

# Compile piHPSDR & Fldigi, WSJTX, FreeDV from the source-code (Linux) (Desktop PC / Laptop / RaspBerry Pi)

Christoph van Wüllen, DL1YCF  
Kaiserslautern, Germany

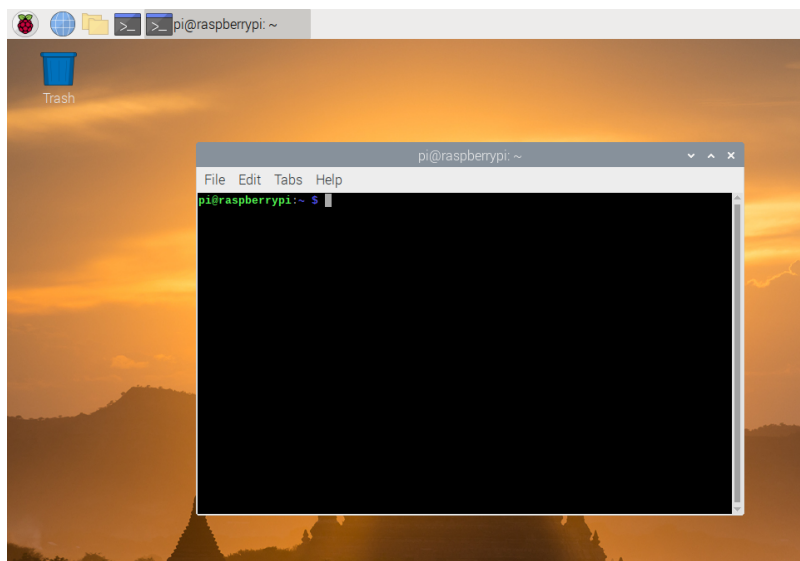
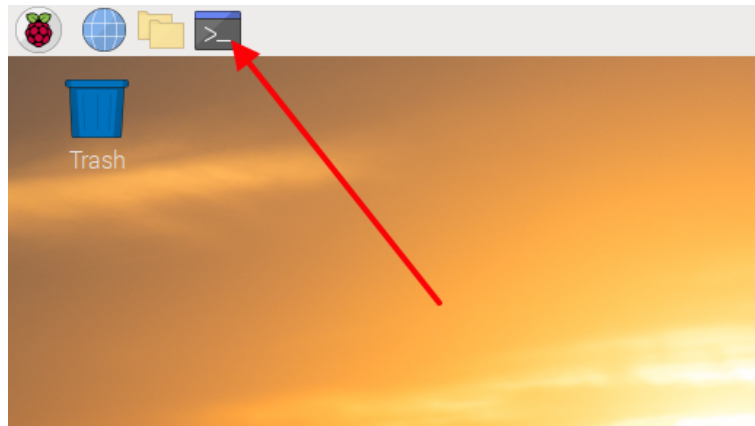
... and lots of others, including my "guinea pigs" have helped to bring this into shape.

Latest change to this document: December 26, 2021

## A) Hardware, Software, and Skills required to follow the instructions in this document.

- An obvious prerequisite is that you have a **computer running the Linux operating system** (OS) in order to compile and run piHPSDR there. This can be a Desktop or laptop PC, or a small single-board computer (SBC) such as the RaspberryPi (RaspPi). The single but essential difference between the Raspberry Pi on one side, and a Linux installation on a Desktop PC or laptop on the other hand, is that only the SBCs have general-purpose input/output (GPIO) connections that can be used for connecting push-buttons, rotary encoders, Morse keys etc. So it should be clear that all instructions concerning GPIO only apply to SBCs. Controllers or Morse Keys to be used with Desktop PCs must use MIDI.
- Since we need to install additional software components, the computer **needs an internet connection**, no matter whether this connection is realized by an ethernet cable or using a WLAN. At the very end, when piHPSDR is up and running, you most likely need an ethernet cable network connection to connect to the radio.
- It will be necessary that you have the Linux OS running on this computer. In **Appendix A** some instructions how to obtain and install Linux from the internet are given. Installing Linux is actually the most complicated part of the whole business here, but in most cases you do not need it: if you want to use piHPSDR on a Linux PC, then most likely you already have one, and if you buy or have bought a RaspPi, the vendor almost certainly also offers SD cards with a pre-installed Linux OS that you simply have to insert in the SD card slot. Take care to use an SD card which holds at least 8 GByte. For RaspPi users: look into Appendix A how to enable the I2C interface, since you will need it if you want to connect the "piHPSDR controller" to the RaspPi.

- The commands given in this document must be entered in a terminal window, so you must know how to open such a window. On a RaspPi, the terminal is behind this symbol in the top menu bar (see red arrow) but can also be opened from the Raspberry menu Accessoires ==> Terminal. A terminal window opens and looks like displayed in the figure below.



In this terminal window, you can now type in commands. Begin with typing in the command

`echo $HOME`

Throughout this manual, commands to be typed into a terminal window are set in blue colour with a monospaced font. You have to type the command either exactly as printed here. Fear not, there is very little you have to type in, because all the complicated stuff is done in so-called "script files" which come separately (you should get a file called `scripts.tar` together with this document).

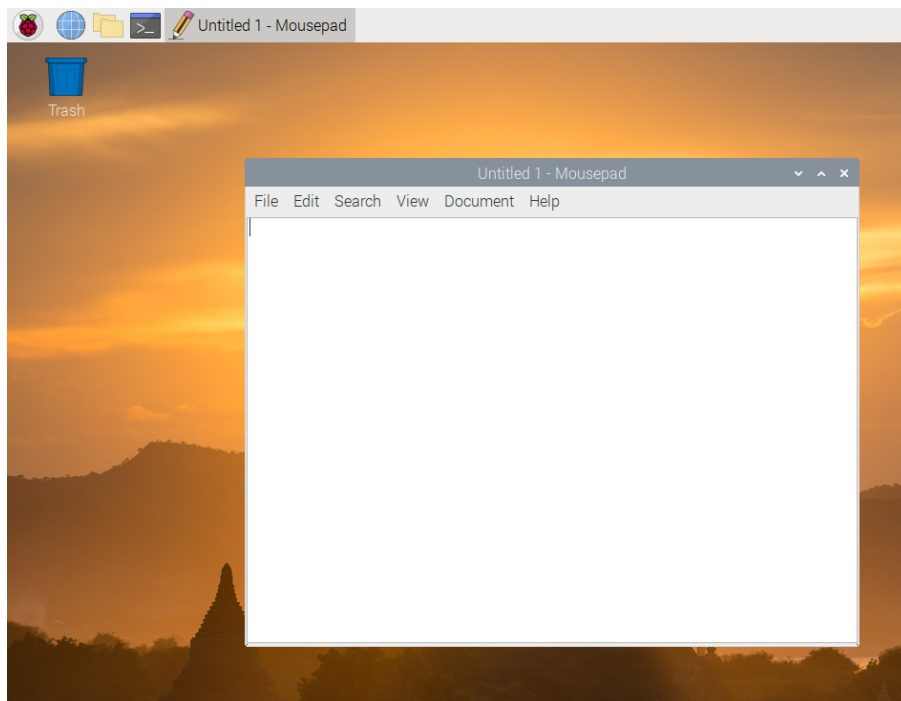
As a result of the command `echo $HOME`, the name of your home directory should be printed on the screen. This is `/home/pi` for the RaspPi and `/home/user` for Desktop/Laptop Linux computers, where "user" is replaced by your Linux user name there.

- You most likely have to install additional software components, and this requires administrator privileges. To check if you can execute a command with administrator privileges, type in the command

```
sudo ls -l
```

This should list all the files in your home directory. On the RaspPi, this usually works without further a-do. On other systems, you might get asked the administrator password before the command is executed (if you do not know this password, you cannot manage the system and must ask the person who installed the Linux). On some Desktop/Laptop Linux systems, the security level is even higher and you must be explicitly entitled to use the "sudo" command. Ask a local Linux guru how to achieve this (most likely, your user name must be included in the "sudoers" file, but this may depend on the system and/or the security level imposed there).

