"From scratch" install of piHPSDR on a RaspBerry Pi or a TinkerBoard

In at least one case, the compilation of piHPSDR failed in the very beginning since the `make` command was not found. I looked around and did not find even a "minimal" Raspian image that does not contain the `make`, `gcc`, `git`, and the `pkg-config` packages (which are part of the core utilities). Although it is certainly possible to install all required packages, the existence of such a "tweaked" system possibly has the reason that the whole system must fit on a small (4 GByte) micro-SD card, so I strongly recommend to remove this card and store it in a safe place and buy a larger (8 or 16 GByte) micro-SD card (costs less than 10 bucks) and install a new system. I actually did this for documentation purposes.

The TinkerBoard comes with a built-in 16 GB eMMC, so you do not need to get a micro-SD card (although it is possible to use such cards with the TinkerBoard).

Here you find a step-by-step protocol how to initialize the card, download the necessary software, compile WDSP and piHPSDR, and finally running it.

Note that these instruction are also largely valid for compilation of piHPSDR on a normal LINUX desktop or laptop computer, with the exception that GPIO must not be activated in the Makefile (see below).

Step A: Obtain Raspian or TinkerOS Image

RaspPi: From the web page `https://www.raspberrypi.org/downloads/raspbian`, obtain the file `2019-09-26-raspbian-buster.zip` which is deonoted as the "Raspian buster with desktop" image. Note that the release date which is encoded in the file name may vary. This is a compressed disk image, so un-zip this file such that it becomes the file `2019-09-26-raspbian-buster.img` which is about 3.6 GByte long.

While I am sure that there are other sources of suitable image files, the following protocol (instructions) have been tested with exactly this one.

TinkerBoard: There is a bunch of different LINUX-type systems for the TinkerBoard. This description refers to the "official" one obtained from the web page

`https://www.asus.com/uk/Single-Board-Computer/Tinker-Board/HelpDesk_Download/`

From there I got the compressed image file (about 1.13 GByte long) with file name `20190821-tinker-board-linaro-stretch-alip-v2.0.11.img`. Again, the release date encoded in the file name may vary.

Step B: "Burn" this image file on-to a micro-SD card (RaspPi) or the internal eMMC (TinkerBoard)

How to do this varies depending on which computer you are using. Detailed instructions how to "burn" an image to an SD card from, say, a computer running various operating systems can be found on the internet, see for example

`https://www.raspberrypi.org/documentation/installation/installing-images`

Note that "burning" can take several minutes, since the I/O speed is about 10 MB/sec on most cards.

Step C: First-time boot

RaspPi: The micro-SD-card was then inserted in the RaspPi and the machine booted (with keyboard, mouse and monitor attached). The RaspPi should be connected to a router with a DHCP server via an Ethernet cable.
The system boots, asks for the country/timezone, and for the password of the default user "pi". It automatically connects to the internet and updates all installed software. When this is complete, the system should be restarted.

In the following, it is implied that you open a terminal window to type the LINUX commands shown. The commands are indented and shown in blue.

TinkerBoard: Connect the TinkerBoard to Keyboard, Mouse and Ethernet and power up. Nothing has to be done.

Step D: Install required software packages

Although this has just been done, one should keep the system up-to-date before installing any *additional* packages, so open a terminal window and issue the commands

```
sudo apt-get update
sudo apt-get upgrade
```

It is a good idea to restart the system at this place, especially if the kernel has been updated. As a result, you have a "plain vanilla virgin" Raspian system with all installed software upgraded to the latest version. Although I have not seen a pre-installed system lacking the "git" and "pkg-config" packages, just to be sure load them:

```
sudo apt-get install git
sudo apt-get install pkg-config
```

For compiling WDSP, you need the fftw3 library, so issue the command

```
sudo apt-get install libfftw3-dev
```

For compiling piHPSDR, you need tons of additional libraries. Fortunately, installing a component automatically installs are prerequisites, so we need only very few commands. Note that what is pre-installed may differ between different distros, so some of the following commands may actually report that the package is already there. Furthermore, some of the libraries may only be necessary when compiling with special options (e.g. libusb for the USBOZY option that supports *very* old SDR radios connected by an USB cable, and libcurl4 for the STEMLAB_DISCOVERY option that is meant to start the SDR app on RedPitaya-based radios).

```
sudo apt-get install libgtk-3-dev
sudo apt-get install libasound2-dev
sudo apt-get install libcurl4-openssl-dev
sudo apt-get install libusb-1.0-0-dev
sudo apt-get install wiringpi
```

Note on RaspPi-4 and GPIO:

If you want to use the piHPSDR controller and/or a Morse key attached to the GPIO, you need the latest version of the wiringPi library. The version can be checked with the command

```
gpio -v
```

and it should be at least version 2.52. To install the latest version from the wiringPi web site, use the commands

```
cd /tmp
wget https://project-downloads.drogon.net/wiringpi-latest.deb
sudo dpkg -i wiringpi-latest.deb
```

But it is still not working, because (at least in version 2.52) the wiringPi library does not set the pull-up options correctly for the new Broadcomm chip used in the RPi-4. So you have to set either manually or permanently at startup the GPIO pin modes. Here is the instruction to do so (info provided by John Melton):

First you need to disable all Interfaces except SSH and I2C using Raspeberry Pi Configuration->Interfaces. You may see the Serial Console is enabled but is grayed out so you cannot disable it. To disable it you need to edit `/boot/cmdline.txt` and remove the text "`console=serial0,115200 console=tty1`".

