Getting Started with RadioHat

These of course are only ad hoc temporary instructions to get you going. This is not intended to be a normal end-user experience. These are all hardware testing experimental programs. If you have any questions or problems figuring this out, contact me and I'll talk you through what you need.

Most of the final configuration for the hat will be written into the EEPROM on the card and automatically handled at boot by Raspbian, but for now the EEPROM is blank so we have to do a few extra things before we use the card.

1. Download latest 32 bit Raspbian, configure it with your passwords and options and update it. In raspi-config, make sure you enable the I2C, I2S and serial port with RTS/CTS interfaces.

2. Install the following to allow normal operation and modifications :
    1. libncurses-dev by "sudo apt-get install libncurses-dev"
    2. gnuradio by "sudo apt-get install gnuradio"
    3. Audacity using "apt-get install audacity"
    4. pavucontrol using "sudo apt-get install pavucontrol-qt"
    5. fldigi using "sudo apt-get install fldigi"
    6. pulse audio jack redirector using "sudo apt-get install pulseaudio-module-jack"
    7. WSJTX by downloading latest from their home page and double clicking on .deb file
       if it fails, you may need to do it this way (to install missing dependencies):

       ```
        wsjtx_2.3.1_armhf.deb
       sudo apt-get --fix-broken install
       ```

3. Create a folder called "radiohead" in your home directory
    1. Unpack "radiohead.zip" contents into that folder
    2. sudo cp ~/radiohead/RPI as\ SLAVE/ IIS\ setup/ RaspberryPi_I2S_Slave-master/genericstereoaudiocodec.dtbo /

boot/overlays/genericstereoaudiocodec.dtbo

4. Add the following to the end of /Boot/config.txt:
   ```
   # Radiohat 0.9
   dtoverlay=i2c-rtc,ds3231,addr=0x68
   dtoverlay=dwc2
   dtparam=i2s=on
   dtoverlay=genericstereoaudiocodec

   # initialize the gpio pins for RX and TX enabling the mixers
   gpio=22,23=op,dh
   # initialize the serial port RTS pin
   gpio=17=a3
   ```

5. To enable launching shell scripts from the file manager you must first execute this command to deal with a bug in Raspbian “Buster”:
   “sudo cp /usr/bin/lxterminal /usr/bin/xterm”

6. Edit /lib/udev/hwclock-set by changing HWCLOCKPARS line to read “HWCLOCKPARS= --delay=0.7” (requires sudo)

You now can perform some simple hardware tests:

1. Install the hat

2. Temporarily jumper the radiohat to use power from the pi by installing the power jumper between pins 9 and 10. This will produce very noisy IO, but is adequate for testing.

3. Connect a cellphone headset to the headset jack (or just a pair of headphones will do for now)

4. connect an antenna to the RF IN connector

5. Power up the pi and check for smoke, etc. You can check the voltages and voltage regulator chip outputs for 1.8 and 3.3V. The 5V regulator will be inactive in this mode, but 5V should be present on

the power header pin 9

6. Boot the Pi and connect it to the internet

7. This should show the RTC time (probably wrong):

```
sudo hwclock —verify
```

   You can set it by “sudo hwclock --systohc --delay=0.7”
   It should then show up in that program if you type “timedatectl”

8. You now should be able to run the following command from two separate shells (to keep both displays open):

1.

```
~/radiohead/pitrans1/pitrans
```

```
~/radiohead/headset.py
```

   If all goes well, you will see a window generated by gnuradio containing some simple controls and instrumentation graphs in a tabbed display and may even hear signals at the frequency shown in the pitrans window. If yo

1. u instead see audio errors in the parent shell window that launched headset.py, you will need to troubleshoot the audio configuration.

2. You should now be able to tune around a bit using cursor keys in the frequency control of the pitrans window and pla

1. y around with some of the other controls. Don’t modify any pitrans audio controls for now.

   If you have a load connected, you can experiment with transmitting using the “t” and “r” commands. If you have a headset connected, its microphone and the microphone gain in pitrans (has only 4 steps) should be active during transmit - otherwise the tone oscillator should work to generate output.

You will probably need/want to recompile pitrans with different options. If you've properly installed the needed tools, you can do this:

```
cd ~/radiohead/pitrans1
c++ pitrans.cpp si5351.cpp -lncurses -L/usr/local/lib -lwiringPi
```

You can add "-D WWV10" to the end of the compilation command to more quickly check the frequency calibration by causing startup to always occur at 10Mhz.

If you make an ugly multicommand line like this:

```
clear; cd ~/radiohead/pitrans1/ ; c++ pitrans1.cpp si5351pi.cpp -lncurses
-L/usr/local/lib -lwiringPi -D WWV10 -D CALFACTOR=10000 ; ./a.out
```

you should be able to easily determine the crystal error and try different CALFACTORs in parts per billion to get a perfect zero beat on WWV in "headset.py". A good starting point is the value written on the cards serial number label. It will vary with the cards temperature, so you want to finalize this in the final packaging and environment.

Using a mouse, you can expand the spectrum display around the zero frequency peak tip to make this easier. Don't forget the tuning correction you make is NOT the same as the crystal error. It needs to be scaled by the relationship between the crystal frequency (nominally 27 MHz and test frequency 10 MHz.

You can type q to quit from pitrans then use up arrow to recall the shell command for modification. This will allow you to compile with different for CALFACTORs and try the whole thing again. Headset.py will keep running until the new version of a.out starts (it's still using the last value loaded into the synthesizer chip).