- Although no longer required since the "scripts" do everything for you, you should be able to create and modify text files. I usually do this from within a terminal window using the "vi" command, but this is really old-school since I am working with Linux/Unix systems for more than 30 years. I guess if you are reading these instructions this means that you have *not* been working with Unix/Linux since the 1980s, and then the learning curve for mastering the vi program is rather steep. So these instructions are made such that you can equally well use a text editor with a graphical user interface (GUI). On the RaspPi, a very simple such text editor ("Mousepad") can be found behind the Raspberry: "Accessories->Text Editor". This opens a new window such that the screen looks like this:



In the white area, you can type in text. Type in the text

```
Now is the time  
for all brave men  
to come to the party.
```

Throughout this document, contents of text files to be created, or parts of text files that need to be modified, are shown in green color and a mono-spaced font. If you type in the text, do not forget the Enter key after typing in the last line (such that the cursor jumps to the beginning of the fourth line). If the text has been entered, you can go to the menu "File->Save As" of the text editor, and enter a name for the file (take the name `MyCuteTextFile`) into the line at the top (behind "Name:"). After entering the name, the "Save" button at the bottom right of the "Save As" window becomes active and must be clicked. Close the text editor window and go back to the terminal window by clicking somewhere in its area and type in the command

```
cat MyCuteTextFile
```

This should produce as output the text shown in green above. As the last step, we must be able to modify existing text files. To test if we can do this, open the text editor again and choose the file named `TextFile` through the "File->Open" menu by double-clicking the file. Now you can use the mouse but also the arrow keys on the keyboard to navigate, say, to a position following the word "men" and add the words "`and women`" at the end of that line. Save the modified file through the "File->Save" menu and close the text editor window. Then activate the terminal window again and type in the command

```
cat MyCuteTextFile
```

again. Now the modified text should be printed on the screen.

**If you do not succeed in performing these tasks so far, it makes no sense to continue reading. It is strongly recommended to go to the next local radio amateur meeting and ask for help. What we have done so far is just very basic Linux, if you have difficulties you cannot overcome already at this stage you won't be able to proceed further.**

## **B) Obtain useful scripts to help with installation/compilation**

In the next steps we will obtain and install software packages that may or may not be already present on your system and are needed to compile piHPSDR (and possibly other programs). To facilitate this, I have prepared a file `scripts.tar` which contains a bunch of so-called "shell scripts" that perform the tasks required. (Note to experts: bundling the scripts in a tar file instead of making them available as separate files circumvents problems with Windows-type end-of-line markers, missing execute permissions, etc.) You should obtain this file along with this document. The file `scripts.tar` must be placed into the home directory of your "pi" account on the RaspPi. A straightforward way to do so is to copy it onto a USB stick which is then inserted into the RPi. So copy the file `scripts.tar` to the USB stick on your "main" computer (no matter if it runs Windows, Linux, or MacOS), and then insert the USB stick into the

RaspPi. A window pops up where you can choose to open the USB stick in the file manager (do so!). Then, with a mouse drag+drop the file `scripts.tar` into the "Home Folder". To check whether this has succeeded, open a terminal window and type in

```
ls -l
```

This should produce a list of files/directories, with `scripts.tar` among them. If this is the case, proceed by typing in the command

```
tar xvf scripts.tar
```

which produces the list of files extracted from `scripts.tar` on the screen, which all end in ".sh". These scripts greatly facilitate what follows, since you need not type in dozens of commands (this is, of course, error-prone) but simply execute a script.

## C) Some post-install modifications of the computer setup

Most of this is only necessary in special situations. If you make any changes described in this section, you should re-boot the system before proceeding with the next section.

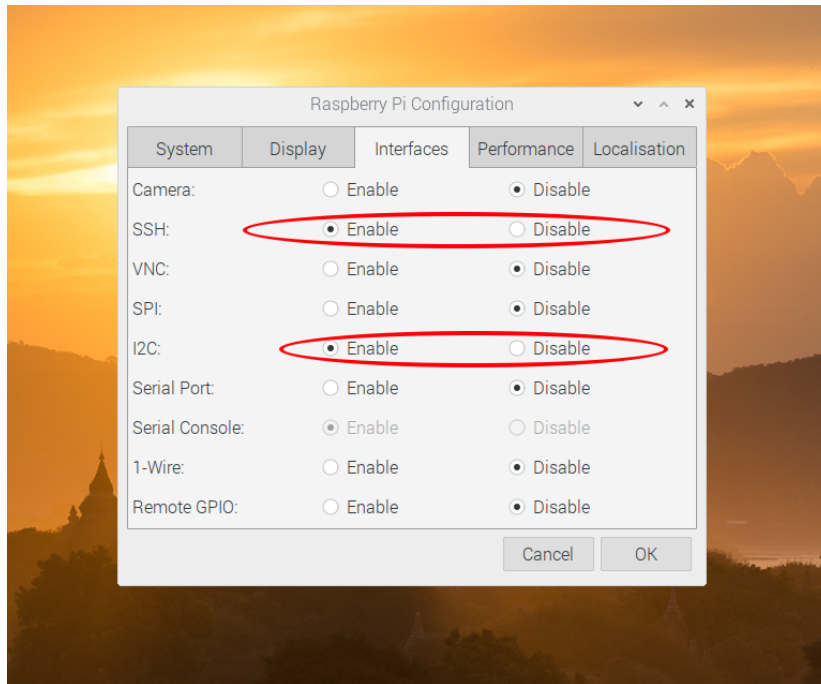
### C.1) RaspPi only: configure interfaces

*Note: Enabling SSH is only necessary if you want to log in into your RaspPi from another computer, and enabling I2C is only relevant if you use a new (Version 2) piHPSDR controller. If neither of this applies, you may well skip this.*

Use the mouse, and chose from the "Raspberry" menu

Raspberry ==> Preferences ==> RaspberryPi Configuration

You can configure dozens of things there (e.g. the keyboard), but for this I refer to the internet. What is important here is the "Interfaces" tab, and if you click there, you get the following window, in which you can enable/disable features by clicking the radio buttons. I suggest to disable all features except (possibly) SSH and I2C. SSH is needed if you want to "log in" on your RaspPi from outside (e.g. from another PC), while I2C is needed if you want to connect the new versions of the "piHPSDR controllers".



## C.2) Keyboard settings

*Note. This probably only applies to users in Germany who have a German keyboard but nevertheless use the English language on the RaspPi. The problem may also occur in other non-english-speaking countries.*

If the keyboard layout does not fit, goto to "Raspberry" ==> "Preferences" ==> "Keyboard and Mouse" and select the "Keyboard" tab. In this tab click "Keyboard Layout..." and select the "German" (or whatever layout you have" layout. The change becomes effective immediately.

## C.3) Setting a fixed IP address

*Note. This is only necessary if you plan to connect the computer and the radio directly by an ethernet cable, without a router in-between.*

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 on my RaspPi and 192.168.1.99/24 on 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). This is simply performed by the command

```
./ip.sh
```

This will set 192.168.1.50 as fixed IP address on your RaspPi after the next reboot. You can choose other addresses by editing the file ip.sh, it should be clear how to do. Note that even when using a fixed IP address, the RaspPi will use DHCP and obtain an address from the router if connected. This means, even if you use "direct

connection" from computer to radio, you can, from time to time, connect the RaspPi with the router and so some work that requires internet connection.

#### C.4) Fixing some GPIO problems (RaspPi only)

*Note. This skip **must** be skipped if you run piHPSDR on a desktop/laptop, and **can** be skipped if you do not intend to use the RaspPi GPIO lines (that is, you do not intend to connect a piHPSDR controller or CW keys or a PTT switch from the microphone to the GPIO).*

When the RaspPi4 replaced the RaspPi3, a new Broadcom I/O chip has been introduced and some libraries still have problems to correctly configure the GPIO lines. Therefore a script is provided that lets the RaspPi on each system start configure the GPIO such that GPIO lines 4–27 are programmed as input lines with internal pull-up resistor. To do so, simply use the command

```
./gpio.sh
```

which puts appropriate instructions into the file `/boot/config.txt` but remember do do so **only** on a Raspberry Pi.