To fix the problems of WiringPi not being able to set the GPIO pullups correctly you can add the following 2 lines to `/boot/config.txt`

```
# setup GPIO for piHPSDR Controller 2
gpio=4-13,16-27=ip,pu
```

Now reboot the system and all should be OK.

Note that editing the two files requires administrator privileges, so I did it using the commands

```
sudo vi /boot/cmdline.txt
sudo vi /boot/config.txt
```

Step E: download, compile and install WDSP

Go to your home directory and dowload WDSP using these commands:

```
cd
git clone https://github.com/g0orx/wdsp
```

Then, go to the wdsp directory just created and compile and install:

```
cd wdsp
make -j 4
sudo make install
```

Step F: download/adjust/compile piHPSDR

Go to your home directory and download piHPSDR using these commands:

```
cd
git clone https://github.com/g0orx/pihpsdr
cd pihpsdr
```

The Makefile needs some adjustment before you start compilation. You need a text editor of your choice (I am old-school and always use `vi`). There are many lines that start with a number sign (#) and contain an equal sign (=) thus setting some variables that determine which features are compiled into the program. Here I give a comprehensive list of all features that I have enabled in my sample installation:

```
GPIO_INCLUDE=GPIO
PURESIGNAL_INCLUDE=PURESIGNAL
LOCALCW_INCLUDE=LOCALCW
STEMLAB_DISCOVERY=STEMLAB_DISCOVERY_NOAVAHI
MIDI_INCLUDE=MIDI
```

This means that the program is, for example, compiled with MIDI support which does not harm if you do not plan to use MIDI input devices.

Note 1: `STEMLAB_DISCOVERY` is an option to support RedPitaya-based SDRs where the SDR program on the RedPitaya has to be started through a web interface. If you do not use RedPitaya-based SDRs, leave the number sign in the first column of the two lines containing the key word `STEMLAB_DISCOVERY`, since this speeds up program start by about 15 seconds (since it is not tried to detect the STEMlab web server).

Note 2: On desktop or laptop computers running LINUX, the line containing `GPIO_INCLUDE` must be deactivated and thus read

```
#GPIO_INCLUDE=GPIO
```

Note 3: There is support for using devices that are supported by SoapySDR. These include the LimeSDR, PlutoSDR and RTLSDR (for the latter: receive only).

To compile support for SoapySDR the line in the Make file should read:

```
SOAPYSDR_INCLUDE=SOAPYSDR
```

If support for SoapySDR is not required the line in the Makefile should read:

```
#SOAPYSDR_INCLUDE=SOAPYSDR
```

It is advisable to compile all the components required for SoapySDR support rather than install then from the repository as the latest versions on github have several improvements and bug fixes. See the appendix at the end of this document for details of downloading and compiling these components.

After having modified the Makefile, the piHPSDR program can be compiled by the command

```
make -j 4
```

(note that the option "`-j 4`" indicates that the compilation can proceed on four CPU cores in parallel – you can omit this option then compilation takes considerably more time).

The program should be built without any errors.

Step G: creating Desktop icon

Normally you will want to have the piHPSDR program as a "click-able" program on the Desktop. To do so, start with the following commands

```
sudo chown root pihpsdr
sudo chmod u+s pihpsdr
cp relase/pihpsdr/hpsdr.png .
```

This makes piHPDR a "setuid root" application, which is recommended/necessary for programs that have access to the GPIO. The third command ensures that when starting piHPSDR, the HPSDR logo is nicely drawn in the top-left corner. Now, go to your desktop folder `/home/pi/Desktop` and edit a text file `pihpsdr.desktop` with the following content:

```
[Desktop Entry]
Name=piHPSDR
Icon=/home/pi/pihpsdr/release/pihpsdr/hpsdr.png
Exec=/home/pi/pihpsdr/pihpsdr
Type=Application
Terminal=false
Path=/home/pi/.pihpsdr
```

As soon as you have created this file (e.g. using the `vi` editor), an icon with text `piHPSDR` and the HPSDR logo should occur on your Desktop. In order to get rid of being asked each time whether the program should be run in a terminal window, you issue the command (within a terminal window, and in the directory `$HOME/Desktop`) the command `pcmanfm`. In the window that opens, you select the file `pihpsdr.desktop` and manoeuver to the `Edit -> Preferences -> General` menu tab, and check the option "`Don't ask options on launch executable file`".

