13-February-2022 by Mario Vano AE0GL

Getting Started with *RadioHat*

This document describes one way to perform a simple installation of the *RadioHat* board onto a Raspberry Pi4, and add and operate the software needed to test it.

This is not meant to be a cookbook. The instructions assume you are familiar with downloading files, expanding them, and using the Raspberry Pi's command line interface. Be very careful with all connections to and from the Pi, as common causes of damage to Pi's include misconnection of wiring or application of incorrect voltages.

The procedures described here and this hardware are experimental and in the event of failures or errors may present some risk to the Pi - it's obviously not a good idea to use a "mission critical" board for this kind of work.

This guide was tested while it was being written using a Raspberry Pi4 with 4Gb of memory installed in a *Geekworm Passive Cooling* case and a 16Gb μSD card. Most other configurations should work just as well.

We are going to power the *RadioHat* from the Pi's on-board 5V supply. This is a simple way to confirm that the board is working, but for most uses, it is not a good way to power things. That 5V power is very noisy and will introduce some low frequency signals into the *RadioHat*'s receiver. We can ignore this problem while we get started with the card.

Prepare the Raspberry Pi and μSDHC card

- **Download and install** a copy of the latest 32 bit default release of Raspbian (*bullseye* at this time) into a 16Gb or 32Gb μSDHC card using the official *Raspberry Pi Imager* from the RaspberryPi official web site.

- **Install the μSD card** into the Pi's SDHC slot, Boot up, test the installation and do basic configuration including setting display preferences, hostname, networking and a new password. Make sure it is working properly before going on.

Install the *RadioHat* hardware

- **Shut down** the operating system and Pi and unplug all cables.

- **Jumper settings:** Install a bi-pin jumpers between pins 9 and 10 of J5 - the two pins closest to the GPIO connector edge of the board. If your board came with a jumper connected between pins 1 and 2 to protect the power pins, you can leave it installed. There should be no other jumpers installed on the board.

- **Carefully plug the board** into the GPIO connector, ensuring that you've got all the pins plugged in properly and that the board is not resting on anything that would cause a short circuit. Some cases with oversize heatsinks require the use of an "extension" pass-through connector to raise the board safely off the heatsink.

- **Reconnect** your display, keyboard, mouse and any network cabling then carefully power up the board and see if it boots up normally. If you see any problem at all or if the lights on the end of the Pi away from the USB connectors don't begin blinking right away, power down immediately and resolve the problem before proceeding.

Install software tools

You will need the following tools and libraries on your Pi:

1. *git* repository manager
2. *c++* Compiler
3. *dtc* Device Tree Compiler
4. *GnuRadio* with GnuRadio Companion
5. *ncurses5* development library
6. *WiringPi* development library

git, *c++* and *dtc* should already be present. If they are not, locate and install them.

Note that in newer versions of Raspbian the GnuRadio installation is now 3.8 instead of 3.7. This means that the *.py* files downloaded drom the *RadioHat* repository will no longer work since they were created by the older version.

Fortunately, if you open the corresponding *.grc* file for each *.py* file using *GnuRadio* and run it at least once, that file will be recreated for use with version 3.8 and will once again work as a stand alone program.

- **Install GnuRadio** from the Raspbian default repository using the command:

  ```
  sudo install apt-get gnuradio
  ```

- **Install ncurses5** with the command:

  ```
  sudo apt-get install libncurses5-dev
  ```

- **WiringPi** has become obsolete since the time *pitrans* was written, however, it is still needed for this software suite. This is probably not a bad thing, but we will have to work around it for now.

  You will need to install an open source replacement. The library name in c++ linker directives is now slightly different - use *-lwiringPiDev* instead of *-lwiringpi*. Install this version using the following commands:

  ```
  cd ~/.Downloads
  git clone https://github.com/WiringPi/WiringPi.git
  cd wiringPi
  ./build
  ```

Download and install the *RadioHat* folder

Make sure you are in your home directory (usually */home/pi*) and install the *RadioHat* support tools folder there using one of the two procedures shown below. (Using git to clone the repository is the easiest and best way to do this).