#### C.5) Instrumenting the computer for audio connection between piHPSDR and digimode programs (WSJTX, Fldigi, FreeDV)

*Note. This can all be skipped if you do not plan to run piHPSDR along with a digimode program such as wsjtx or fldigi on the same computer.*

The command is simply

```
./pulseaudio.sh
```

This creates a file `$HOME/.config/pulse/default.pa` which configures the pulseaudio sound system (and restarts pulseaudio). This way, the change becomes effective immediately and upon each system boot. Two additional so-called "null-sink" audio output devices with name "SDR-RX" and "SDR-TX" are created which can be used to transport audio data from one application to the other.

Furthermore, the default input and output devices for pulseaudio are set to "SDR-TX" (default output device) and "SDR-RX.monitor" (default input device). This is necessary since both Fldigi and FreeDV can use pulseaudio, but they do not allow to select a device so they automatically use the default input and output device.

**Hint.** It has been reported that pulseaudio may do odd things if piHPSDR is running for a long time. The cure was to locate the following line in the file named `/etc/pulse/default.pa`

and locate the line

```
load-module module-suspend-on-idle
```

and either delete it or deactivate it by inserting a number sign "#" in the first column.



## D) Compiling the programs

### D.1) Download software

The software is downloaded by the three commands

```
./packages.sh
./hamradio.sh
./desktop.sh
```

The first two commands essentially fetch all required software from the internet, so they make take some time especially if your internet connection is not very fast. The third commands creates icons for piHPSDR, Fldigi, WSJTX and FreeDV on the Desktop, such that you can start these programs (after they have been compiled) just by double-clicking the icon. I just tested this procedure on my RaspPi4, executing `packages.sh` took 6 minutes, `hamradio.sh` took 2 minutes, and `desktop.sh` less than a second. The times depend both on the speed of your internet connection and on the speed of your SD card.

**ATTENTION:** *the command "hamradio.sh" deletes all directories before it loads them from the internet. So if you have made any modifications (e.g. of Makefiles) and run the "hamradio.sh" script again all your modifications are lost. Normally, use this command only once after setting up the system.*

### D.2) Configure piHPSDR

Note: piHPSDR comes with some options that one can activate or deactivate at compile time. Most users will **not need to anything here and my skip D.2).**

There are only three cases where you have to do something, namely

- a) you run piHPSDR on a desktop/laptop computer (no GPIO available)
- b) you want to include the SoapySDR module for running piHPSDR with an AdalmPLUTO radio
- c) you want to run piHPSDR with RedPitaya-based radios where the SDR application has to be started via a web interface, and you want piHPSDR to do this for you.

To configure piHPSDR, you have to modify the file `Makefile` inside the `piHPSDR` directory in your home directory, e.g. using the "Mousepad" editor as described in section A). You have to locate certain lines in the Makefile, change them, and save the Makefile. In the following I will print the line as it stands in the Makefile in blue, and how it must look like in green. The recipe is such, that an option that is active can be deactivated by inserting a number sign in the first column of that line, and an option that is inactive can be activated by deleting the number sign in the first column.



**Case a):** your Computer does not have GPIO input/output lines. Locate the blue line below and replace it by the green line following

```
GPIO_INCLUDE=GPIO
#GPIO_INCLUDE=GPIO
```

**Case b):** you want to use SoapySDR radios. Currently, only the AdalmPLUTO is supported. Locate the blue line below and replace it by the green line following

```
#SOAPYSDR_INCLUDE=SOAPYSDR
SOAPYSDR_INCLUDE=SOAPYSDR
```

**Case c):** you want to use RedPitay based radios. Locate the blue line below and replace it by the green line following

```
#STEMLAB_DISCOVERY=STEMLAB_DISCOVERY_NOVAHI
STEMLAB_DISCOVERY=STEMLAB_DISCOVERY_NOVAHI
```

### D.3) Compile all the programs

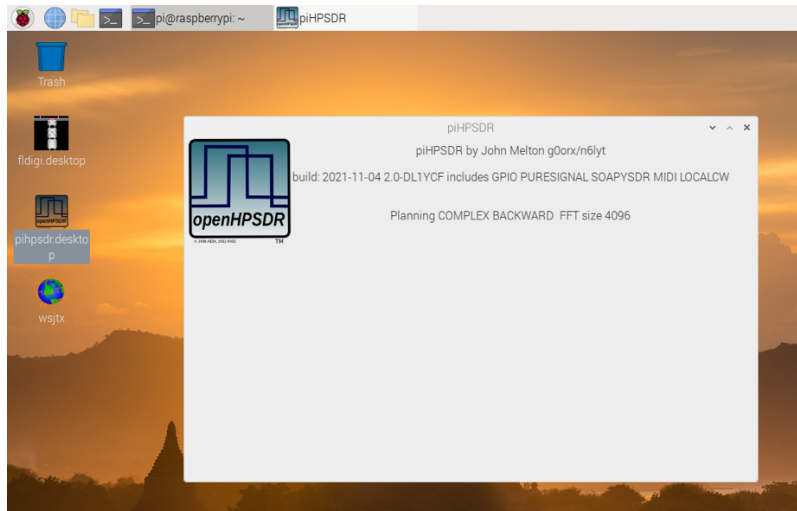
This is all done by the command

```
./compile.sh
```

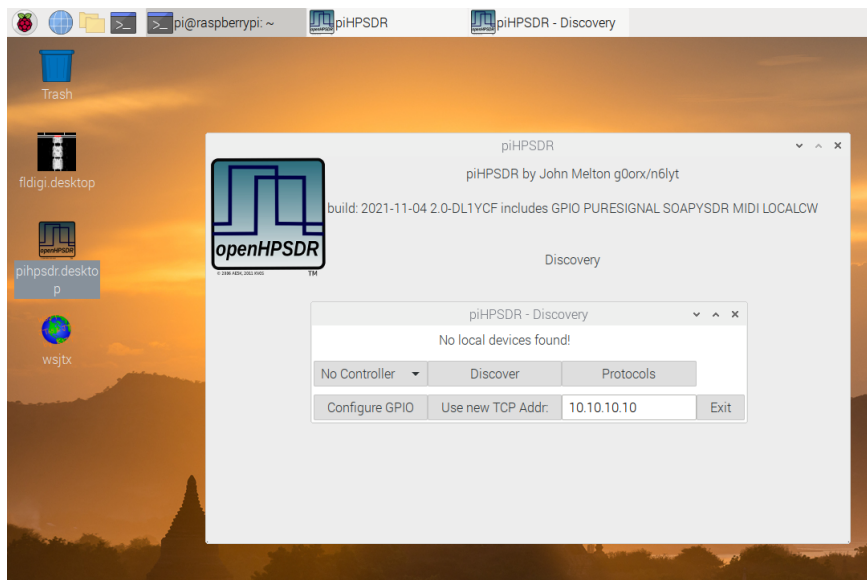
This takes about 50 min on my RaspPi4, a large part thereof is required to compile Fldigi which can only be compiled using a single CPU core due to excessive memory demand for the compilation of one of the source code files. Note everything is put into this script. That is, it not only compiles piHPSDR, Fldigi, WSJTX and the FreeDV program, but also all necessary support libraries including those for SoapySDR, even if they are not needed or wanted. This is to keep things simple.