Then, create the local working directory of piHPSDR and copy the HPSDR logo into it, using the commands

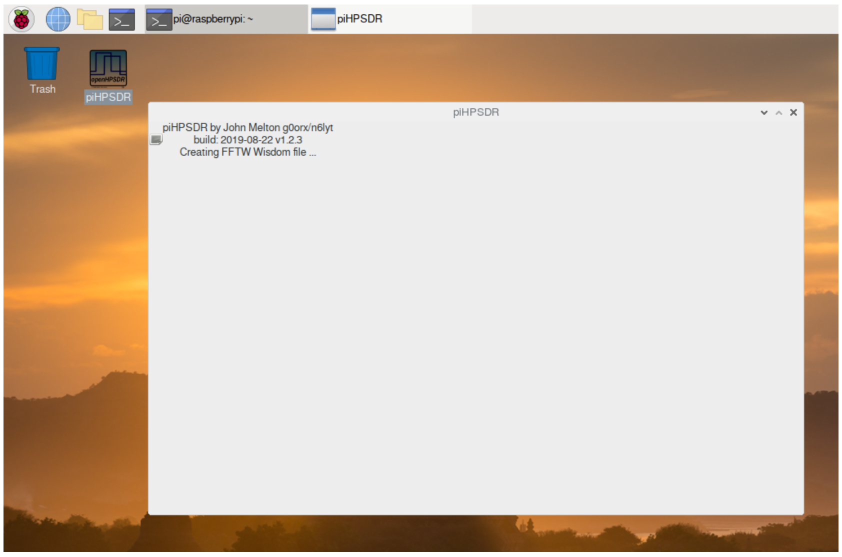
```
mkdir /home/pi/.pihpsdr
cp /home/pi/pihpsdr/release/pihpsdr/hpsdr.png /home/pi/.pihpsdr
```

Note: The `Path` option in the file specifies that the working directory of the piHPSDR program is `/home/pi/.pihpsdr`, which means that the WDSP wisom file (`wdspWisdom00`) as well as the configuration files (`*.props`) reside in that directory. Note that if you simply start piHPSDR from a terminal window using (e.g. for testing or debugging purposed), the current directory will be used as the piHPSDR work directory.
Having a separate directory to store the piHPSDR work files has the advantage that you can delete your entire `/home/pi/pihpsdr` directory, re-install it from github, compile the program and proceed with your previous settings.

TinkerOS case: For TinkerOS, the procedure is essentially the same. Replace "/home/pi" everywhere by "/home/linaro".

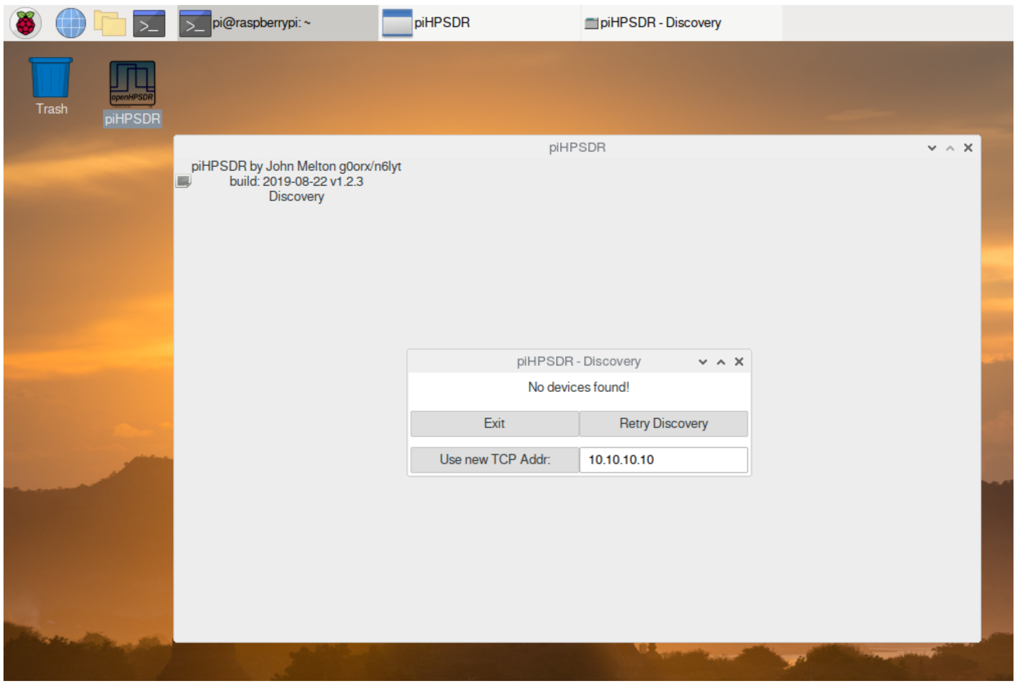
Step H: starting piHPSDR for the first time

Now you can start piHPSDR by double-clicking on the piHPSDR icon on your desktop. The piHPSDR window should open and look like this (if you have copied the hpsdr.png file into .pihpsdr, you will see the HPSDR logo in the top left of the piHPSDR window)



```
piHPSDR by John Melton g0orx/n6lyt
      build: 2019-08-22 v1.2.3
   Creating FFTW Wisdom file ...
```

