to limit transmissions to about 10 seconds in case something goes wrong. **Don't try using the original 555** because it needs a minimum of 4.5 volts and we have only 3.3 here. The time can be increased by making the 10 µF capacitor larger. It's roughly 1 second for each µF.

One person reported that the timeout did not work properly and the problem was solved by adding a 0.01 µf capacitor from pin 4 to ground. I suspect this was caused by RF, from the transmitter, getting into the digital circuitry.



7.2.1 Configuration - original (sysfs)

This section describes the old method which is now deprecated. Depending on your operating system, it might still be available. If not, you will need to use the new method in the next section.

Add a PTT GPIO command to each CHANNEL section. Precede the pin number with "-" to invert the signal.

PTT GPIO [-]*pin-number*

Examples:

```
PTT  GPIO  25

PTT  GPIO  -25
```

There are more details in the ***Raspberry-Pi-APRS.pdf*** document.

Behind the scenes, the GPIO pins have device paths such as /sys/class/gpio/gpio25. You could set the pin state with a command such as:

```
echo 1 > /sys/class/gpio/gpio25/value
```

You don't need to know that level of detail; I'm just providing a lead in for the next section.

This functionality has been removed for the bookworm version of Raspberry Pi OS for the RPi 5. We now need to use a new method, which unfortunately is a little more complicated. I would like to hide all the details behind the scenes but haven't figured out a way to do that yet.

If you try using the old method on a Raspberry Pi 5 with bookworm, you should get a message something like this:

```
Error writing "14" to /sys/class/gpio/export, errno=22
Invalid argument
It looks like gpio with sysfs is not supported on this operating system.
Rather than the following form, in the configuration file,
   PTT GPIO  14
try using gpiod form instead.  e.g.
   PTT GPIOD  gpiochip0  14
You can get a list of gpio chip names and corresponding I/O lines with "gpioinfo" command.
```

7.2.2 Configuration - new character device API (libgpiod)

(New in release 1.8 - "dev" branch as I write this.)

Here are some explanations of the difference:

https://embeddedbits.org/linux-kernel-gpio-user-space-interface/

https://www.beyondlogic.org/an-introduction-to-chardev-gpio-and-libgpiod-on-the-raspberry-pi/

(1) If you have never built from source before do this first, otherwise skip to step (2).

```
sudo apt-get install cmake
sudo apt-get install libasound2-dev
sudo apt-get install libudev-dev

git clone https://www.github.com/wb2osz/direwolf
```

(2) You must install these two additional packages for RPi bookworm.

```
sudo  apt  install  gpiod  libgpiod-dev
```

(2) Rebuild from source for release 1.8 (currently in "dev" branch). Be sure to remove any existing build directory. Otherwise cmake might use cached information and not include the new library.

```
cd ~
cd direwolf
git pull
git checkout dev
rm -rf build
mkdir build
cd build
cmake ..
make -j4
sudo make install
```

(3) When direwolf starts up, it should have libgpiod listed among the optional features included:

```
Includes optional support for:  libgpiod
```

If you don't see libgpiod in there, go back and fix it. The following commands won't work without it.

Unfortunately we can no longer simply refer to the GPIO pin with just a number. We need to specify:

- The gpio controller name, named **gpiochip***N* although it's not a separate chip.
- The line number.

Type "gpioinfo" to get a list of the gpiochips and their lines.

On a Raspberry Pi 3 (Debian 11 - bullseye) it looks like this:

```
$ gpioinfo
```

```
gpiochip0 - 54 lines:
        line   0:     "ID_SDA"       unused   input  active-high
        line   1:     "ID_SCL"       unused   input  active-high
        line   2:       "SDA1"       unused   input  active-high
        line   3:       "SCL1"       unused   input  active-high
        line   4:  "GPIO_GCLK"       unused   input  active-high
        line   5:      "GPIO5"       unused   input  active-high
        line   6:      "GPIO6"       unused   input  active-high
        line   7:  "SPI_CE1_N"       unused   input  active-high
        line   8:  "SPI_CE0_N"       unused   input  active-high
        line   9:   "SPI_MISO"       unused   input  active-high
        line  10:   "SPI_MOSI"       unused   input  active-high
        line  11:   "SPI_SCLK"       unused   input  active-high
        line  12:     "GPIO12"       unused   input  active-high
        line  13:     "GPIO13"       unused   input  active-high
        line  14:       "TXD1"       unused   input  active-high
        line  15:       "RXD1"       unused   input  active-high
        line  16:     "GPIO16"       unused   input  active-high
        line  17:     "GPIO17"       unused   input  active-high
        line  18:     "GPIO18"       unused   input  active-high
        line  19:     "GPIO19"       unused   input  active-high
        line  20:     "GPIO20"       unused   input  active-high
        line  21:     "GPIO21"       unused   input  active-high
        line  22:     "GPIO22"       unused   input  active-high
        line  23:     "GPIO23"       unused   input  active-high
        line  24:     "GPIO24"       unused   input  active-high
        line  25:     "GPIO25"       unused   input  active-high
        line  26:     "GPIO26"       unused   input  active-high
        line  27:     "GPIO27"       unused   input  active-high
        line  28:         "NC"       unused   input  active-high
        line  29: "LAN_RUN_BOOT"     unused   input  active-high
        line  30:       "CTS0"       unused   input  active-high
        line  31:       "RTS0"       unused   input  active-high
        line  32:       "TXD0"       unused   input  active-high
        line  33:       "RXD0"       unused   input  active-high
        line  34:    "SD1_CLK"       unused   input  active-high
        line  35:    "SD1_CMD"       unused   input  active-high
        line  36:  "SD1_DATA0"       unused   input  active-high
        line  37:  "SD1_DATA1"       unused   input  active-high
        line  38:  "SD1_DATA2"       unused   input  active-high
        line  39:  "SD1_DATA3"       unused   input  active-high
        line  40:   "PWM0_OUT"       unused   input  active-high
        line  41:   "PWM1_OUT"       unused   input  active-high
        line  42:    "ETH_CLK"       unused   input  active-high
        line  43:   "WIFI_CLK"       unused   input  active-high
        line  44:       "SDA0"       unused   input  active-high
        line  45:       "SCL0"       unused   input  active-high
        line  46:   "SMPS_SCL"       unused   input  active-high
        line  47:   "SMPS_SDA"       unused  output  active-high
        line  48:   "SD_CLK_R"       unused   input  active-high
        line  49:   "SD_CMD_R"       unused   input  active-high
        line  50: "SD_DATA0_R"       unused   input  active-high
        line  51: "SD_DATA1_R"       unused   input  active-high
        line  52: "SD_DATA2_R"       unused   input  active-high
        line  53: "SD_DATA3_R"       unused   input  active-high
gpiochip1 - 2 lines:
        line   0:      unnamed        "ACT"  output  active-high [used]
        line   1:      unnamed       unused   input  active-high
gpiochip2 - 8 lines:
        line   0:      "BT_ON"       unused  output  active-high
        line   1:      "WL_ON"       unused  output  active-high
```

```
        line   2: "STATUS_LED"       unused  output  active-high
        line   3:    "LAN_RUN"       unused  output  active-high
        line   4: "HDMI_HPD_N"        "hpd"   input   active-low [used]
        line   5:  "CAM_GPIO0" "cam1_regulator" output active-high [used]
        line   6:  "CAM_GPIO1"       unused  output  active-high
        line   7:  "PWR_LOW_N"        "PWR"   input  active-high [used]
```