## E) Initial run of piHPSDR

To test the compilation, we make an initial run of piHPSDR without any radios connected to the computer. To do so, just double-click the piHPSDR icon on the Desktop. If a window pops up asking you how the program should be executed click the first tab "Execute" (see section F.2). Then, the piHPSDR window should open and the screen should look like this

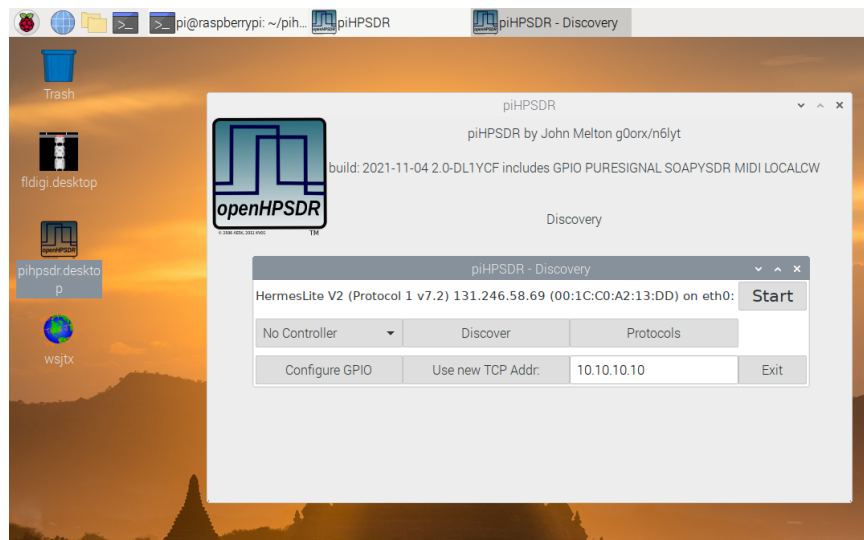


Because it is the first time you started the program, the WDSP library determines (once and for all) the optimum way to do the fast-Fourier-transforms (this will take few minutes, there is a progress report on the screen). After this time, the piHPSDR window looked like this:

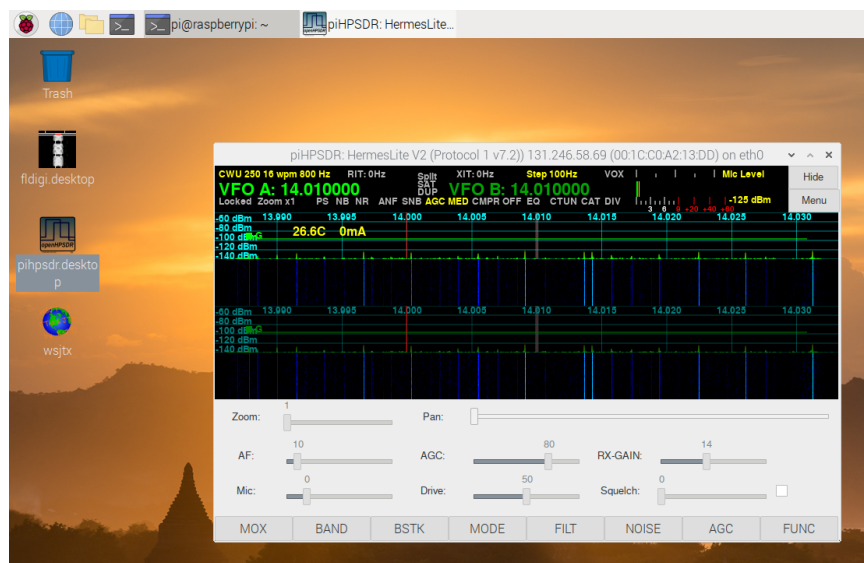


Since we have no radios connected (and therefore no devices have been found), clicking the "Exit" button is the only thing we can do at the moment. If we had connected radios (for example SOAPY devices such as the Adalm-Pluto via USB, or an ANAN radio via an ethernet cable) these devices should be "discovered" and the radio can

be started via a "Start" button. In the next picture, this situation is shown, an HermesLite-II has been connected via Ethernet:

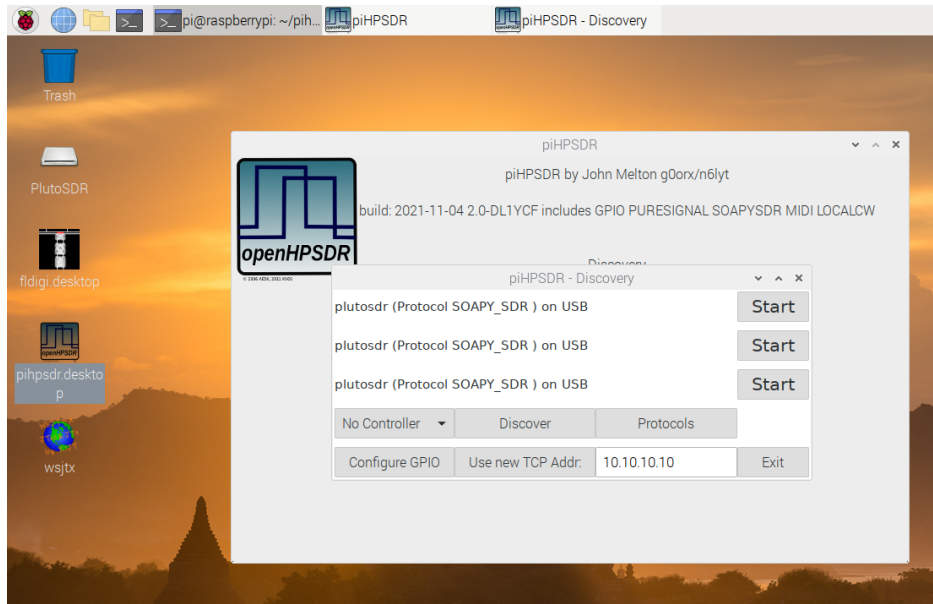


It takes some time to arrive here, because piHPSPDR tries to "discover" HPSDR protocol1, HPSDR protocol2, and SoapySDR devices. Using the "Protocols" menu by clicking the tab with that name, one can disable those protocols for which no hardware is present. Clicking the "Start" button then leads to the following

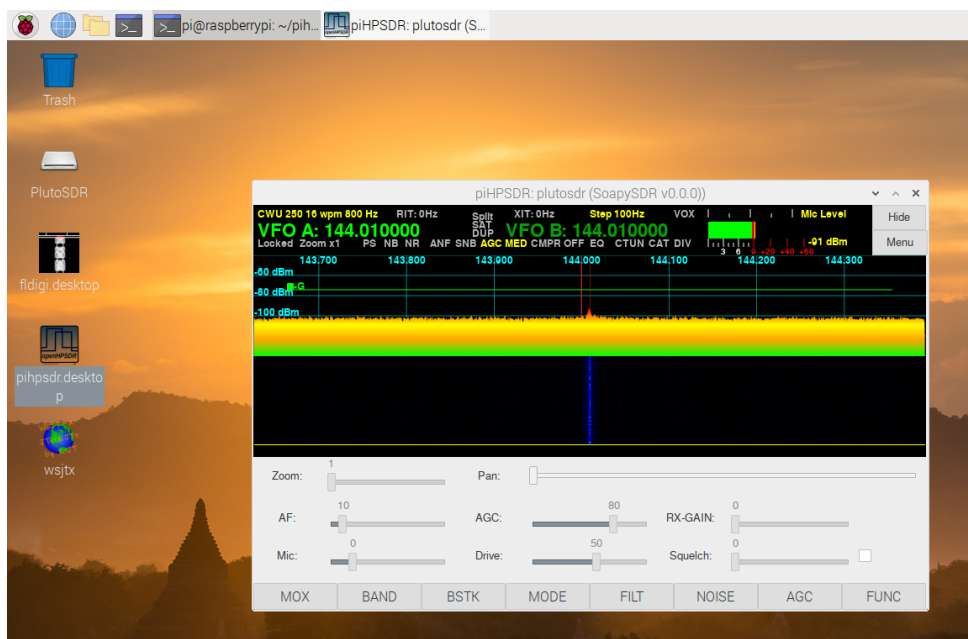


As you can see, the radio starts by default with two receivers, and both receivers have the panadapter and the waterfall on display. This can be configured of course within piHPSPDR and is stored in a local file so the next time you start piHPSPDR, these settings are restored.