Because it is the first time you started the program, the WDSP library determines (once and for all) the optimum way to do the fast-Fourier-transforms (this will take few minutes). After this time, the piHPSDR window looked like this:



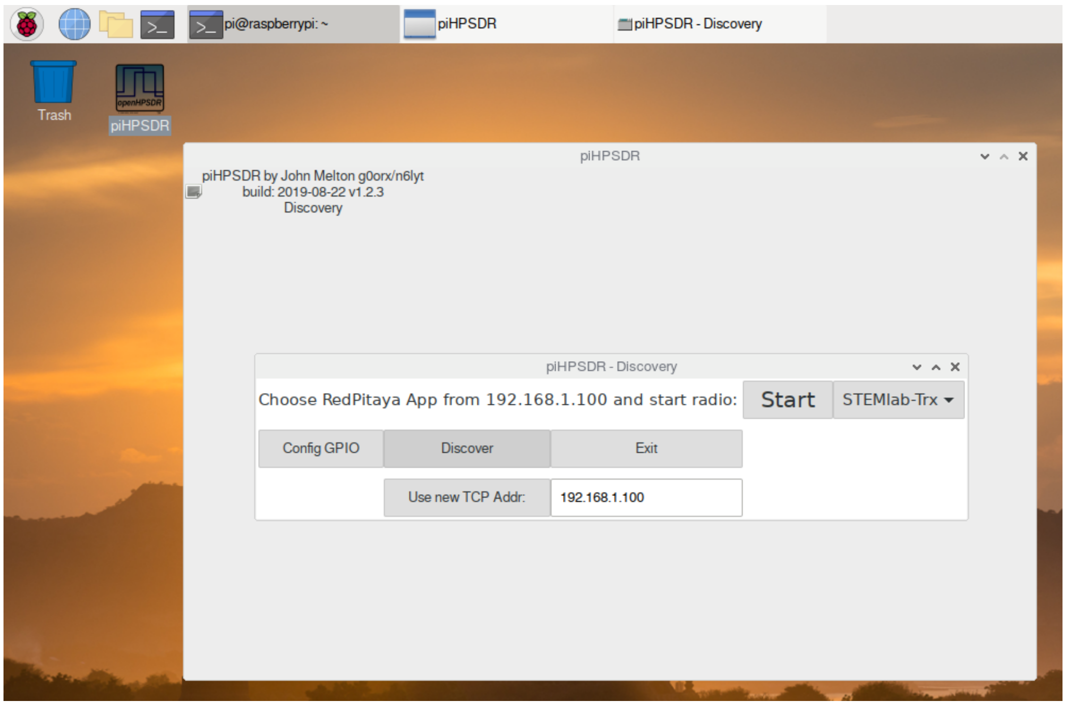
piHPSDR by John Melton g0orx/n6lyt
build: 2019-08-22 v1.2.3
Discovery

piHPSDR - Discovery

No devices found!

10.10.10.10

This is how the screen looks like if no SDR was attached. In my case, I had a RedPitaya based SDR (STEMlab/HAMlab) and for these radios, one must know its IP address. In my setup the (fixed) IP address of the HAMlab is 192.168.1.100 so I over-wrote the field reading "10.10.10.10" with that IP and clicked on "Use new TCP Addr", then the screen changed to



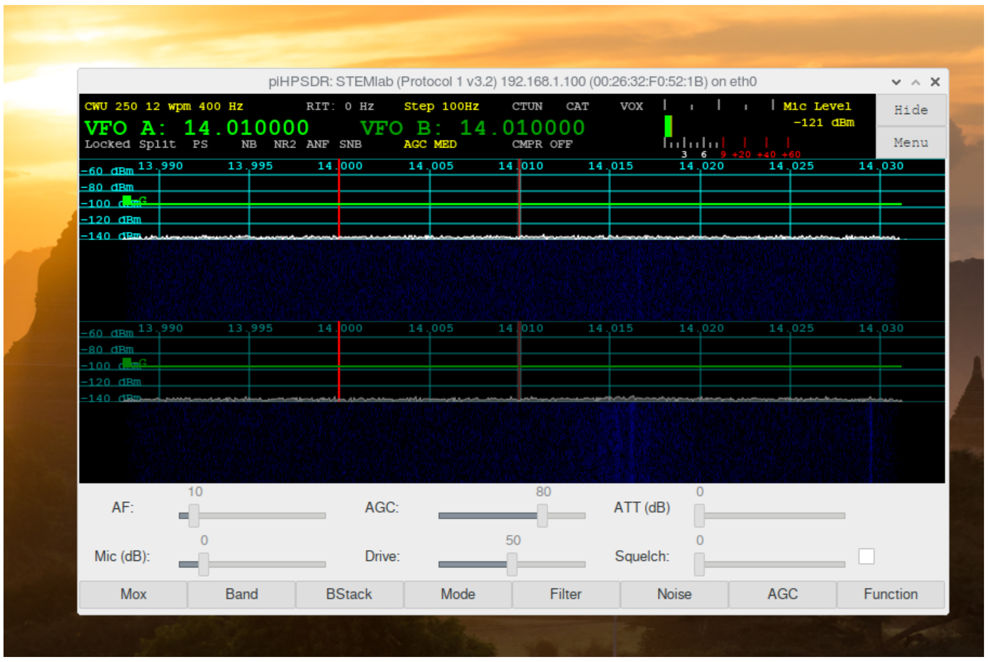
piHPSDR by John Melton g0orx/n6lyt
build: 2019-08-22 v1.2.3
Discovery