On the RPi 5 (Debian 12 - bookworm), it looks like this:

```
$ gpioinfo
gpiochip0 - 32 lines:
        line   0:          "-"       unused   input  active-high
        line   1: "2712_BOOT_CS_N" "spi10 CS0" output active-low [used]
        line   2: "2712_BOOT_MISO" unused input active-high
        line   3: "2712_BOOT_MOSI" unused input active-high
        line   4: "2712_BOOT_SCLK" unused input active-high
        line   5:          "-"       unused   input  active-high
        line   6:          "-"       unused   input  active-high
        line   7:          "-"       unused   input  active-high
        line   8:          "-"       unused   input  active-high
        line   9:          "-"       unused   input  active-high
        line  10:          "-"       unused   input  active-high
        line  11:          "-"       unused   input  active-high
        line  12:          "-"       unused   input  active-high
        line  13:          "-"       unused   input  active-high
        line  14:   "PCIE_SDA"       unused   input  active-high
        line  15:   "PCIE_SCL"       unused   input  active-high
        line  16:          "-"       unused   input  active-high
        line  17:          "-"       unused   input  active-high
        line  18:          "-"       unused   input  active-high
        line  19:          "-"       unused   input  active-high
        line  20:   "PWR_GPIO" "pwr_button"   input   active-low [used]
        line  21: "2712_G21_FS" unused input active-high
        line  22:          "-"       unused   input  active-high
        line  23:          "-"       unused   input  active-high
        line  24:     "BT_RTS"       unused   input  active-high
        line  25:     "BT_CTS"       unused   input  active-high
        line  26:     "BT_TXD"       unused   input  active-high
        line  27:     "BT_RXD"       unused   input  active-high
        line  28:      "WL_ON"  "wl_on_reg"  output  active-high [used]
        line  29:      "BT_ON"   "shutdown"  output  active-high [used]
        line  30: "WIFI_SDIO_CLK" unused input active-high
        line  31: "WIFI_SDIO_CMD" unused input active-high
gpiochip1 - 4 lines:
        line   0: "WIFI_SDIO_D0" unused input active-high
        line   1: "WIFI_SDIO_D1" unused input active-high
        line   2: "WIFI_SDIO_D2" unused input active-high
        line   3: "WIFI_SDIO_D3" unused input active-high
gpiochip2 - 17 lines:
        line   0:    "RP1_SDA"       unused   input  active-high
        line   1:    "RP1_SCL"       unused   input  active-high
        line   2:    "RP1_RUN" "RP1 RUN pin" output active-high [used]
        line   3: "SD_IOVDD_SEL" "vdd-sd-io" output active-high [used]
        line   4:  "SD_PWR_ON" "sd_vcc_reg"  output  active-high [used]
        line   5:  "SD_CDET_N"       unused   input  active-high
        line   6:   "SD_FLG_N"       unused   input  active-high
        line   7:          "-"       unused   input  active-high
        line   8:  "2712_WAKE"       unused   input  active-high
        line   9: "2712_STAT_LED" "ACT" output active-low [used]
```

```
        line  10:          "-"       unused   input  active-high
        line  11:          "-"       unused   input  active-high
        line  12:   "PMIC_INT"       unused   input  active-high
        line  13: "UART_TX_FS"       unused   input  active-high
        line  14: "UART_RX_FS"       unused   input  active-high
        line  15:          "-"       unused   input  active-high
        line  16:          "-"       unused   input  active-high
gpiochip3 - 6 lines:
        line   0:  "HDMI0_SCL"       unused   input  active-high
        line   1:  "HDMI0_SDA"       unused   input  active-high
        line   2:  "HDMI1_SCL"       unused   input  active-high
        line   3:  "HDMI1_SDA"       unused   input  active-high
        line   4:   "PMIC_SCL"       unused   input  active-high
        line   5:   "PMIC_SDA"       unused   input  active-high
gpiochip4 - 54 lines:
        line   0:      "ID_SD"       unused   input  active-high
        line   1:      "ID_SC"       unused   input  active-high
        line   2:       "PIN3"       unused   input  active-high
        line   3:       "PIN5"       unused   input  active-high
        line   4:       "PIN7"       unused   input  active-high
        line   5:      "PIN29"       unused   input  active-high
        line   6:      "PIN31"       unused   input  active-high
        line   7:      "PIN26"       unused   input  active-high
        line   8:      "PIN24"       unused   input  active-high
        line   9:      "PIN21"       unused   input  active-high
        line  10:      "PIN19"       unused   input  active-high
        line  11:      "PIN23"       unused   input  active-high
        line  12:      "PIN32"       unused   input  active-high
        line  13:      "PIN33"       unused   input  active-high
        line  14:       "PIN8"       unused   input  active-high
        line  15:      "PIN10"       unused   input  active-high
        line  16:      "PIN36"       unused   input  active-high
        line  17:      "PIN11"       unused   input  active-high
        line  18:      "PIN12"       unused   input  active-high
        line  19:      "PIN35"       unused   input  active-high
        line  20:      "PIN38"       unused   input  active-high
        line  21:      "PIN40"       unused   input  active-high
        line  22:      "PIN15"       unused   input  active-high
        line  23:      "PIN16"       unused   input  active-high
        line  24:      "PIN18"       unused   input  active-high
        line  25:      "PIN22"       unused   input  active-high
        line  26:      "PIN37"       unused   input  active-high
        line  27:      "PIN13"       unused   input  active-high
        line  28: "PCIE_RP1_WAKE" unused input active-high
        line  29:   "FAN_TACH"       unused   input  active-high
        line  30:   "HOST_SDA"       unused   input  active-high
        line  31:   "HOST_SCL"       unused   input  active-high
        line  32:  "ETH_RST_N"  "phy-reset"  output   active-low [used]
        line  33:          "-"       unused   input  active-high
        line  34: "CD0_IO0_MICCLK" "cam0_reg" output active-high [used]
        line  35: "CD0_IO0_MICDAT0" unused input active-high
        line  36: "RP1_PCIE_CLKREQ_N" unused input active-high
        line  37:          "-"       unused   input  active-high
        line  38:    "CD0_SDA"       unused   input  active-high
        line  39:    "CD0_SCL"       unused   input  active-high
        line  40:    "CD1_SDA"       unused   input  active-high
        line  41:    "CD1_SCL"       unused   input  active-high
        line  42: "USB_VBUS_EN" unused output active-high
        line  43:   "USB_OC_N"       unused   input  active-high
        line  44: "RP1_STAT_LED" "PWR" output active-low [used]
        line  45:    "FAN_PWM"       unused  output  active-high
```

```
line  46: "CD1_IO0_MICCLK" "cam1_reg" output active-high [used]
line  47:  "2712_WAKE"       unused   input  active-high
line  48: "CD1_IO1_MICDAT1" unused input active-high
line  49: "EN_MAX_USB_CUR" unused output active-high
line  50:          "-"       unused   input  active-high
line  51:          "-"       unused   input  active-high
line  52:          "-"       unused   input  active-high
line  53:          "-"       unused   input  active-high
```

Notice there are some important differences.

- gpiochip4 (four) rather than gpiochip0 (zero).
- The same line is named PIN18 rather than GPIO24.

This means that a configuration for one RPi model might not work on a different model.

(You are probably thinking, why not automatically figure out the gpiochip name from the line name. That would be feasible if the naming was consistent. Instead we see GPIO24 in one case and PIN18 in another. Other similar computers might have an entirely different naming convention.)

Example:

In the RPi 3 case, we need to use gpiochip0 and we see the familiar names such as GPIO25 corresponding to line 25.

In this case, the configuration file PTT command would be:

PTT GPIOD gpiochip0 25

“gpiochip0” is case sensitive; it must be exactly as seen in the list from gpioinfo command.

If you wanted to invert the signal so transmit would be indicated by a low voltage, put a minus sign immediately in front of the line number:

PTT GPIOD gpiochip0 -25

Suggestions on how to make this easier on the user are most welcome. I would like to continue using the old configuration file representation and hide all the ugly details when using the new library. I only use two RPi models as examples, but we must keep this general so it will work with other similar single board computers which might have different naming conventions.

7.3 Hamlib - CAT

Most ham transceivers made in the past 30 years or so, have the ability to be controlled from a computer. Originally these used RS-232 serial ports but now USB is the norm. The commands are not standardized so "hamlib" is generally used so each application developer doesn't have to figure out all the different possibilities for various brands and models.

Hamlib support is an optional feature. When direwolf starts up, it displays the optional features that were included when it was built. e.g.

```
Includes optional support for:  gpsd hamlib cm108-ptt
```

If you don't see "hamlib" in there, you will need to rebuild it from source.

If the optional feature is enabled, you can also use this form of the PTT configuration:

`PTT  RIG` *model port [rate]*

Where,

model identifies the type of radio.

A rig number, not a name, is required here.
For example, if you have a Yaesu FT-847, specify 101.
See https://github.com/Hamlib/Hamlib/wiki/Supported-Radios for details.

Get a list of values by running "`rigctl --list`".

2 is used to communicate with "rigctld."

"AUTO" will try to guess what is connected.

port is name of serial port connected to radio.

In the case where model is 2, this would be a host name/address and optional port number. Default port is 4532

rate is an optional serial port rate for CAT control.

Sometimes you might need to override the hamlib default behavior.

Here is an example of where the serial port data rate had to be set explicitly:
https://groups.io/g/RaspberryPi-4-HamRadio/topic/75478708

Examples:

- Yeasu FT-817 on /dev/ttyUSB0: `PTT RIG 120 /dev/ttyUSB0 9600`
- rigctld on localhost: `PTT RIG 2 localhost:4532`
- Try to guess what is on /dev/ttyS0: `PTT RIG AUTO /dev/ttyS0`

For more details, see http://sourceforge.net/p/hamlib/wiki/Hamlib/

FAQ: http://sourceforge.net/p/hamlib/wiki/FAQ/

This would be a good place to go with questions: http://sourceforge.net/p/hamlib/discussion/

7.3.1 Hamlib PTT Example: Use RTS line of serial port.

A normal person would not want to use this Rube Goldberg approach because this directly supported by direwolf. This is just a training exercise to show how it works.

First let's try it manually. In one terminal window, start up a daemon with the desired configuration.

```
rigctld -m 1 -p /dev/ttyS0 -P RTS -t 4532
```

"`/dev/ttyS0`" is the serial port on the mother board.

"`-m 1`" is for the "dummy" backend, not some particular type of radio.

"`-t 4532`" is not really necessary because that is the default port.

In another window,

```
echo "\set_ptt 1" | nc localhost 4532
```

```
echo "\set_ptt 0" | nc localhost 4532
echo "\set_ptt 1" | nc localhost 4532
echo "\set_ptt 0" | nc localhost 4532
```

You should observe that the RTS control line changed. If you bought the USB to Serial cable that I suggested, the built-in indicator lights reveal what is going on. Otherwise, hook up a voltmeter. Next…

```
rigctl -m 2 -r localhost:4532
T 1
T 0
T 1
T 0
q
```

"`-m 2`" means talk to "rigctld."

"`-r localhost:4532`" indicates where rigctld is running. You can leave off the ":4532" because that is the default port. You might also see examples with 127.0.0.1 which is equivalent but obscure and confusing to those without any networking background. Actually, it seems you can omit the "`-r`" option entirely because localhost is the default for rig "model" 2.

Again, we should observe the RTS line of serial port /dev/ttyS0 changing. To use this for PTT, put this in your Dire Wolf configuration file:

```
PTT  RIG  2  localhost:4532
```

The "2" is very important. It means communicate with the instance of "rigctld" that we already started up. In this case it is running on the same host but it could be running on a different computer.