To complete the test, the piHPSPDR program was left (through the Menu==>Exit button in the top right corner), the Hermeslite-II disconnected and instead, an AdalmPluto was connected to the RaspPI via an USB cable. Starting piHPSPDR again (by double-clicking its icon) then leads to the following



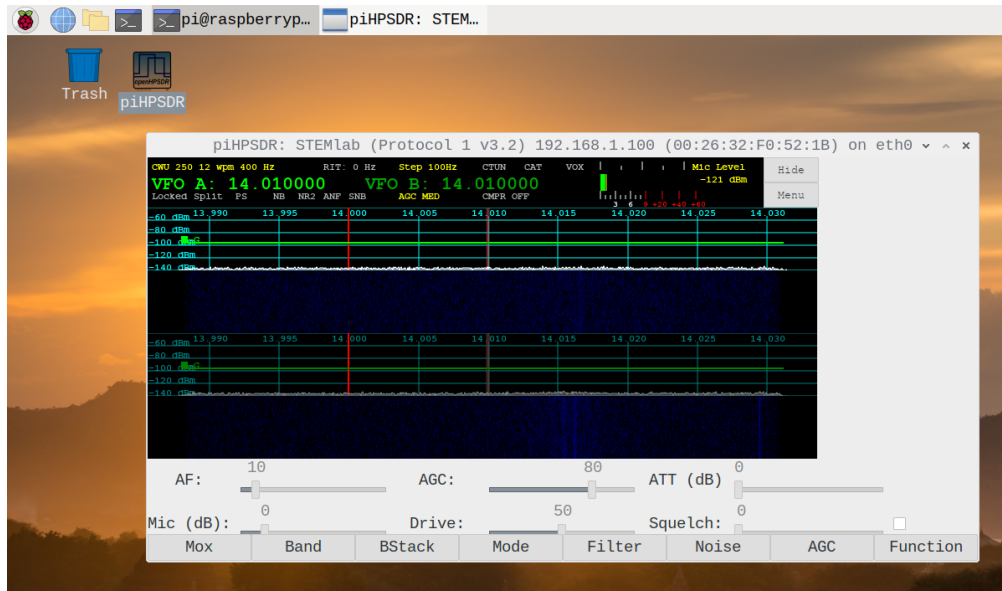
and I cannot say why the same device has been "discovered" three times. This was reported by the Soapy library. Anyway, clicking the topmost "Start" button then starts the Pluto radio (I have modified mine such that it can do 144 Mhz)



## F) Trouble-shooting some issues that occasionally arise

### F.1) Too large font sizes (only RaspPi):

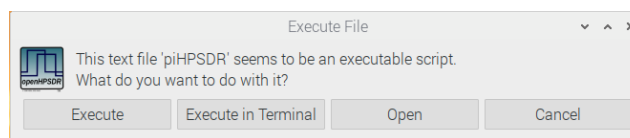
Some RaspPi users have reported that the radio window is messed up and looks like this:



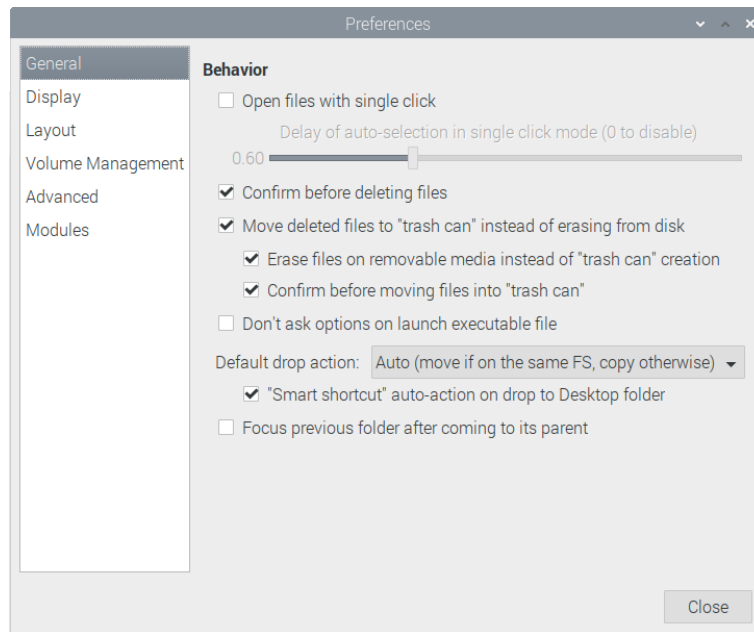
This happens especially when using a large monitor. The reason is, that the system may automatically choose a large font when using a large monitor, which is not reasonable for piHPSPDR since it is using a fixed-size window. This is easily fixed from the Raspberry -> Preferences -> Appearance Settings 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 piHPSPDR window looks OK.

### F.2) Desktop icons not working properly

When you double-click the one of the icons on your desktop, then probably the following dialogue pops up:



This can be suppressed. Simply invoke the file manager (the icon in the top bar to the right of the browser "earth globe" icon, navigate to the "Desktop" folder in your home directory and single-click "piHPSPDR". Then go to the menu Edit --> Preferences which looks like this:



Simply check the box at the beginning of the line "Don't ask options...", close the menu and close the file manager. That's it, you have to do this only once. In the (unlikely) case that you have no file manager icon in the top menu bar, you can open a terminal window and enter the command

`pcmanfm`

to start the file manager.

## **G) Running piHPSDR along with digimode programs (WSJTX, Fldigi, FreeDV) on the same computer**

*Note: If your computer has a very small screen (say, less than 1024\*768 pixels), then it makes not much sense to run piHPSDR along with anything, since the screen space is just too small. This is the case if you have the "piHPSDR controller" or a similar device, where the RaspPi is put into a small box together with switches and knobs and a 7-inch touch screen.*

*In such a case, you may use the VNC software (search the internet). VNC is built into the RaspPi OS but you need a client for your main (desktop) computer. Then you can add a second (virtual) screen to your RaspPi and display its contents on your main computer. If your controller contains a recent RaspPi, changes are also good that you can connect a second (external HDMI) monitor to it. If you want to do RTTY for example, you also need to connect a keyboard to the RaspPi,*

To run piHPSDR with a digimode program, we need to "connect" piHPSDR and the digimode program in two ways. The first "connection" is rig control, that is, the digimode program can change piHPSDR's VFO frequency and the mode, induce RX/TX and TX/RX transitions in piHPSDR, and so forth. Then we need "audio transport", that is, RX audio samples (that would normally end up at your headphone) must go to the digimode program, and audio samples created by the digimode application must go to piHPSDR and treated within piHPSDR as if they came from a microphone. All

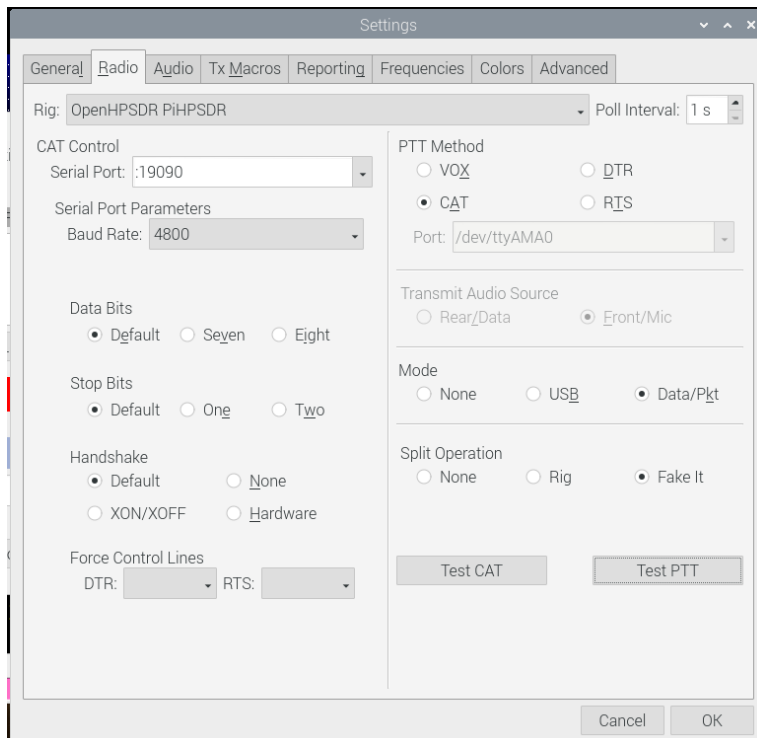
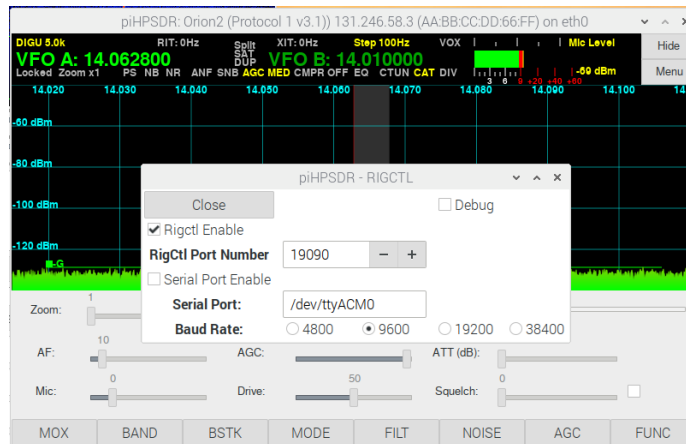