piHPSDR - Discovery

Choose RedPitaya App from 192.168.1.100 and start radio:

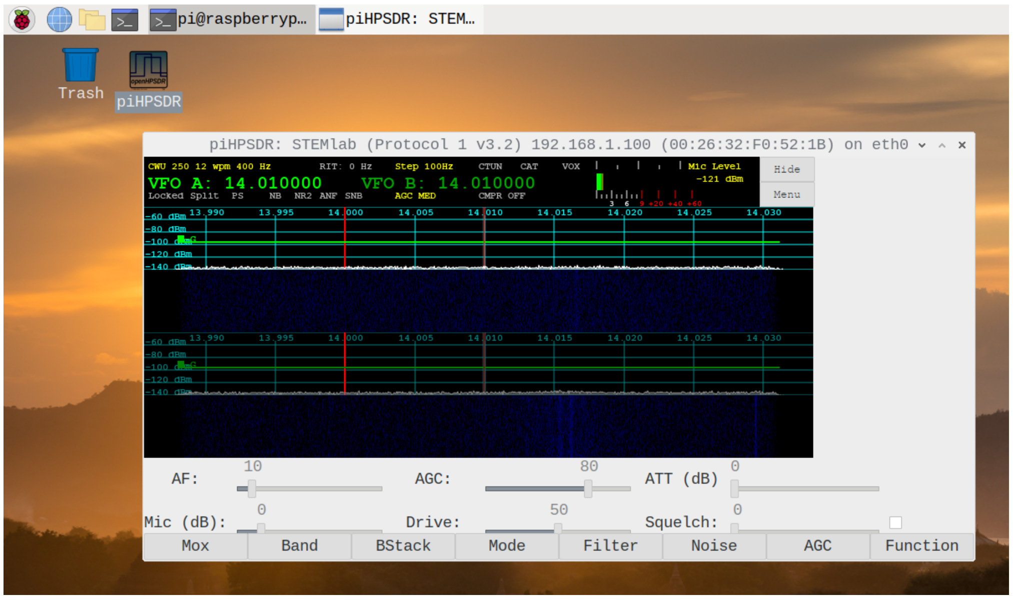
192.168.1.100

and after clicking the "Start" button, the radio finally started:



Some possible trouble with the font sizes (only RaspPi):

Some RPi users have reported that the radio window is messed up and looks like this:



This happens especially when using a large monitor. The reason is, that the system may automatically choose a large font when using a large monitor, which is not reasonable for piHPSDR since it is using a fixed-size window. This is easily fixed from the `Raspberry -> Preferences -> Appearance Settings` menu, in the window that opens you click the `System` bar and change the font to a small one, e.g. `FreeSans` with font size 10. Then immediately the piHPSDR window looks OK.

Step I: setting a fixed IP address

This step is not necessary as long you have both the RaspPi/TinkerBoard and the radio (e.g. the ANAN) connected to a router which offers a DHCP service. Personally, I like connecting the RaspPi and the ANAN *directly* by an Ethernet cable, and have a fixed IP address for both of them. Then I can do QSOs without having any IP routers or switches involved. For example, I use the fixed IP address 192.168.1.50/24 for my RaspPi and 192.168.1.99/24 for my ANAN. These were chosen such that the devices can also be run when connected to my router (for example, if the RaspPi should be connected to the Internet).

To enable a static fixed IP address, go to the directory `/etc/network/interfaces.d` (this one should be empty) and create a file with arbitrary name (e.g. "eth0") and contents (using the command "`sudo vi eth0`")

```
auto eth0
 iface eth0 inet static
 address 192.168.1.50
 netmask 255.255.255.0
 gateway 192.168.1.1
 dns-nameservers 192.168.1.1
```

Reboot the machine and you will have a fixed IP address (192.1681.50 in the example). Note that there is the possibility to set a fixed IP address through a graphical user interface, but that unfortunately did not work for me for an unknown reason.

Step K: creating loop-back "sound cards"

Loopback sound cards are interesting if you plan to run piHPSDR and a digimode program such as Fldigi or WSJTX on the same computer (say, both are running on a RaspPi, or both are running on a Linux laptop). In this case, you want to transfer audio from piHPSDR to the digimode program and back, and this is best done *directly*, that is, without ever going analog.