Once you are happy with the value, edit pitrans.cpp to use the final value in the line near the beginning of the code that reads:

```
long gCalibrationFactor = nnnnn
```

Recompile one more time WITHOUT the -D options and then copy a.out to “pitrans” (you could rename the old pitrans first if you want to keep it around).

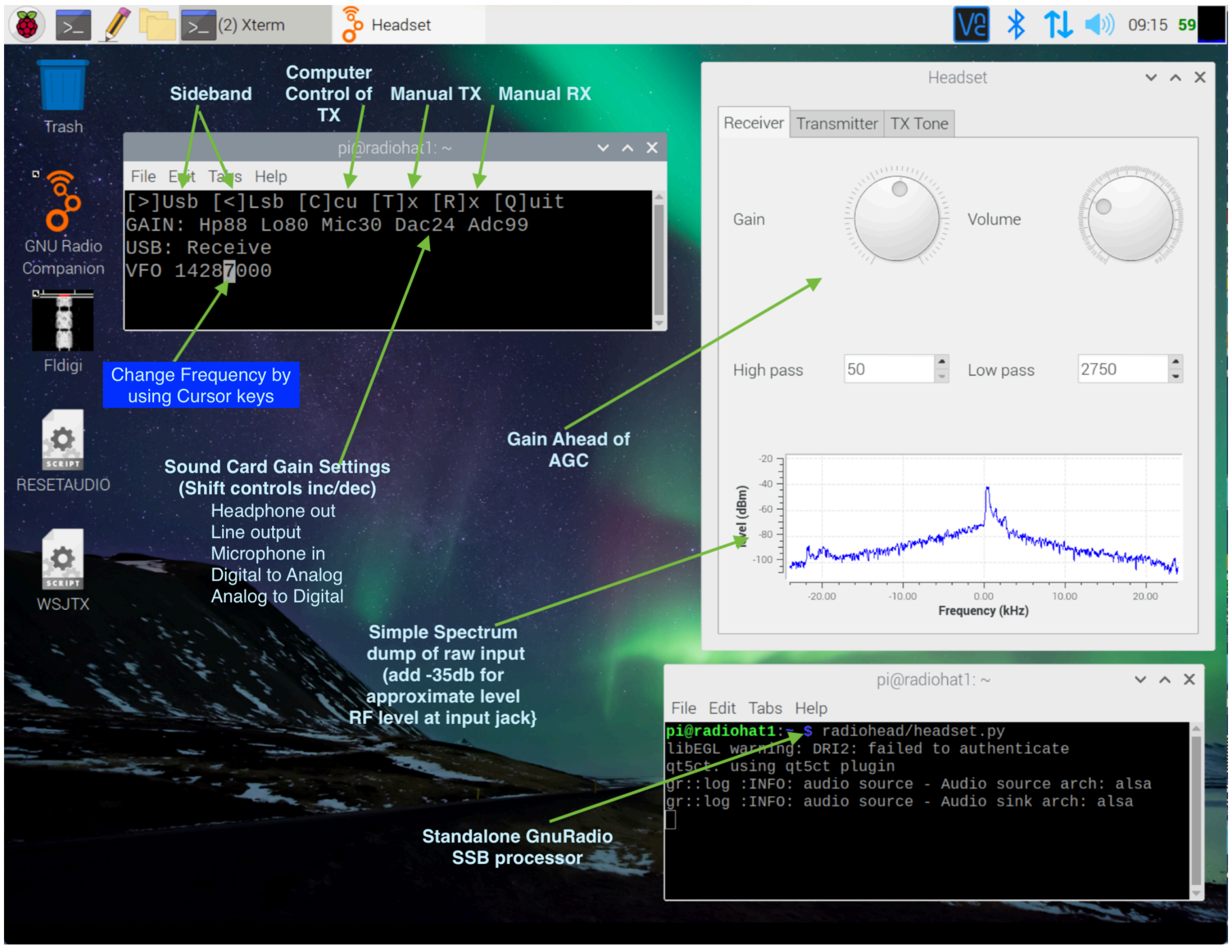
Obviously this is just a kluge for debugging hardware - not the way to do this in end-use software, but it contains all the tools needed to easily test the hardware and take measurements, and even to try some QSO’s if you are connected to an amplifier board, low pass filter and TR switch. If you have an ads1115 based VSWR sensor you can even modify pitrans to use it on transmit!

There are two other SSB DSP programs for gnu radio included that use various generic USB devices for headset IO to simplify operation. These don’t need to share the Audio IO connections between the Radio IO and the Headset, so they’re a little easier to experiment with. The one called USBDevice works with most cheap generic USB headset dongles and mini audio cards.

If you open the GnuRadio Companion program, it will allow you to edit any of these files in their source (.grc) form and by running them from that environment you will update the standalone .py files.

If you find GnuRadio Companion is lagging seconds behind mouse operations, you may need to disable the experimental 3D drivers that are turned on in the Pi’s config file by default. Do this by commenting out this line in your config.tx:

```
#dtoverlay=vc4-fkms-v3d
```



```
[>]Usb [<]Lsb [C]cu [T]x [R]x [Q]uit
GAIN: Hp88 Lo80 Mic30 Dac24 Adc99
USB: Receive
VFO 14287000
```

High pass 50   Low pass 2750

```
pi@radiohat1:~ $ radiohead/headset.py
libEGL warning: DRI2: failed to authenticate
qt5ct: using qt5ct plugin
gr::log :INFO: audio source - Audio source arch: alsa
gr::log :INFO: audio source - Audio sink arch: alsa
```

Another document describes operation of the digital mode test environment.