7.4 VOX (Voice Operated Transmit)

The transmitter can be activated automatically when transmit audio is present. The ***SignaLink USB*** uses this technique. Be sure to turn the Delay setting to the minimum position so the transmitter turns off quickly after transmit audio stops.

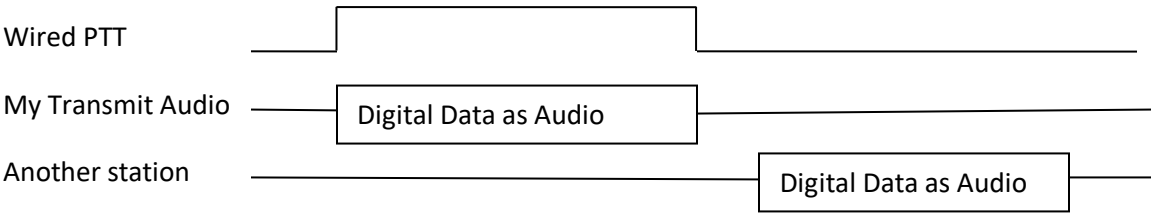
Here are some examples of homebrew circuits:

- http://wa8lmf.net/ham/tonekeyer.htm#NEW
- https://sites.google.com/site/kh6tyinterface/

It might be tempting to use the VOX built into your transceiver and avoid the extra circuitry for the PTT signal. Is this a good idea?

- For APRS, the short answer is NO!
- For connected mode packet, the short answer is ***NO!!!***

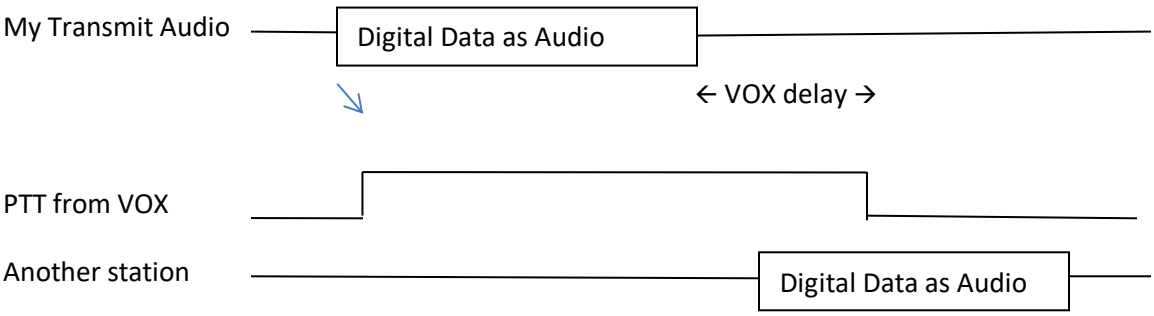
First let's consider the case where we have a wired connection to activate the transmitter. The transmitter is turned on, we send our digital data as an audio signal, and then turn off the transmitter when the audio is finished. Another station detects no other signal, waits a random amount of time (usually less than 1/3 of a second), and starts transmitting.



VOX stands for **Voice** Operated Transmit. It is designed for voice which contains small gaps between words and sentences. It responds quickly when speech begins so it doesn't chop off too much from beginning of the first word. We don't want the transmitter going on and off for every little gap in speech so there is a built in delay before turning the transmitter off again. This is usually referred to as the "VOX delay." On one popular HT, the default is 500 milliseconds and the minimum setting is 250.

Another popular HT doesn't allow the delay time to be configured and the documentation doesn't mention how long it is. It would not be unreasonable to assume they picked something around the default time for another brand.

Keeping the transmitter on a half second after the sound ends is fine for voice. But what about digital data? We saw in the previous section that another station waiting for a clear channel, will usually transmit less than 1/3 second after it no longer hears another digital signal.



Using VOX in this case might be easier but, it is:

- Inconsiderate of the community:

  You are interfering with others by sending a quiet carrier, possibly overpowering other stations trying to be heard.

- Bad for APRS:

  In this case, you will probably miss the beginning of the next station transmitting because you haven't switched back to receive yet.

- ***REALLY BAD*** for connected mode packet:

  Connected mode uses a rapid back-and-forth exchange to acknowledge that information was received and retry if something gets lost. It is quite likely that the next frame is a response to something you sent. If that response frame is lost, your station will keep trying over again and eventually give up.

My recommendation is to avoid using VOX, built into a transceiver, unless you can be sure the transmitter will turn off very soon (e.g. less than 50 mSec.) after the audio signal is no longer present. If you insist on using VOX, be sure the "VOX delay" setting is at the shortest setting.

The SignaLink USB has a built in VOX circuit but it is adjustable into an appropriate range. Turn the delay down to the minimum (fully counterclockwise). According to the documentation, this should turn off the transmitter around 15 or 30 milliseconds after the transmit audio has ended.

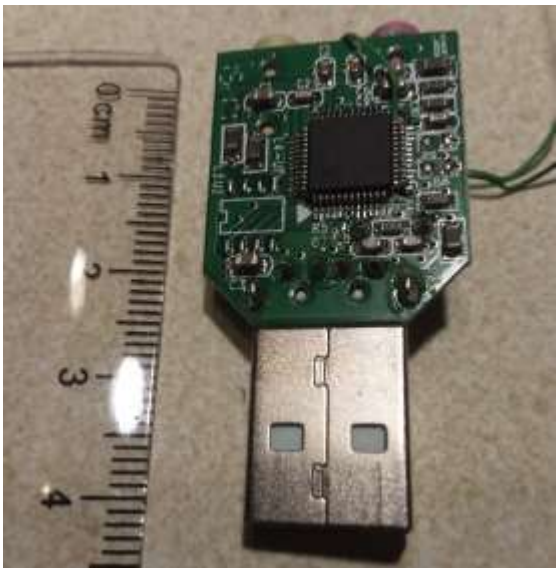
7.5 USB Audio Adapter GPIO - Linux

The C-Media CM108 and CM119 chips are very popular for USB to Audio adapters. They have GPIO pins that we can use for the PTT signal. This is a very tidy solution because everything goes through a single USB cable rather than having separate audio and PTT.

The C-Media CM108, CM119, and similar chips, used in many USB-audio adapters, have varying numbers of general purpose input output (GPIO) pins. These are sometimes connected to push buttons or LEDs.

C-Media chip pin	GPIO number	CM108	CM108AH CM108B	CM109 CM119 CM119A CM119B
43	1	X	X	X
11	2	X		X
13	**3**	**X**	**X**	**X**
15	4	X	X	X
16	5			X
17	6			X
20	7			X
22	8			X

Here, I pried open a SYBA USB Audio adapter, which happen to contain a CM119, and soldered wires to ground and one of the GPIO pins. This signal would then need to go to a resistor and transistor to pull down the PTT control.



Many products use this technique. Some examples:

- **DINAH** https://hamprojects.info/dinah/
- **DMK URI** http://www.dmkeng.com/URI_Order_Page.htm
- **RB-USB RIM** http://www.repeater-builder.com/products/usb-rim-lite.html
- **RA-35** http://www.masterscommunications.com/products/radio-adapter/ra35.html

There are several similar homebrew projects:

- http://www.qsl.net/kb9mwr/projects/voip/usbfob-119.pdf
- http://rtpdir.weebly.com/uploads/1/6/8/7/1687703/usbfob.pdf
- http://www.repeater-builder.com/projects/fob/USB-Fob-Construction.pdf
- http://www.egloff.eu/index.php/en/la-technique/interface-audio-usb

GPIO 3 (pin 13) is used in the homebrew designs because it is easier to tack solder a wire to a pin on the end. All of the commercial products, that I'm aware of, also use the same pin for PTT. It is possible to specify a different pin, in the configuration file, but you probably won't need to.

7.6 USB Audio Adapter GPIO - Linux

The tricky part is that the single physical USB adapter shows up as separate audio and HID (human interaction) devices. Dire Wolf includes a utility application, called **cm108**, to help you make sense of it all.

This is some of what we see for a single USB Audio Adapter:

```
pi@raspberrypi:~ $ cm108
    VID  PID   Product                         Sound                ADEVICE        ADEVICE            HID [ptt]
    ---  ---   -------                         -----                -------        -------            ---------
**  0d8c 0008  C-Media USB Audio Device        /dev/snd/pcmC1D0c    plughw:1,0     plughw:Device,0    /dev/hidraw3
**  0d8c 0008  C-Media USB Audio Device        /dev/snd/pcmC1D0p    plughw:1,0     plughw:Device,0    /dev/hidraw3
**  0d8c 0008  C-Media USB Audio Device        /dev/snd/controlC1                                     /dev/hidraw3
    413c 2010  Dell USB Keyboard                                                                      /dev/hidraw1
    046d c040  USB-PS/2 Optical Mouse                                                                 /dev/hidraw2

** = Can use Audio Adapter GPIO for PTT.
```

VID is Vendor ID. 0d8c is C-Media.
PID is Product ID. This is not a good way to distinguish different chips because it is often programmable.

A single USB audio adapter shows up as multiple sound devices:

- pcmC1D0c Card 1, Device 0, “c” for capture (i.e. input)
- pcmC1D0p Card 1, Device 0, “p” for playback (i.e. output)
- controlC1 Card 1, volume control, etc.

The first ADEVICE column has what you would normally use for ADEVICE in the configuration file. The “1” and “0” correspond to the “card” and “device”. “Device” is an unfortunate term because it has too many different meanings. But that is the terminology used by ALSA.

Ignore the second ADEVICE column for now. We will get back to it later.