Since there is some misunderstanding around what this is and how it must be configured, imagine you have a conventional rig and a computer running a digimode program. To do so, you need two (physical) cables:

- cable #1: connects the headphone jack of the computer with the microphone jack of the TRX
- cable #2: connects the microphone jack of the computer with the headphone jack of the TRX

In reality these cables can transport audio in either direction, but you can imagine defining for each cable an "input side" and an "output side", e.g. marked with a red (input side) and a blue (output side) plug, and that you alway connect red plugs with

headphone jacks and blue plugs with microphone jacks. Then, audio always flows through the cables from the red (input side) to the blue (output side) plug.
A loopback device is a virtual sound card offering a "headphone/speaker" and a "microphone" device. This forms a virtual cable where the "red plug" (input side) is always associated with the headphone device and the "blue plug" (output side) is always associated with a microphone device. Audio data written to the "input side" of the device from one application can be read by another application at the "output side".

Since we need two cables, we need to have two such loopback "sound cards", which can be created with the following command in a terminal window:

```
sudo modprobe snd-aloop enable=1,1 index=4,5 id=vac1,vac2 pcm_substreams=2,2
```

The two "virtual audio cable" (vac) sound cards will have the numbers 4 and 5 and the names vac1 and vac2. I chose the numbers 4 and 5 since this leaves some room for other audio devices already connected. On the other hand it does not matter if devices in-between (e.g. with numbers 2 and 3) do not exist. You will see them using the command

```
      aplay -l
```

if you want. As the output of this command, you should see, among others (many lines deleted):

```
**** List of PLAYBACK Hardware Devices ****
card 0: ALSA [bcm2835 ALSA], device 0: bcm2835 ALSA [bcm2835 ALSA]
card 4: vac1 [Loopback], device 0: Loopback PCM [Loopback PCM]
card 4: vac1 [Loopback], device 1: Loopback PCM [Loopback PCM]
card 5: vac2 [Loopback], device 0: Loopback PCM [Loopback PCM]
card 5: vac2 [Loopback], device 1: Loopback PCM [Loopback PCM]
```

Now imagine that a digimode program and piHPSDR is running on the same computer. You will need two such virtual cables and the configuration is as follows:

piHPSDR:

- TX local microphone activated, device:	"cable #1 output side"
- RX local audio activated, device:	"cable #2 input side"

fldigi, wsjtx etc.:

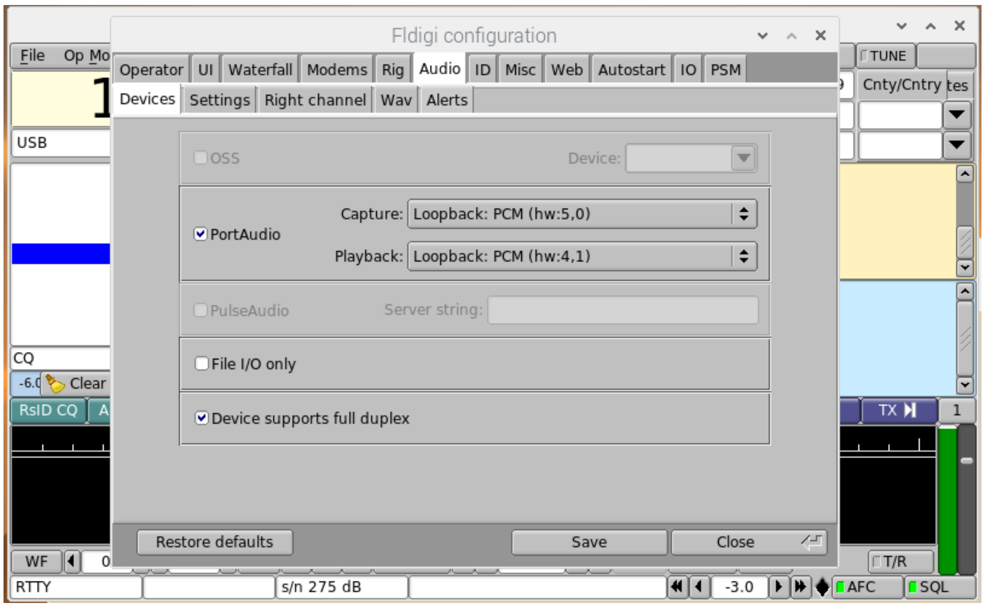
- Audio output (playback) device:	"cable #1 input side"
- Audio input (capture) device:	"cable #2 output side"