If you've done everything properly, you should now have a folder named "radiohat" in your *pi* home directory containing the contents of the *RadioHat* github repository. If it has any other name you may need to rename it correctly before proceeding.

- **install using *git*:**

```
git clone https://github.com/mpvano/radiohat.git
```

- **…or install by downloading a *.zip* file:**

    1. Start a web browser and type the same URL into its address bar. This should take you to the repository home page. Then, click on the green button labeled *Code*. A popup menu should appear containing several choices. Choose *Download zip*.

    2. When the download finishes, you should find a file named *radiohat-main.zip* in your Downloads folder. change your directory to the browser's *Downloads* folder and unzip that file using the command

    ```
    unzip radiohat-main.zip
    ```

    3. You should now see a folder called *radioHat-main* in your Downloads folder. Rename it to eliminate the suffix by using the command

    ```
    mv radiohat-main radiohat
    ```

    4. Confirm that this worked by using the *ls* command.

    5. Change back to your home directory and move the folder there using the commands:

    ```
    cd
    mv Downloads/radiohat .
    ```

Install the Audio Codec Driver Overlay

The *RadioHat*'s audio codec uses drivers that are already present in Raspbian. It directs the operating system to use them by installing a file called a *Device Tree Overlay* into your *boot* folder. You must copy this overlay file to the folder where Raspbian keeps them. You will then add a line to *boot.txt* telling the operating system to use it.

After booting, the Audio Codec device will appear in programs but it won't function properly until the devices on the *RadioHat* are initialized. For now, we're just going to manually initialize them by running the *pitrans* program.

Running *pitrans* needs to be done at least once after each reboot to initialize the codec chip so that it can be used as a sound interface. This can easily be automated, but for now, we're just going to do it manually since pitrans will be run alongside any other program we use to test *RadioHat*.

- **Install the overlay** by giving the following commands:

```
cd ~/radiohat/RPI as\ SLAVE/ IIS\ setup/
cd RaspberryPi_I2S_Slave-master/genericstereoaudiocodec.dtbo"
sudo cp genericstereoaudiocodec.dtbo  /boot/overlays/genericstereoaudiocodec.dtbo
```

These folder and file names look hard to type, but if you use the shell's tab expansion feature you can type just a few letters of each name and then hit the *Tab* key to do the rest of the work!

- **Add lines** to the Pi's */boot/config.txt* file

  Several lines need to be added to this file (which controls the Pi's startup options) to enable the *RadioHat's* devices. You can use either the *mousepad* or *nano* editor to do this, but you must run the editor using *sudo*. Here's how to do this using *mousepad*:

- **Open your editor** using the command:

```
sudo mousepad /boot/config.txt
```

- **add the following lines** to the end of the file. (startup of the optional clock/calendar chip is shown disabled in this example):

```
# *RadioHat* 1.0
#for the optional rtc
#dtoverlay=i2c-rtc,ds3231,addr=0x68
dtparam=i2c=on
dtparam=i2s=on
dtoverlay=genericstereoaudiocodec

# initialize the *RadioHat* GPIO
gpio=22,23=op,dh
gpio=17=a3
```

Compiling the *pitrans* VFO and Codec control program

You must recompile *pitrans* at least once to ensure that it uses the correct libraries for your machine. It will probably just crash if you do not do this! These instructions will also be useful if you decide to make modifications to its behavior.

- **Compile** *pitrans* with these commands:

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

- You can optionally add *-D WWV=n* where *n* is *5*, *10* or *15* to the end of the compilation command to more quickly check calibration by causing startup to occur tuned to WWV.

- Once you're happy with the results, you can copy the resulting *a.out* to *pitrans*, *pitrans1* and/or *pitrans2* or these can be alternate versions:

```
cd ~/radiohat/pitrans1
cp a.out pitrans
cp a.out pitrans1
cp a.out pitrans2
```

- There is no longer any need to compile calibration settings into *pitrans* as it now looks for a file named *CALFACTOR.txt* in the directory it loads from. This file should contain only a single line with a correction value in parts per billion.