Finally HID is the corresponding GPIO device in the same chip. There is not a simple relationship between the names. The same USB audio adapter is Sound Card 1 and HID 3.

In this case, your direwolf configuration file would contain:

```
ADEVICE plughw:1,0
CHANNEL 0
PTT cm108
```

When direwolf is started up, important configuration settings are displayed as a help in troubleshooting.

```
Reading config file direwolf.conf
Audio device for both receive and transmit: plughw:1,0  (channel 0)
Channel 0: 1200 baud, AFSK 1200 & 2200 Hz, A+, 44100 sample rate / 3.
Using /dev/hidraw3 GPIO 3 for channel 0 PTT control.
```

It automatically figures out which HID corresponds to the soundcard. You could optionally override the automatic mapping but you would probably end up "shooting yourself in the foot" as we will see later. All of the products and homebrew projects, that I have seen, all use GPIO 3 so that is the default if you don't specify a different GPIO bit number.

Now let's add two more USB audio adapters.

```
pi@raspberrypi:~ $ cm108
    VID  PID   Product                       Sound                ADEVICE        ADEVICE            HID [ptt]
    ---  ---   -------                       -----                -------        -------            ---------
**  0d8c 0008  C-Media USB Audio Device      /dev/snd/pcmC1D0c    plughw:1,0     plughw:Device,0    /dev/hidraw3
**  0d8c 0008  C-Media USB Audio Device      /dev/snd/pcmC1D0p    plughw:1,0     plughw:Device,0    /dev/hidraw3
**  0d8c 0008  C-Media USB Audio Device      /dev/snd/controlC1                                     /dev/hidraw3
**  0d8c 000c  C-Media USB Headphone Set     /dev/snd/pcmC3D0c    plughw:3,0     plughw:Set,0       /dev/hidraw5
**  0d8c 000c  C-Media USB Headphone Set     /dev/snd/pcmC3D0p    plughw:3,0     plughw:Set,0       /dev/hidraw5
**  0d8c 000c  C-Media USB Headphone Set     /dev/snd/controlC3                                     /dev/hidraw5
    1b3f 2008  USB Audio Device              /dev/snd/pcmC2D0c    plughw:2,0     plughw:Device_1,0  /dev/hidraw4
    1b3f 2008  USB Audio Device              /dev/snd/pcmC2D0p    plughw:2,0     plughw:Device_1,0  /dev/hidraw4
    1b3f 2008  USB Audio Device              /dev/snd/controlC2                                     /dev/hidraw4
    413c 2010  Dell USB Keyboard                                                                    /dev/hidraw1
    046d c040  USB-PS/2 Optical Mouse                                                              /dev/hidraw2

** = Can use Audio Adapter GPIO for PTT.
```

Put this in our configuration file to use 3 radios:

```
ADEVICE plughw:1,0
CHANNEL 0
PTT cm108

ADEVICE1 plughw:3,0
CHANNEL 2
PTT cm108

ADEVICE2 plughw:2,0
CHANNEL 4
PTT cm108
```

Once again, it automatically matches up the soundcard number with the corresponding HID.

```
Reading config file direwolf.conf
Warning: USB audio card 2 (Device_1) is not a device known to work with GPIO PTT.
Audio device for both receive and transmit: plughw:1,0  (channel 0)
Audio device for both receive and transmit: plughw:3,0  (channel 2)
Audio device for both receive and transmit: plughw:2,0  (channel 4)
Channel 0: 1200 baud, AFSK 1200 & 2200 Hz, A+, 44100 sample rate / 3.
Channel 2: 1200 baud, AFSK 1200 & 2200 Hz, A+, 44100 sample rate / 3.
Channel 4: 1200 baud, AFSK 1200 & 2200 Hz, A+, 44100 sample rate / 3.
Using /dev/hidraw3 GPIO 3 for channel 0 PTT control.
Using /dev/hidraw5 GPIO 3 for channel 2 PTT control.
Using /dev/hidraw4 GPIO 3 for channel 4 PTT control.
```

Note that the third USB Audio Adapter does not have a C-Media chip so attempts to use the GPIO pins will probably fail. I can't test it to find out. I pried one of the adapters open and the chip was buried under a blob so there were no pins to probe.

Next, we will unplug this keyboard and mouse then reboot. We don't need them because I'm logging in with ssh over the network.

```
pi@raspberrypi:~ $ cm108
    VID  PID   Product                         Sound                ADEVICE        ADEVICE            HID [ptt]
    ---  ---   -------                         -----                -------        -------            ---------
**  0d8c 0008  C-Media USB Audio Device        /dev/snd/pcmC1D0c    plughw:1,0     plughw:Device,0    /dev/hidraw0
**  0d8c 0008  C-Media USB Audio Device        /dev/snd/pcmC1D0p    plughw:1,0     plughw:Device,0    /dev/hidraw0
**  0d8c 0008  C-Media USB Audio Device        /dev/snd/controlC1                                     /dev/hidraw0
**  0d8c 000c  C-Media USB Headphone Set       /dev/snd/pcmC2D0c    plughw:2,0     plughw:Set,0       /dev/hidraw1
**  0d8c 000c  C-Media USB Headphone Set       /dev/snd/pcmC2D0p    plughw:2,0     plughw:Set,0       /dev/hidraw1
**  0d8c 000c  C-Media USB Headphone Set       /dev/snd/controlC2                                     /dev/hidraw1
    1b3f 2008  USB Audio Device                /dev/snd/pcmC3D0c    plughw:3,0     plughw:Device_1,0  /dev/hidraw2
    1b3f 2008  USB Audio Device                /dev/snd/pcmC3D0p    plughw:3,0     plughw:Device_1,0  /dev/hidraw2
    1b3f 2008  USB Audio Device                /dev/snd/controlC3                                     /dev/hidraw2

** = Can use Audio Adapter GPIO for PTT.
```

What happened?

The soundcards are listed in the same order, but the numbers have changed.

- The soundcard numbers are now 1, 2, 3 rather than 1, 3, 2.
- The HID numbers are now 0, 1, 2 rather than 3, 5, 4.

That's why you should not specify explicit HID numbers. They change when you add or remove USB devices and reboot. Let direwolf figure out the relationship.

The soundcard numbers have also changed. Suppose you had

- plughw:3,0 connected to the radio for the APRS frequency.
- plughw:2,0 connected to the radio for the Packet BBS frequency.

After rebooting they have been reversed!

Don't despair because there is an easy solution.

I've only been showing you part of the cm108 utility results. Here is the rest of it

```
Notice that each USB Audio adapter is assigned a number and a name.  These are not predictable so you could
end up using the wrong adapter after adding or removing other USB devices or after rebooting.  You can assign a
name to each USB adapter so you can refer to the same one each time.  This can be based on any characteristics
that makes them unique such as product id or serial number.  Unfortunately these devices don't have unique serial
numbers so how can we tell them apart?  A name can also be assigned based on the physical USB socket.
Create a file like "/etc/udev/rules.d/85-my-usb-audio.rules" with the following contents and then reboot.

SUBSYSTEM!="sound", GOTO="my_usb_audio_end"
ACTION!="add", GOTO="my_usb_audio_end"
DEVPATH=="/devices/platform/soc/3f980000.usb/usb1/1-1/1-1.3/1-1.3:1.0/sound/card?", ATTR{id}="Fred"
DEVPATH=="/devices/platform/soc/3f980000.usb/usb1/1-1/1-1.4/1-1.4:1.0/sound/card?", ATTR{id}="Wilma"
DEVPATH=="/devices/platform/soc/3f980000.usb/usb1/1-1/1-1.5/1-1.5:1.0/sound/card?", ATTR{id}="Pebbles"
LABEL="my_usb_audio_end"
```

.

Follow the instructions. Copy/paste the last section into /etc/udev/rules.d/85-my-usb-audio.rules then reboot

We can now turn our attention to the second ADEVICE column.

```
pi@raspberrypi:~ $ cm108
    VID  PID   Product                         Sound                ADEVICE        ADEVICE            HID [ptt]
    ---  ---   -------                         -----                -------        -------            ---------
**  0d8c 0008  C-Media USB Audio Device        /dev/snd/pcmC1D0c    plughw:1,0     plughw:Fred,0      /dev/hidraw0
**  0d8c 0008  C-Media USB Audio Device        /dev/snd/pcmC1D0p    plughw:1,0     plughw:Fred,0      /dev/hidraw0
**  0d8c 0008  C-Media USB Audio Device        /dev/snd/controlC1                                     /dev/hidraw0
**  0d8c 000c  C-Media USB Headphone Set       /dev/snd/pcmC2D0c    plughw:2,0     plughw:Wilma,0     /dev/hidraw1
**  0d8c 000c  C-Media USB Headphone Set       /dev/snd/pcmC2D0p    plughw:2,0     plughw:Wilma,0     /dev/hidraw1
**  0d8c 000c  C-Media USB Headphone Set       /dev/snd/controlC2                                     /dev/hidraw1
    1b3f 2008  USB Audio Device                /dev/snd/pcmC3D0c    plughw:3,0     plughw:Pebbles,0   /dev/hidraw2
    1b3f 2008  USB Audio Device                /dev/snd/pcmC3D0p    plughw:3,0     plughw:Pebbles,0   /dev/hidraw2
    1b3f 2008  USB Audio Device                /dev/snd/controlC3                                     /dev/hidraw2

** = Can use Audio Adapter GPIO for PTT.
```

I can now put this in my configuration file, using a name rather than a number which can change.

```
ADEVICE plughw:Fred,0
```

The same technique could be used to assign names based on the VID/PID but that would not help with multiple identical adapters.