Unfortunately, the actual device names to be used for, e.g. "cable #1 input side" differ between the programs. Therefore we list the various possiblities, and the programs where we have "seen" this:

cable#1 input side	`Loopback: PCM (hw:4,1)`	Fldigi
	`plughw:4,1 Loopback`	piHPSDR
	`plughw:CARD=vac1,DEV=1`	WSJT-X

cable#1 output side	`Loopback: PCM (hw:4,0)`	`Fldigi`
	`plughw:4,0 Loopback`	`piHPSDR`
	`plughw:CARD=vac1,DEV=0`	`WSJT-X`
cable#2 input side	`Loopback: PCM (hw:5,1)`	`Fldigi`
	`plughw:5,1 Loopback`	`piHPSDR`
	`plughw:CARD=vac2,DEV=1`	`WSJT-X`
cable#2 output side	`Loopback: PCM (hw:5,0)`	`Fldigi`
	`plughw:5,0 Loopback`	`piHPSDR`
	`plughw:CARD=vac2,DEV=0`	`WSJT-X`

As an example, we provide screen-shots from Fldigi (SoundCard panel) and piHPSDR (RX and TX panels):



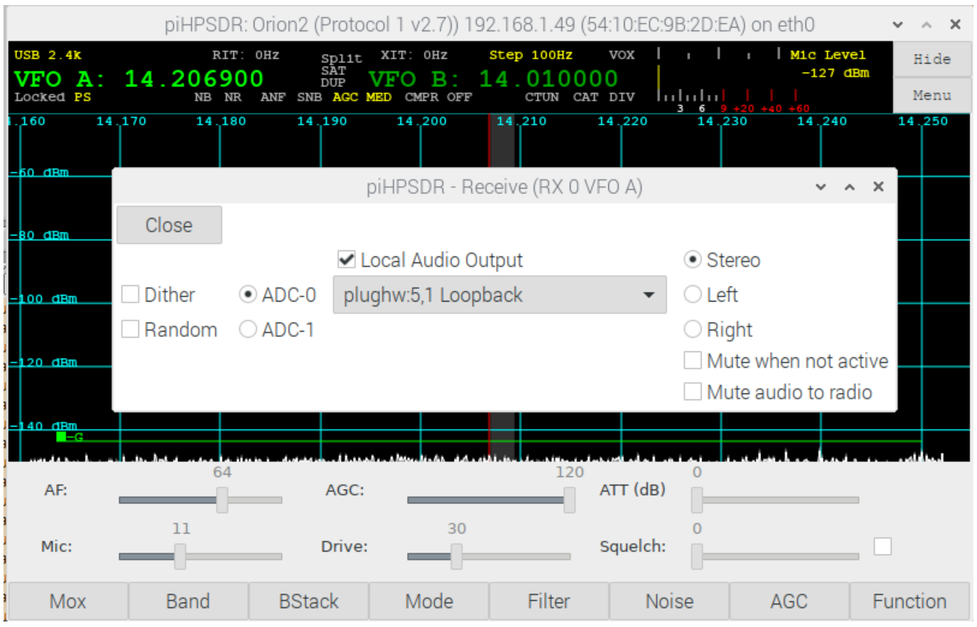
OSS Device:

[✓] PortAudio
Capture: Loopback: PCM (hw:5,0)
Playback: Loopback: PCM (hw:4,1)

PulseAudio Server string:

File I/O only

[✓] Device supports full duplex


piHPSDR: Orion2 (Protocol 1 v2.7)) 192.168.1.49 (54:10:EC:9B:2D:EA) on eth0

USB 2.4K RIT: 0Hz Split SAT DUP XIT: 0Hz Step 100Hz VOX Mic Level -127 dBm

VFO A: 14.206900 VFO B: 14.010000

Locked PS NB NR ANF SNB AGC MED CMPR OFF CTUN CAT DIV

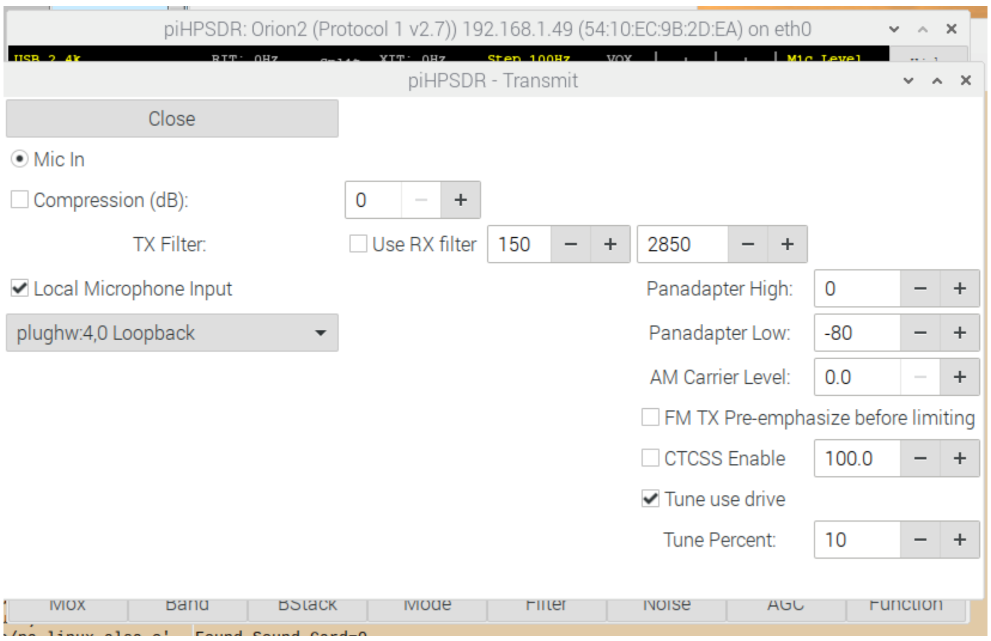
piHPSDR - Receive (RX 0 VFO A)