necessary ingredients are already there, so we just give screen shots how to adjust things.

Note: The FreeDV program actually needs **four** audio connections. Two audio connections are used for transporting audio samples between piHPSPDR and FreeDV in both directions, and this is the same as for the other digimode programs. In addition, using FreeDV one also needs a "true" head-phone and microphone. For testing and producing the screen shots, I connected a USB sound adapter named "iMic USB Audio" to the RaspPi, and connected a headphone and a microphone to that adapter.

### G.1) Rig Control

In piHPSPDR, click the RIGCTL tab in the main menu and check the "Rigctl Enable" box: Make sure that the "port number" is 19090 since we need this number in the other programs.

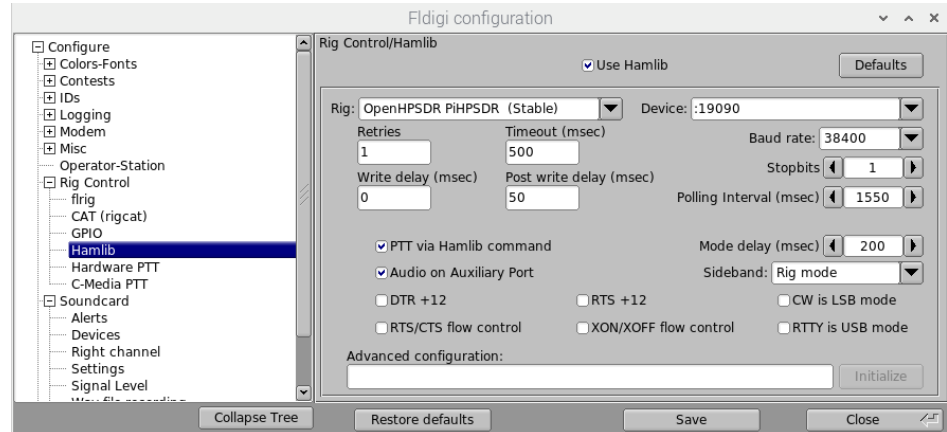


In WSJTX, go to "File ==> Preferences" and click the "Radio" tab. Choose "OpenHPSDR PiHPSPDR" as the rig, enter ":19090" as the serial port (note the colon at the beginning of the string) and enable "CAT" for the PTT method. Choose "Data/Pkt" for the mode (this means that the DIGU mode is chosen in piHPSPDR). If you want to TX at audio frequencies below 150 or above 2850 Hz, you also have to choose "Fake It" in the "Split Operation" field.

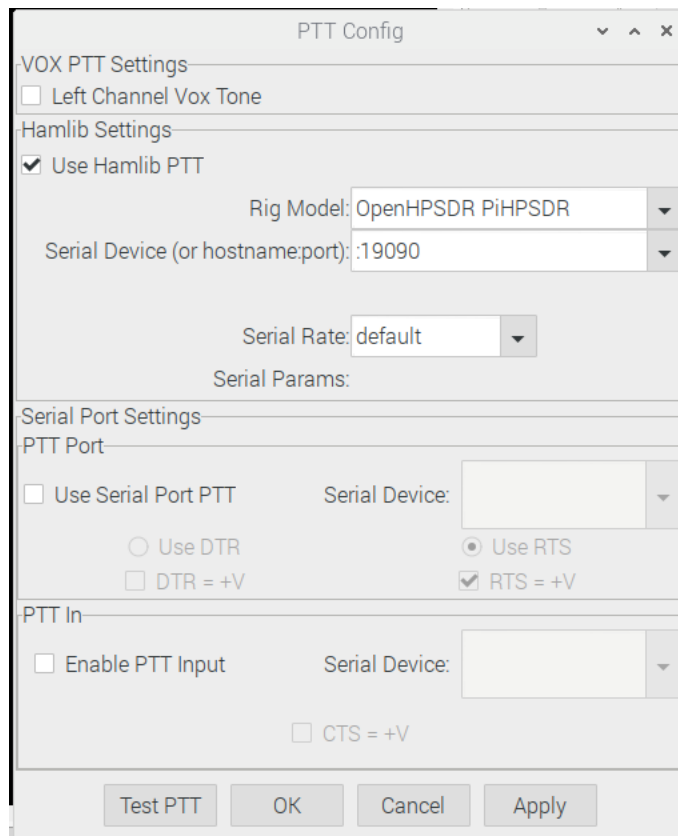
Note the other fields (Serial port parameters, Data Bits, Stop Bits, Handshake etc. have no meaning when using TCP connection and can be left "as is".



For FLdigi, go to "Configuration ==> Config menu" and expand the collapsed list such that it shows the "Rig Control ==> Hamlib" screen: Again, choose the Rig and the



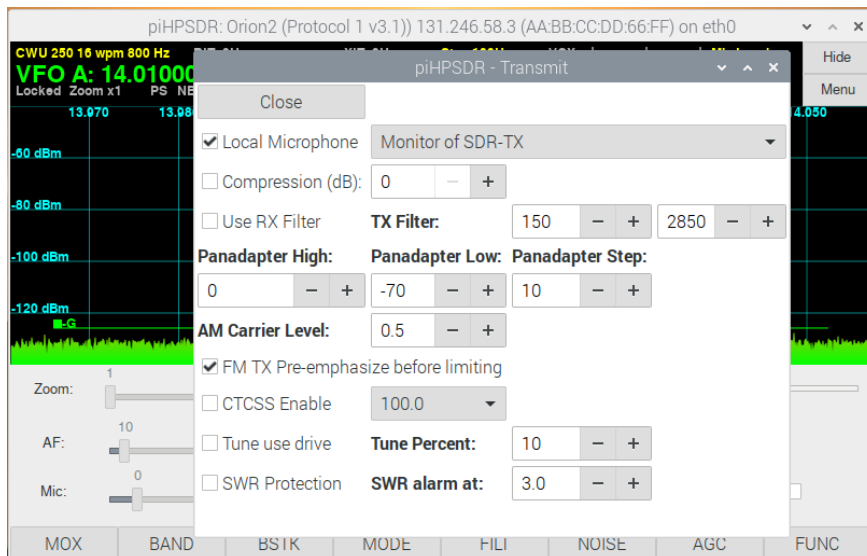
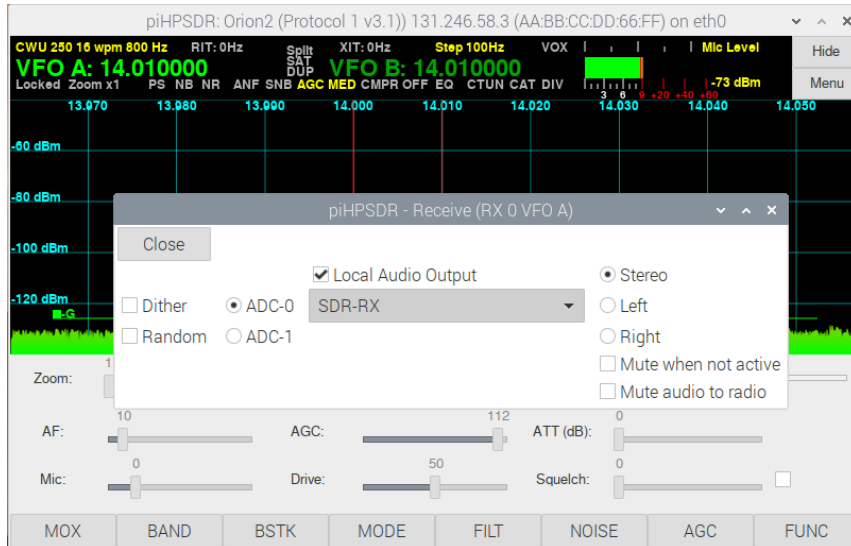
Device (as in wsjtx and as shown in the figure). Then you must check the box at the top "Use Hamlib" and then can hit the "Initialize" button at the bottom right. Do not forget to "Save" the configuration then you can "Close" the window.