  As a starting point, it can be set to the offset value written on the label adhered to the headphone

jack of *RadioHead* during testing. The value on the label was measured using the si5351 load capacitance set for 8pf as specified by the crystal manufacturer. It's possible that by compiling with the load capacitance set to 10pf the correction required may be lower.

Configure the clock/calendar option if installed

If you have the optional DS3231 clock calendar installed, and if you've uncommented the line we added to /boot/config.txt that loads its driver, then you should also edit (using *sudo*) the file */lib/udev/hwclock-set*.

Change:

```
HWCLOCKPARS
```

to read:

```
HWCLOCKPARS= --delay=0.7
```

Testing

1. Connect a cellphone headset or a pair of headphones to the 3.5mm jack on *RadioHat*.

2. connect an antenna (or signal generator) to the RF IN connector

3. Power up the pi and recheck for proper operation. You can check the voltages and voltage regulator chip outputs on *RadioHat* for 1.8 and 3.3V. The 5V regulator will be inactive in this mode, but at least 4.5V should be present on power header's pin 9 (this is dropped about .4V by the series schottky diode on *RadioHat*.

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

   ```
   ~/radiohat/pitrans1/pitrans
   ```

   then:

   ```
   ~/radiohat/headset.py
   (or open and run *headset.grc* from *gnuradio companion*)
   ```

   If all goes well, you will see a QT window that contains some simple controls and instrumentation graphs in a tabbed display and you may even hear signals at the frequency shown in the pitrans window.

   If you instead see audio errors in the parent shell window, you will need to troubleshoot the audio configuration. the *Audacity* editor should be of some use in doing this, since the RadioHead devices should work fine with it. The board defaults to feeding audio from the QSD to the input device in the *Audacity* audio editing program if you have it installed and accepting output from Audacity to send to the headset earpieces.

5. Once everything is working, you should be able to **tune around** a bit using cursor keys to change the frequency control section of the pitrans window and keyboard commands to control its other features. In general, upper case letters increment values and lower case ones decrement them. Don't

modify any pitrans audio controls for now.

6. If you have a load connected, you can experiment with **transmitting** using the *t* and *r* commands in *pitrans*. If you have a headset connected, its microphone and the microphone gain in *pitrans* work during transmit - otherwise the tone oscillator tab can be used to generate output.

7. connect the Pi to the internet

8. If you have the optional clock/calendar installed you can **do a quick check of it**.

   This command should show show the RTC time (probably wrong):

   ```
   sudo hwclock –verify
   ```

   You can set it with:

   ```
   sudo hwclock --systohc --delay=0.7
   ```

   It should then show up correctly in:

   ```
   timedatectl
   ```

Fixing Pi Problems

- 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
  ```

- In some versions of Raspbian, you may need to work around a bug that prevents launching shell scripts from the file manager. If you encounter this problem, try creating a fake copy of the missing terminal program like this:

  ```
  "sudo cp /usr/bin/lxterminal /usr/bin/xterm"
  ```

Next steps

Obviously this configuration is just intended for use debugging hardware! These tools should not be delivered as end-user software. They do, however, contain what you usually need to run tests and take measurements, and even to try some awkward QSO's if you are connected to an amplifier board, low pass filter and TR switch.

pitrans supports adding an ads1115 based VSWR sensor that you can modify for scaling to display your amplifier's output.

There are a few other *gnuradio* DSP programs enclosed that use various generic USB devices for headset IO. If you use an additional headset/microphone device you won't need to share their Audio IO paths with the *RadioHat's* IQ signals, so there will be no need to worry about reconfiguring the Codec chip when going from receive to transmit.

The version called *USBDevice* works with most generic USB headset dongles and mini audio cards.

There are also two programs called *osc.grc* and *twotone.grc* that are useful in transmitter testing. They do not use any other analog IO, so they're quite low in distortion.

If you open the *.grc* file for one these programs using *GnuRadio* you can edit it and then by running it from that environment you will recreate an up-to-date standalone *.py* file.