Names can be assigned based on serial numbers but these adapters don't have serial numbers.

So, we are left with assigning names based on the physical USB socket on the computer.

The keyboard will be plugged in again, and the second USB Audio Adapter (PID 000c) removed. After rebooting again, we see:

```
pi@raspberrypi:~ $ cm108
    VID  PID   Product                         Sound                ADEVICE        ADEVICE            HID [ptt]
    ---  ---   -------                         -----                -------        -------            ---------
**  0d8c 0008  C-Media USB Audio Device        /dev/snd/pcmC1D0c    plughw:1,0     plughw:Fred,0      /dev/hidraw0
**  0d8c 0008  C-Media USB Audio Device        /dev/snd/pcmC1D0p    plughw:1,0     plughw:Fred,0      /dev/hidraw0
**  0d8c 0008  C-Media USB Audio Device        /dev/snd/controlC1                                     /dev/hidraw0
    1b3f 2008  USB Audio Device                /dev/snd/pcmC2D0c    plughw:2,0     plughw:Pebbles,0   /dev/hidraw3
    1b3f 2008  USB Audio Device                /dev/snd/pcmC2D0p    plughw:2,0     plughw:Pebbles,0   /dev/hidraw3
    1b3f 2008  USB Audio Device                /dev/snd/controlC2                                     /dev/hidraw3
    413c 2010  Dell USB Keyboard                                                                      /dev/hidraw2
    046d c040  USB-PS/2 Optical Mouse                                                                 /dev/hidraw4

** = Can use Audio Adapter GPIO for PTT.
```

Once again, the numbers hav shifted around but the name remains associated with the same USB Audio Adapter. Or more accurately, the USB socket that it is plugged into.

So for our final experiment, swap the two USB Audio Adapters. We don't even need to reboot. We see that the name is associated with the USB socket on the Raspberry Pi, not what is plugged into it.

```
pi@raspberrypi:~ $ cm108
    VID  PID   Product                         Sound                ADEVICE        ADEVICE            HID [ptt]
    ---  ---   -------                         -----                -------        -------            ---------
    1b3f 2008  USB Audio Device                /dev/snd/pcmC1D0c    plughw:1,0     plughw:Fred,0      /dev/hidraw0
    1b3f 2008  USB Audio Device                /dev/snd/pcmC1D0p    plughw:1,0     plughw:Fred,0      /dev/hidraw0
    1b3f 2008  USB Audio Device                /dev/snd/controlC1                                     /dev/hidraw0
**  0d8c 0008  C-Media USB Audio Device        /dev/snd/pcmC2D0c    plughw:2,0     plughw:Pebbles,0   /dev/hidraw3
**  0d8c 0008  C-Media USB Audio Device        /dev/snd/pcmC2D0p    plughw:2,0     plughw:Pebbles,0   /dev/hidraw3
**  0d8c 0008  C-Media USB Audio Device        /dev/snd/controlC2                                     /dev/hidraw3
    413c 2010  Dell USB Keyboard                                                                      /dev/hidraw2
    046d c040  USB-PS/2 Optical Mouse                                                                 /dev/hidraw4

** = Can use Audio Adapter GPIO for PTT.
```

For more information on assigning persistent names by manufacturer, model, or serial number, see https://github.com/dh1tw/remoteAudio/wiki/Persistent-USB-Mapping-of-Audio-devices-(Linux)

7.7 USB Audio Adapter GPIO - Windows

You might find it helpful to read the preceding Linux section first, to compare how these are different.

All of this magic is performed by the “udev” device manager on Linux. The Windows equivalent is drastically different so an entirely new implementation needs to be created.

At the current time, I know how to get a list of the HIDs and set the GPIO pins but so far have not been able to figure out how to find the relationship between HID and sound devices in the same package.

I would really like to provide the same completeness and convenience as on Linux, but so far I have not been able to figure out how to do it. (Hint, hint, to any Windows programming experts.)

Reluctantly I’m providing a half-baked implementation. It should be fine for those using a single USB Audio Adapter but would be real annoying when trying to use multiple adapters.

First, let’s consider the case of a single USB Audio Adapter.

Your configuration file could contain this:

```
ADEVICE USB
CHANNEL 0
PTT cm108
```

There is only a single USB audio device so it all works well.

Things get more complicated when there are multiple USB Audio Adapters.

```
$ cm108
    VID  PID   Product                      HID [ptt]
    ---  ---   -------                      ---------
    413c 2010  Dell USB Keyboard            \\?\hid#vid_413c&pid_2010&mi_00#9&2c9e8850&0&0000#{4d1e55b2-f16f-11cf-88cb-001111000030}\kbd
    413c 2010  Dell USB Keyboard            \\?\hid#vid_413c&pid_2010&mi_01&col01#9&14dafc0b&0&0000#{4d1e55b2-f16f-11cf-88cb-001111000030}
    413c 2010  Dell USB Keyboard            \\?\hid#vid_413c&pid_2010&mi_01&col02#9&14dafc0b&0&0001#{4d1e55b2-f16f-11cf-88cb-001111000030}
    413c 2010  Dell USB Keyboard            \\?\hid#vid_413c&pid_2010&mi_01&col03#9&14dafc0b&0&0002#{4d1e55b2-f16f-11cf-88cb-001111000030}
    0461 4d15  USB Optical Mouse            \\?\hid#vid_0461&pid_4d15#8&1d6abc6&0&0000#{4d1e55b2-f16f-11cf-88cb-001111000030}
**  0d8c 000c  C-Media USB Headphone Set    \\?\hid#vid_0d8c&pid_000c&mi_03#8&e70137e&0&0000#{4d1e55b2-f16f-11cf-88cb-001111000030}
**  0d8c 000c  C-Media USB Headphone Set    \\?\hid#vid_0d8c&pid_000c&mi_03#8&2edc3781&0&0000#{4d1e55b2-f16f-11cf-88cb-001111000030}
    1b3f 2008  USB Audio Device             \\?\hid#vid_1b3f&pid_2008&mi_03#8&21578911&0&0000#{4d1e55b2-f16f-11cf-88cb-001111000030}
**  0d8c 0008  C-Media USB Audio Device     \\?\hid#vid_0d8c&pid_0008&mi_03#8&39d3555&0&0000#{4d1e55b2-f16f-11cf-88cb-001111000030}

** = Can use Audio Adapter GPIO for PTT.
```

The good news is that the paths are still there after a reboot.

```
$ cm108
    VID  PID   Product                      HID [ptt]
    ---  ---   -------                      ---------
**  0d8c 000c  C-Media USB Headphone Set    \\?\hid#vid_0d8c&pid_000c&mi_03#8&2edc3781&0&0000#{4d1e55b2-f16f-11cf-88cb-001111000030}
    413c 2010  Dell USB Keyboard            \\?\hid#vid_413c&pid_2010&mi_00#9&2c9e8850&0&0000#{4d1e55b2-f16f-11cf-88cb-001111000030}\kbd
**  0d8c 0008  C-Media USB Audio Device     \\?\hid#vid_0d8c&pid_0008&mi_03#8&39d3555&0&0000#{4d1e55b2-f16f-11cf-88cb-001111000030}
    413c 2010  Dell USB Keyboard            \\?\hid#vid_413c&pid_2010&mi_01&col01#9&14dafc0b&0&0000#{4d1e55b2-f16f-11cf-88cb-001111000030}
    1b3f 2008  USB Audio Device             \\?\hid#vid_1b3f&pid_2008&mi_03#8&21578911&0&0000#{4d1e55b2-f16f-11cf-88cb-001111000030}
    413c 2010  Dell USB Keyboard            \\?\hid#vid_413c&pid_2010&mi_01&col02#9&14dafc0b&0&0001#{4d1e55b2-f16f-11cf-88cb-001111000030}
    413c 2010  Dell USB Keyboard            \\?\hid#vid_413c&pid_2010&mi_01&col03#9&14dafc0b&0&0002#{4d1e55b2-f16f-11cf-88cb-001111000030}
    0461 4d15  USB Optical Mouse            \\?\hid#vid_0461&pid_4d15#8&1d6abc6&0&0000#{4d1e55b2-f16f-11cf-88cb-001111000030}
**  0d8c 000c  C-Media USB Headphone Set    \\?\hid#vid_0d8c&pid_000c&mi_03#8&e70137e&0&0000#{4d1e55b2-f16f-11cf-88cb-001111000030}

** = Can use Audio Adapter GPIO for PTT.
```

Verify that it is working properly by running cm108 again with the USB Audio HID path on the command line. E.g.

```
>cm108 "\\?\hid#vid_0d8c&pid_0012&mi_03#8&2a5a85b6&0&0000#{4d1e55b2-f16f-11cf-88cb-001111000030}"

01010101010101010101010101010
```

IMPORTANT: The device path has contains characters with special meaning to the command interpreter so it needs to be enclosed in quotation mark (") characters.

The alternating 0 and 1 are displayed as the PTT control is turned off and on. If you are using an interface with a PTT LED indicator, it should be alternating between off and on.

If you have a single USB audio adapter, you should be able to simply use this short form in the direwolf configuration file:

```
PTT cm108
```

However, if you are using multiple devices, you will need to explicitly specify the device path like this:

```
PTT cm108  "\\?\hid#vid_0d8c&pid_000c&mi_03#8&2edc3781&0&0000#{4d1e55b2-f16f-11cf-88cb-001111000030}"
```

8 Software Defined Radio

When using software defined radios (SDR), the audio will be coming from another application rather than a "soundcard."