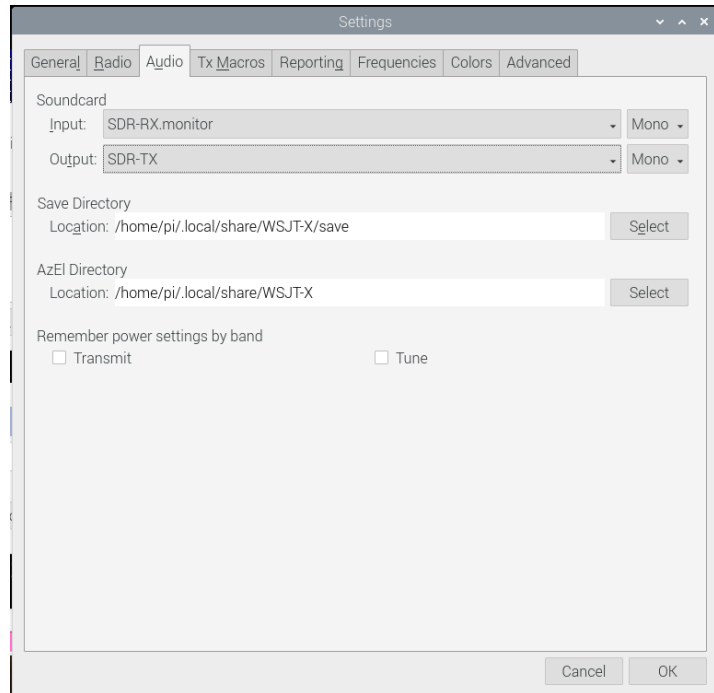
In the FreeDV program, go to "Tools ==> PTT Config". Check the box specifying that Hamlib is used, and choose the Rig model (OpenHPSPDR PiHPSPDR) and the TCP port (:19090, don't forget the leading colon!). With the button "Test PTT" you can verify that FreeDV can induce a RX/TX transition in piHPSPDR.

## G.2) Audio Transport

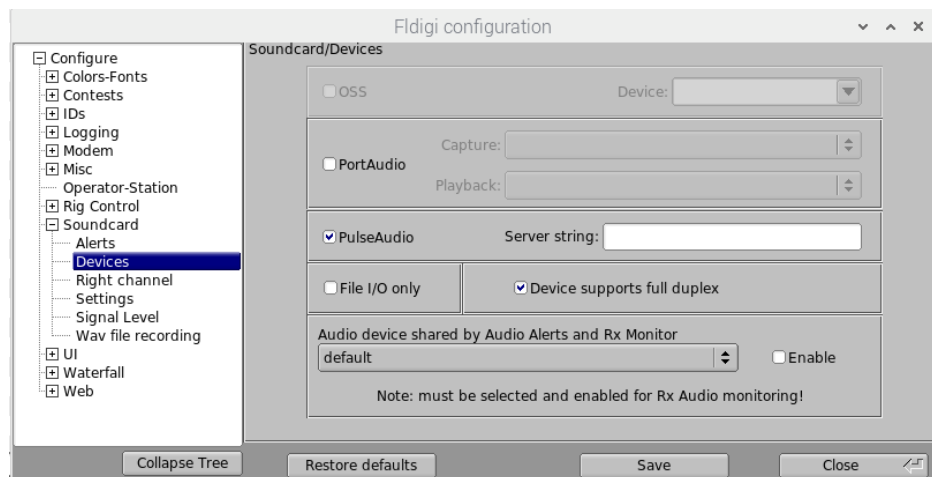
piHPSDR must be instrumented in its main RX and TX menus to use the so-called "null-sink" devices SDR-RX and SDR-TX that were created. Note "SDR-RX" is used in the RX menu, and "SDR-TX" is used in the TX menu. Since we need a sound input device in the TX menu, we must use the "monitor" device associated with SDR-TX.



For wsjtx everything is in the "Audio" tab that we can reach through the File ==> Preferences menu. Here we use "SDR-TX" as the output device since here data is sent to piHPSDR upon TX, while we use the monitor device associated with SDR-RX to capture the RX audio:

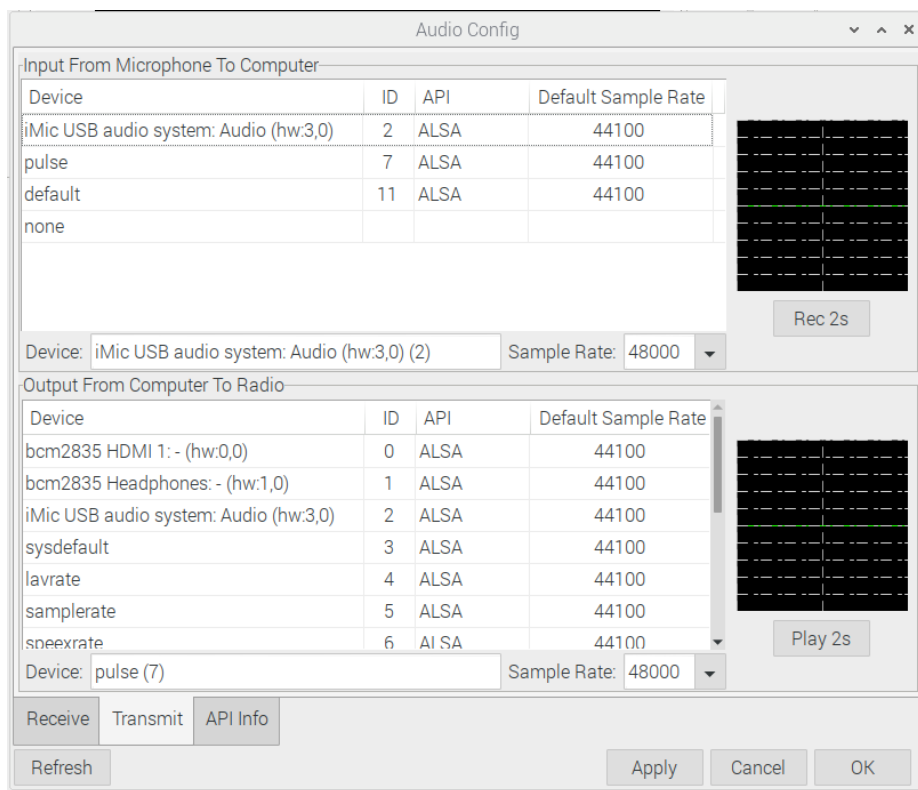
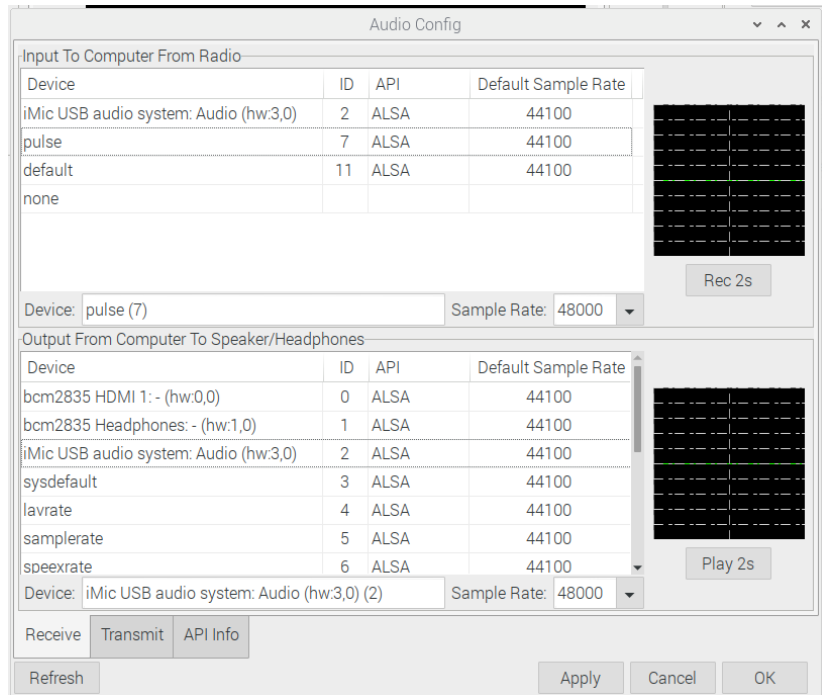