Close

- ☐ Dither
- ☐ Random
- ● ADC-0
- ○ ADC-1
- ☑ Local Audio Output: plughw:5,1 Loopback
- ● Stereo
- ○ Left
- ○ Right
- ☐ Mute when not active
- ☐ Mute audio to radio

AF: 64 AGC: 120 ATT (dB): 0

Mic: 11 Drive: 30 Squelch: 0


piHPSDR: Orion2 (Protocol 1 v2.7)) 192.168.1.49 (54:10:EC:9B:2D:EA) on eth0

piHPSDR - Transmit

Close

- ● Mic In
- ☐ Compression (dB): 0
- TX Filter: ☐ Use RX filter 150 2850
- ☑ Local Microphone Input: plughw:4,0 Loopback
- Panadapter High: 0
- Panadapter Low: -80
- AM Carrier Level: 0.0
- ☐ FM TX Pre-emphasize before limiting
- ☐ CTCSS Enable 100.0
- ☑ Tune use drive
- Tune Percent: 10

Through the "modprobe" command, the virtual sound cards were created once, and this gets lost after a re-boot. One should create the "virtual audio cables" automatically when starting the machine. There are different ways to do this.

If the file `/etc/rc.local` exists, it is most easy to automatically execute the modprobe command on startup by inserting a line reading

```
modprobe snd-aloop enable=1,1 index=4,5 id=vac1,vac2 pcm_substreams=2,2
```

at the bottom of the file before the final line reading "`exit 0`". This can be done with a text editor, but since you need administrator privileges to to so, the text editor should be started from a terminal window, by one of the following two commands

```
      sudo vi /etc/rc.local
      sudo nano /etc/rc.local
```

depending on which text editor you are most familiar with. This worked on my RasPi4. Some Linux variants do not have the file `/etc/rc.local` but a file named `/etc/modules` and a directory names `/etc/modprobe.d` exists. In this case you have to edit the file /etc/modules and add a line reading

```
snd-aloop
```

and you have to create a new file with name loopback.conf in the directory /etc/modprobe.d which contains one line reading

```
options snd-aloop enable=1,1 index=4,5 id=vac1,vac2 pcm_substreams=2,2
```

Appendex A: Compiling and installing SoapySDR components

You will need "cmake" and " libusb1.0" to be installed on your system, so enter the commands

```
sudo apt install cmake
sudo apt-get install libusb-1.0-0-dev
```

If these packages are already installed on your system, issuing these commands does no harm

Step 1: Download, compile and install SoapySDR

```
cd $HOME
git clone https://github.com/pothosware/SoapySDR.git

cd SoapySDR
mkdir build
cd build
cmake ..
make -j 4
sudo make install
sudo ldconfig
```

Step 2: For LimeSDR download, compile and install LimeSuite

```
cd $HOME
git clone https://github.com/myriadrf/LimeSuite.git

cd LimeSuite
cd build
cmake ..
make -j 4

sudo make install
```

Install the udev rules:

```
cd $HOME/LimeSuite
cd udev-rules
sudo ./install.sh
```

Step 3: For PlutoSDR download, compile and install SoapyPlutoSDR

Install libad9361 and some more prerequisites

```
sudo apt install libad9361-dev
sudo apt install bison
sudo apt install flex
sudo apt install libxml2-dev
```

Download, compile and install libiio:

```
cd $HOME
git clone https://github.com/analogdevicesinc/libiio.git

cd libiio
mkdir build
cd build
cmake ..
make
sudo make install
sudo ldconfig
```

Download, compile and install SoapyPlutoSDR

```
cd $HOME
git clone https://github.com/pathosware/SoapyPlutoSDR

cd SoapyPlutoSDR
mkdir build
cd build
cmake ..
make -j 4
sudo make install
```

To install the udev rules read and follow the instructions at `https://wiki.analog.com/university/tools/pluto/drivers/linux` (that is, copy the file `53-adi-plutosdr-usb.rules` to `/etc/udev/rules.d`).

Step 4: For RTLSDR download, compile and install SoapyRTLSDR

Download and install librtlsdr

```
cd $HOME
git clone git://git.osmocom.org/rtl-sdr.git

cd rtl-sdr
mkdir build
cd build
cmake ..
make
sudo make install
sudo ldconfig
```

Download, compile and install SoapyRTLSDR

```
cd $HOME
git clone https://github.com/pothosware/SoapyRTLSDR

cd SoapyRTLSDR
mkdir build
cd build
cmake ..
make
sudo make install
```