See ***Radio-Interface-Guide ,pdf*** for information about:

8.1 gqrx

Gqrx (2.3 and later) has the ability to send streaming audio through a UDP socket to another application for further processing. As explained in http://gqrx.dk/doc/streaming-audio-over-udp, select the Network tab of the audio settings window. Enter the host name or address where Dire Wolf will be running. Use "localhost" if both are on the same computer. Pick some unused UDP port which will also be used in the direwolf configuration. Here we use the same UDP port number as in the gqrx documentation example.

Use the following Dire Wolf configuration file options:

```
ADEVICE  udp:7355  default

ARATE  48000

ACHANNELS 1
```

This means that Dire Wolf will obtain audio from a UDP stream for receiving. If you transmit on that channel, audio will go to the ALSA device named "default."

Alternatively, you can override the configuration file settings with command line options like this:

```
direwolf -n 1 -r 48000 -b 16 udp:7355
```

- "-n 1" sets number of audio channels to 1.
- "-r 48000" means audio sample rate of 48000 per second.
- "-b 16" means 16 bits per sample, signed, little endian.

Note that these command line options apply only to the first audio device (ADEVICE0) and the first channel (CHANNEL 0).

8.2 rtl_fm

Other SDR applications might produce audio on stdout so it is convenient to pipe into the next application. In this example, the final "-" means read from stdin.

```
rtl_fm -f 144.39M -o 4 - | direwolf -n 1 -r 24000 -b 16 -
```

Instead of command line options, you could do the same thing in the configuration file like this:

```
ADEVICE0  stdin  default
ARATE  24000
```

What do the rtl_fm options mean?

-f 144.39M	Pretty obvious. You can guess. Frequency accuracy is notoriously bad for the cheapest models. The –p option can be used for calibration. Models
-o 4	By trial and error this seemed to work better.
-	Send audio to stdout where we pipe it to another application. This is the default so it is probably redundant.

Note that the default is 24000 samples per second out. This is adequate for 1200 baud but you would want it to be higher for 9600 baud.

You might be able to get better results by playing with the sampling and filtering options.

See http://kmkeen.com/rtl-demod-guide/index.html for rtl_fm documentation.

What do the direwolf options mean?

-n 1	Single audio channel (mono). This the default but will override Any stereo option in the configuration file.
-r 24000	Audio sample rate. Must match the source.

-b 16	Bits per audio sample. This is the default so it is really redundant.
-	Take audio from stdin. In this case it is piped from rtl_fm.

Here is another possible variation you might want to try. In one window, start up Dire Wolf listening to a UDP port. Note that rtl_fm has a default sample rate of 24000.

```
direwolf -n 1 -r 24000 -b 16 udp:7355
```

In a different window, run rtl_fm and use the netcat utility to send the audio by UDP.

```
rtl_fm -f 144.39M -o 4 - | nc -u localhost 7355
```

Note that the SDR and Dire Wolf can be running on different computers, even different operating systems. You could use the command above on Linux but change localhost to the address of a Windows machine where Dire Wolf is running.

If you see some warning about audio input level being too high, don't worry about in this case.

It's only a potential problem when using the analog input of a sound card. If the analog audio input is too large, it can exceed the range of the A/D converter, resulting in clipping, distortion of the signal, and less reliable demodulation. The warning level is overly cautious. The input level can go much higher before it reaches the A/D range limit.

In this case, where 16 bit digital audio is going from one application to another, there is no danger of overflowing the signal range.

I found this to work well for 9600 baud.

```
rtl_fm -p 62 -f 144.99M -o 4 -s 48000 | direwolf -c sdr.conf -r 48000 -B 9600 -
```

The default 24000 sample rate is too low for reliable 9600 baud operation so we want to increase it to 48000 or maybe even a little higher. Obviously, both applications need to use the same audio sample rate. Results seemed to be better with the "-o 4" option but it wasn't a very scientific test.

I would like to hear from anyone who has done experimentation with different sampling options and came up with something that works better.

The companion document, ***Raspberry-Pi-SDR-IGate.pdf*** , goes into more detail about the frequency error for the cheaper devices and how to deal with it. Most of the information applies to other Linux systems, not just the Raspberry Pi.

8.3 SDR#

The SDR# website doesn't seem to have any documentation on how to use the software. They just point to a Google search:

http://www.google.com/search?q=sdrsharp+tutorial

Here are some other good locations to help you get started.

http://www.atouk.com/SDRSharpQuickStart.html

http://www.qsl.net/yo4tnv/docs/SDRSharp.pdf

https://learn.adafruit.com/getting-started-with-rtl-sdr-and-sdr-sharp/sdr-number-fm-radio

Essential settings:

Frequency: Set to local APRS frequency. Clicking on the upper half of a digit increases it. On the lower half decreases it.

Source: RTL-SDR (or other for your hardware)

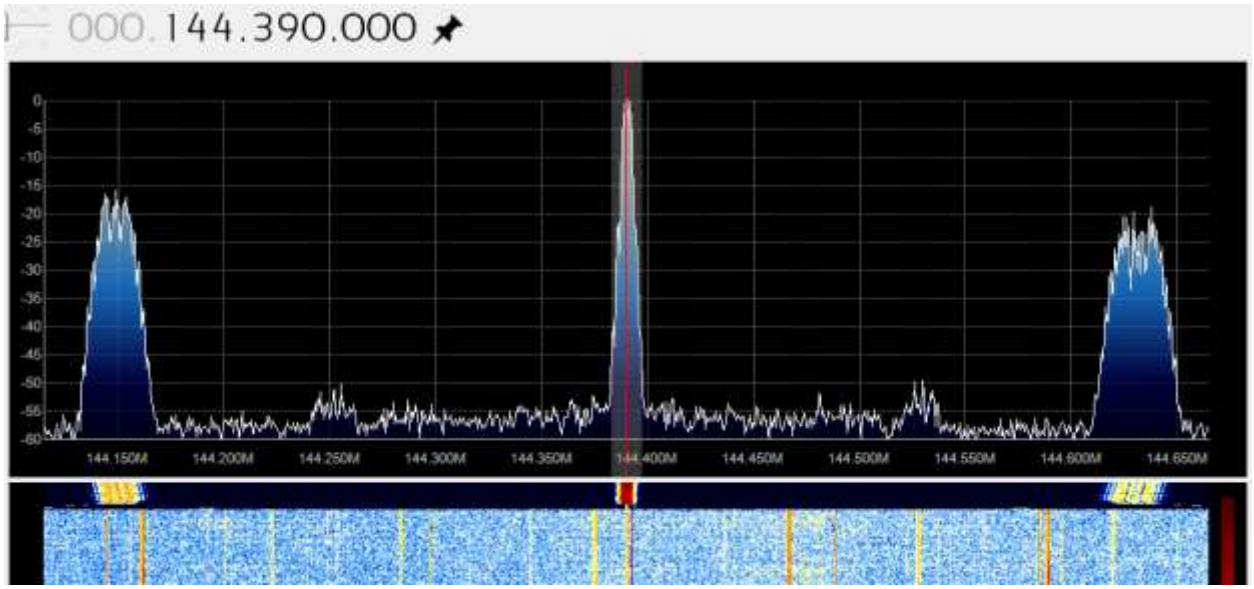
Radio: NFM. This defaults to a bandwidth of 8 kHz which would be appropriate for commercial / public service analog voice with 2.5 kHz deviation. It did work with a very strong signal, but this is probably too narrow for best results. We are dealing with a peak deviation up to ± 5 kHz and the cheap devices are not real accurate about the frequency. You might want to start off with 15,000 here but I'm no expert. You definitely don't want WFM, which is for broadcast FM. This has a 180 kHz bandwidth for 200 kHz channel spacing.

Audio: Note that the sample rate is 48,000 per second. It is grayed out and we can't change it. This will be important later. Set output to speakers for now. We will change this later.

Start button: In the upper left is a triangle pointing to the right. Click to start.

First verify that you can hear the desired signal through the speaker. Check the frequency calibration against a signal of known frequency. My cheap RTL-SDR dongle was off by 64 ppm which is more than 9 kHz on the 2 meter band.

When you receive an APRS signal, it should look something like this:



The side lobes seem to be an artifact from the SDR receiver, not a dirty transmitter. The same thing is seen with a much different transmitter.

How can we send the received audio to another application instead of the speaker? You could install a second sound card and connect the ”line out” from the first to the “line in” of the second. There is another way. Install a “virtual audio cable” which is a pair of imaginary audio devices connected to each other without any hardware in the middle.

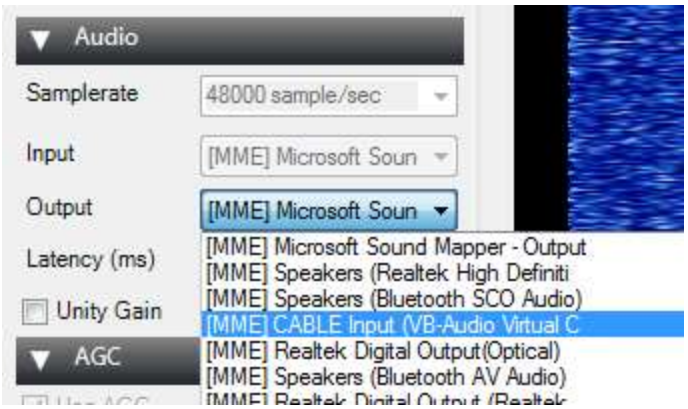
Install VB-CABLE Driver from https://www.vb-audio.com/Cable/index.htm and reboot.

Nothing shows up under the Programs menu so don’t worry when you don’t see anything new there.

If SDR# is currently receiving, you will need to click the pause button

before changing configuration.

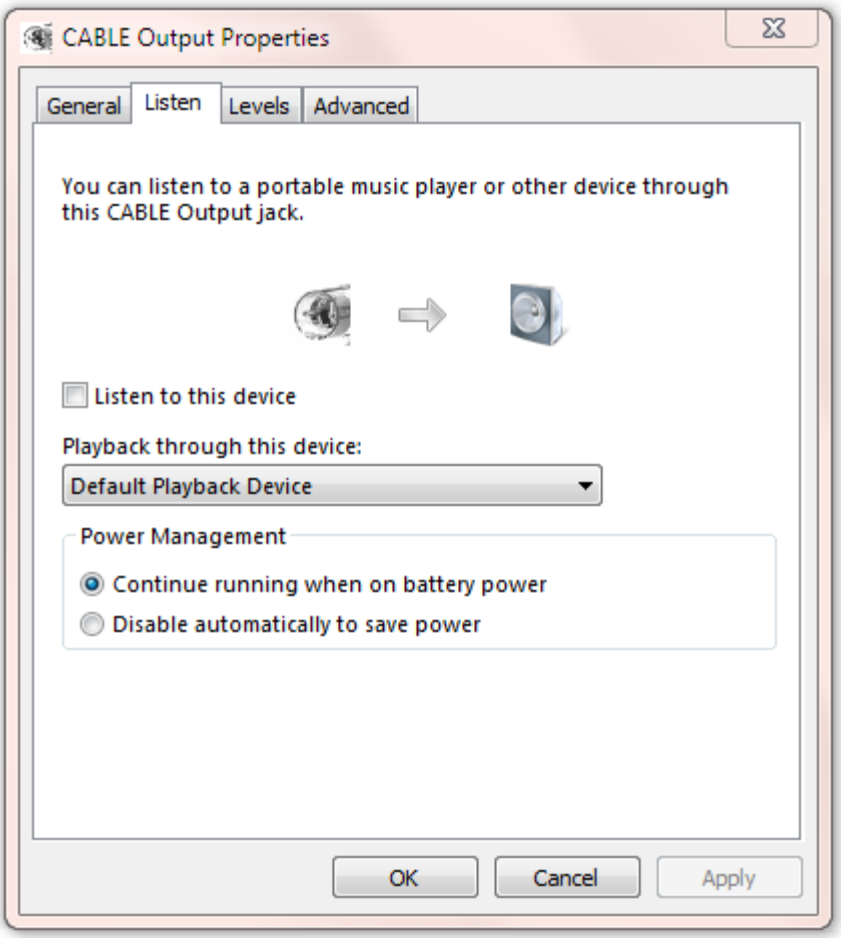
Instead of using the default output, select the new “CABLE Input (VB-Audio Virtual C” instead.



Let's test what we have so far. In the Task Bar notification area (usually bottom right corner) right click on the speaker icon and pick Recording Devices from the pop up menu.

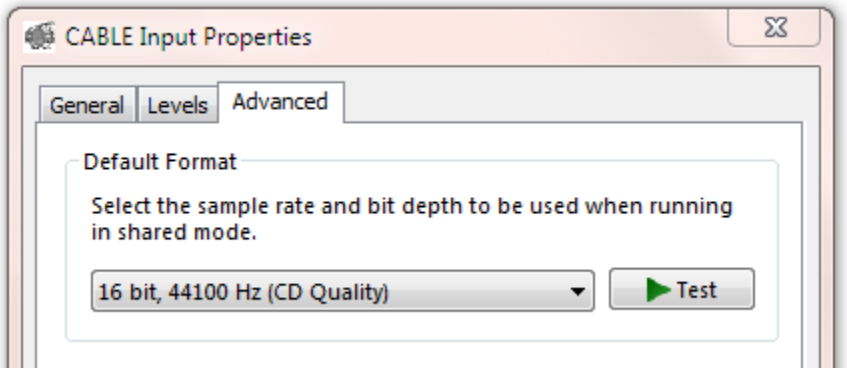
You should see CABLE Output and the level indicator on the right should show some activity. Select it and click the Properties button.

Pick the Listen tab.



Check the "Listen to this device" box and Apply. You should hear audio from SDR# through the speaker. Leave it on for now during the testing. You might want to turn it off again after it is all working.

If you look at the CABLE device Advanced properties,



you will probably find that it says 44100 Hz sample rate. We are using 48000 but this doesn't seem to cause a problem. I don't know if it is performing rate conversions or just pushing the bytes through and not caring.

I found the mismatch to be disturbing and changed it to different values for sample rate, bits per sample, and number of channels. It didn't seem to make any difference. Based on the evidence, this setting seems to be ignored and the bytes just get pushed through. It wouldn't hurt to set it to this just so there is no question:

1 channel, 16 bit, 48000 Hz (DVD Quality)

It is important that the applications producing and consuming the audio stream agree. The delivery service doesn't seem to care.

Put this near the top of your direwolf.conf file and remove any others that would conflict with them.

```
ADEVICE CABLE 0

ARATE 48000
```

When you start up Dire Wolf, you should see something like this:

```
Reading config file direwolf.conf
Available audio input devices for receive (*=selected):
    0: Microphone (Realtek High Defini
    1: Microphone (Bluetooth SCO Audio
    2: Microphone (Bluetooth AV Audio)
 *  3: CABLE Output (VB-Audio Virtual    (channel 0)
Available audio output devices for transmit (*=selected):
 *  0: Speakers (Realtek High Definiti   (channel 0)
    1: Speakers (Bluetooth SCO Audio)
    2: CABLE Input (VB-Audio Virtual C
    3: Realtek Digital Output(Optical)
    4: Speakers (Bluetooth AV Audio)
    5: Realtek Digital Output (Realtek
Channel 0: 1200 baud, AFSK 1200 & 2200 Hz, E+, 48000 sample rate.
Note: PTT not configured for channel 0. (Ignore this if using VOX.)
```

The lines of interest have been highlighted in red.

- The audio input is from the CABLE output device.
- The sample rate matches the value seen in SDR#. ← very important.

You should now be able to decode the packets you hear.

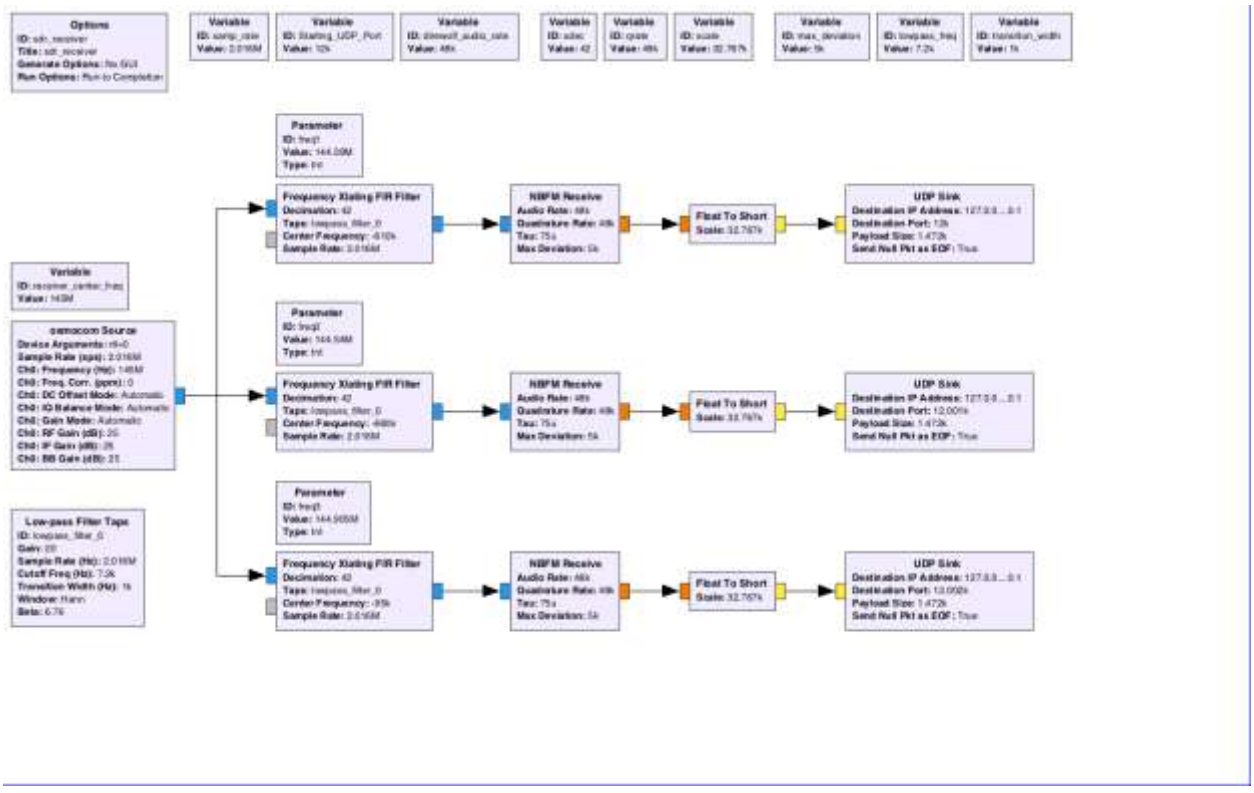
8.4 Gnu Radio – multiple simultaneous channels

Here is a great tip from the forum, posted by N6BA. Thanks, Jeff!!!

https://groups.io/g/direwolf/message/3513

This is a GnuRadio flow graph that listens on 3 frequencies from a single USB attached SDR dongle with the output audio sent over UDP. GRC file attached, which you'll open in gnuradio-companion. You can then subsequently have it generate the python script (sdr_receiver.py) which can then be run from the command line. You'll need to setup direwolf.conf as below to listen on the three UDP ports.

For this example, the center frequency is 145MHz with the sample rate (for the dongle) set to ~2MHz. This will let one listen/tune to frequencies anywhere between 144MHz and 146MHz.



What I've routinely done is to use GnuRadio (ultimately a python script) for listening to multiple frequencies with a single SDR dongle (assuming they're all within ~2.4MHz of each

other). Then have GnuRadio output that audio over a different UDP port for each frequency. That enables one to get away with just a single SDR stick, antenna, and Dire Wolf instance. Then have a Dire Wolf configured something like this:

```
================ snip ================

..
..
..
# this might be for frequency 144.390
ADEVICE0 udp:12000 null
ARATE 48000
ACHANNELS 1
CHANNEL 0
MYCALL mycall
MODEM 1200
FIX_BITS 0


# this might be for frequency 144.340
ADEVICE1 udp:12001 null
ARATE 48000
ACHANNELS 1
CHANNEL 2
MYCALL mycall
MODEM 1200
FIX_BITS 0

# this might be for frequency 145.825
ADEVICE2 udp:12002 null
ARATE 48000
ACHANNELS 1
CHANNEL 4
MYCALL mycall
MODEM 1200
FIX_BITS 0
..
..
..
======================
```

It could make sense to run multiple SDR dongles and antennas if you need to listen on frequencies on different bands (VHF vs UHF, etc.) or are too far apart, MHz-wise, to allow the cheaper dongles to handle. A single GnuRadio script could still be used with multiple SDR dongles as sources, each set to listen on whatever frequencies you're interested in. However, you'll likely get to a point that you'll be asking Dire Wolf to handle more audio channels (aka UDP ports) than it can accommodate without recompiling. In that case you'll need to modify the direwolf.h file and up the number for MAX_ADEVS and recompile:

Default is 3 within direwolf.h:

```
#define MAX_ADEVS 3
```

Cheers,

-Jeff

N6BA

8.5 SDR Troubleshooting

If you can hear packets but they are not being decoded, try adding "-a 10" to the command line. This will print out the audio sample rate and level each 10 seconds. In this example, we see that audio is being received. However, I "accidentally" forgot to set the audio sample rate in the Dire Wolf configuration file so it defaults to 44100. The decoder is expecting 44.1k samples per second, but the actual rate is 48k so the decoding will fail.

```
Reading config file sdrsharp.conf
Available audio input devices for receive (*=selected):
    0: Microphone (Realtek High Defini
    1: Microphone (Bluetooth SCO Audio
    2: Microphone (Bluetooth AV Audio)
 *  3: CABLE Output (VB-Audio Virtual    (channel 0)
Available audio output devices for transmit (*=selected):
 *  0: Speakers (Realtek High Definiti   (channel 0)
    1: Speakers (Bluetooth SCO Audio)
    2: CABLE Input (VB-Audio Virtual C
    3: Realtek Digital Output(Optical)
    4: Speakers (Bluetooth AV Audio)
    5: Realtek Digital Output (Realtek
Channel 0: 1200 baud, AFSK 1200 & 2200 Hz, E+, 44100 sample rate.
Note: PTT not configured for channel 0. (Ignore this if using VOX.)
Ready to accept AGW client application 0 on port 8000 ...
Ready to accept KISS client application on port 8001 ...

ADEVICE0: Sample rate approx. 48.0 k, 0 errors, receive audio level CH0 61

ADEVICE0: Sample rate approx. 47.9 k, 0 errors, receive audio level CH0 63

ADEVICE0: Sample rate approx. 48.0 k, 0 errors, receive audio level CH0 62

ADEVICE0: Sample rate approx. 48.1 k, 0 errors, receive audio level CH0 52

ADEVICE0: Sample rate approx. 47.9 k, 0 errors, receive audio level CH0 55
```

The reported sample rate will vary a little, especially for short collection intervals. This is because the audio samples are transferred in large blocks for efficiency rather than a steady stream of one at a time. The last number on the line is the audio level. It should be somewhere in the range of 10 to 100 when hearing static.

9 Troubleshooting

9.1 General

The first thing you want to do is look at what direwolf says when it starts up.

The optional features included are listed near the beginning. Example:

```
Includes optional support for:  gpsd hamlib cm108-ptt
```

If you want to use hamlib, and it is not listed there, you will need to install hamlib then rebuild from source.

If you want to use a USB audio adapter, with built in PTT control, the "cm108-ptt" optional feature must appear here.

Next we have information about how the modems and PTT are configured. Linux example:

Windows example:

```
Available audio input devices for receive (*=selected):
    0: Microphone (2- C-Media USB Head
Available audio output devices for transmit (*=selected):
    0: Speakers (Realtek High Definiti
    1: DELL U2410-1 (NVIDIA High Defin
    2: Speakers (2- C-Media USB Headph
    3: Realtek Digital Output (Realtek
Channel 0: 1200 baud, AFSK 1200 & 2200 Hz, A+, 44100 sample rate.
Note: PTT not configured for channel 0. (Ignore this if using VOX.)
```

In this case, "ADEVICE" was not specified, in the configuration file, so there is no "*" indicating the selected device. The first one, in each group is taken as the default. What you probably want is:

ADEVICE USB

1200 bps is the xxxxxxxxxxxxxxxxxxxxxxxxxx

xxxxxxxxxx

If not what you were expecting, review the configuration file settings. Check out:

- The wiki https://github.com/wb2osz/direwolf/wiki

- Discussion forum https://groups.io/g/direwolf

You might find your answer there. If not, ask the group for help. Your question, and any answers will help others in the future. Be sure to include a good description any relevant details.

Do not use the github issues section to ask for help. It is for reporting software defects and making enhancement requests.

9.2 Not receiving anything

Check to see if received audio is getting in at an appropriate level.

Elaborate. Not muted.

9.3 Transmitter turns on but no one receives my signal

Listen with another receiver. Does your signal sound like the others? Adjust the transmit audio level to be similar to others.

9.4 Transmitter says on

This is usually caused by RF, from the transmitter, getting back into the PTT circuit and it latches on.

Move the radio, and especially the antenna, farther away from the computer.

Put ferrite chokes on the cable to stop the RF from getting back to the computer.

Some people like to use audio transformers to isolate the radio from the computer. An optoisolator is used for PTT to break up the ground loop.

9.5 System crashes when transmitting

This is a more severe case of the RF getting into your computer.

9.6 "aplay –l" complains, "no soundcards found…"

I had a situation where user "root" could see the devices but an ordinary user could not. The solution was to add the user name to the "audio" group like this.

sudo addgroup john audio

This will not take effect immediately. Log out and log in again.

9.7 Permission problem with USB Audio Adapter GPIO

Normally, all of /dev/hidraw* are accessible only by root.

```
$ ls -l /dev/hidraw*
crw------- 1 root root 247, 0 Sep 24 09:40 /dev/hidraw0
crw------- 1 root root 247, 1 Sep 24 09:40 /dev/hidraw1
crw------- 1 root root 247, 2 Sep 24 09:40 /dev/hidraw2
crw------- 1 root root 247, 3 Sep 24 09:50 /dev/hidraw3
```

Unnecessarily running applications as root is generally a bad idea because it makes it too easy to accidentally trash your system. We need to relax the restrictions so ordinary users can use these devices.

If all went well with installation, the **/etc/udev/rules.d** directory should contain a file called **99-direwolf-cmedia.rules** containing:

```
SUBSYSTEM=="hidraw", ATTRS{idVendor}=="0d8c", GROUP="audio", MODE="0660"
```

I used the "audio" group, mimicking the permissions on the sound side of the device.

```
$ ls -l /dev/snd/pcm*
crw-rw----+ 1 root audio 116, 16 Sep 24 09:40 /dev/snd/pcmC0D0p
crw-rw----+ 1 root audio 116, 17 Sep 24 09:40 /dev/snd/pcmC0D1p
crw-rw----+ 1 root audio 116, 56 Sep 24 09:50 /dev/snd/pcmC1D0c
crw-rw----+ 1 root audio 116, 48 Sep 29 18:14 /dev/snd/pcmC1D0p
```

Notes:

- The double equal == is a test for matching.
- The single equal = is an assignment of a value. In my limited experience, Debian type systems can accept either = or := but Red Hat type systems recognize only the = form.

If you don't have /etc/udev/rules.d/**99-direwolf-cmedia.rules**, create it as shown above and reboot.

Now we observe that the /dev/hidraw*, corresponding to the USB-Audio device now has permissions that allow us to access it.

```
$ ls -l /dev/hidraw*
```

```
crw-rw---- 1 root audio 247, 0 Oct  6 19:24 /dev/hidraw0
crw------- 1 root root  247, 1 Oct  6 19:24 /dev/hidraw1
crw------- 1 root root  247, 2 Oct  6 19:24 /dev/hidraw2
crw------- 1 root root  247, 3 Oct  6 19:24 /dev/hidraw3
```

9.8 Baofeng transmitter stays on ¼ second after PTT released

Disable the STE setting. Menu item 35????

9.9 DINAH transmit volume mysteriously decreases by itself

The receive carrier operated squelch (COS) is connected to the volume down pin on the USB audio chip. You need to cut the JP-3 jumper so the COS signal, from the radio, does not shut down the transmit audio.