In fldigi, very little is to do. Going to the config menu we have to navigate to the Soundcard ==> Devices screen, check the PulseAudio box and leave the server string empty. Then Save & Close.



We can only use the default pulseaudio devices, but our setup in `$HOME/.config/pulse/default.pa` has specified SDR-RX.monitor as the default input and SDR-TX as the default output device, so this is compatible with the choice in piHPSDR (see above).

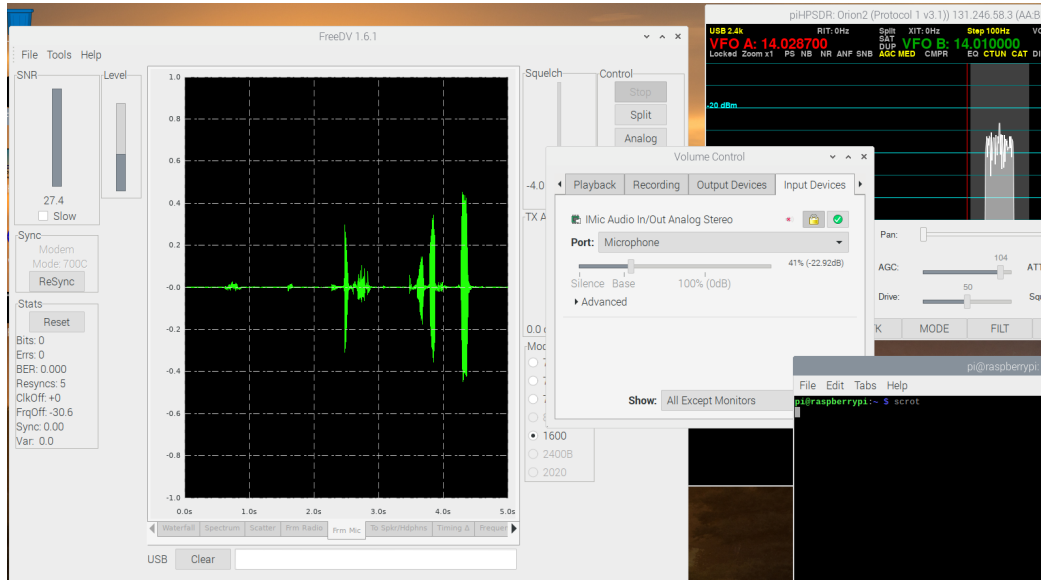
For FreeDV, you have to go to Tools ==> Audio Config. Four audio devices have to be configured, and be sure to use a 48k sample rate for each of them. **That is, your USB headset or USB sound adapter must support the 48000 Hz sample rate!** In the "Receive" tab, choose "pulse" for the "Computer from Radio" section, and the name of your USB headset (sound card) ("iMic USB audio system" in my case) for the "Computer to Headphone" section:



In the "Transmit" tab, choose "pulse" in the "Computer to Radio" section, and the name of your USB headset or sound card in the "Microphone to Computer" section:

Note that FreeDV is very picky about the microphone level. Since I found no adjustment of the microphone audio level in FreeDV, you have to use the pulseaudio volume control program "pavucontrol" (enter this as a command in a terminal window) to adjust the level. Here is an example, in my case I had to go to about -20 dB! You see the piHPSR window at the top right, the FreeDV main window at the top left, and the pavucontrol window in the center. During TX (you go TX by pressing the

PTT tab in the top right of the FreeDV window) you see your microphone signal in the FreeDV spectrum window. Note that the TX signal (to be seen in the piHPSDR panadapter window) always looks the same, no matter if you whistle into the microphone or not. It is a "brick wall" spectrum with the characteristic width of the codec in use (Mode=1600 in the case shown).



## Appendix A: Installing Linux

### Step 1: obtain OS image

**RaspPi:** An operating system image can be found at the RaspberryPi official web site <https://www.raspberrypi.com/software/operating-systems/>. It shows

#### Raspberry Pi OS

Compatible with:  
[All Raspberry Pi models](#)



#### Raspberry Pi OS with desktop and recommended software

Release date: May 7th 2021  
Kernel version: 5.10  
Size: 2,867MB  
[Show SHA256 file integrity hash:](#)  
[Release notes](#)

[Download](#)

[Download torrent](#)

#### Raspberry Pi OS with desktop

Release date: May 7th 2021  
Kernel version: 5.10  
Size: 1,180MB  
[Show SHA256 file integrity hash:](#)  
[Release notes](#)

[Download](#)

[Download torrent](#)

#### Raspberry Pi OS Lite

Release date: May 7th 2021  
Kernel version: 5.10  
Size: 444MB  
[Show SHA256 file integrity hash:](#)  
[Release notes](#)

[Download](#)

[Download torrent](#)

and clicking the "Download" button in the middle ("Raspberry Pi OS with desktop, indicated by the red arrow) one obtains a "zipped" OS image file. The last time I tried this (November 2021), the zipped file had the name 2021-05-07-raspbian-buster-armhf.zip (1.25 GByte) and un-zipping it produced a file with 4 Gbyte and file name 2021-05-07-raspbian-buster-armhf.img.

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

**Desktop/laptop Linux system:** Here it depends on which Linux distribution is being used. The instructions given here have been tested with the "Debian GNU Linux" distribution. To this end, a "small" CD-image file (about 350 MByte) with file name debian-11.0-amd64-netinst.iso has been obtained from the internet page <https://www.debian.org/CD/netinst/> (netinst CD image for the amd64 architecture) and this file has to be "burnt" onto a CD or DVD, or onto a USB stick if the PC/laptop supports booting from a USB stick.

## Step 2: Install operating system

**RaspberryPi:** The OS image file already contains the complete OS. It has to be written (or "burned") onto an micro-SD card. In Nov. 2021 I went to an electronics shop and the smallest SD card I could obtain had a capacity of 32 GByte, this is more than enough. If you still have some older cards, use a card with at least 8 Gbyte, or else your filesystem will most likely overflow. How to do burn the OS image to the SD card 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, e.g. on the "getting started" page for RaspPi

<https://www.raspberrypi.com/documentation/computers/getting-started>

Note that "burning" can take several minutes, since the I/O speed is about 10 MB/sec on most cards (this means you need about 7 minutes to write the OS image to the micro-SD card). If you have "burnt" an SD card, it then has to be inserted in the SD-card slot of the RaspPi.

**Desktop/laptop computer:** Normally one writes the boot image to a USB stick in the same way one "burns" an SD card, but it is also possibly to use a CD/DVD if you are "old style". Then simply boot your desktop PC/laptop off the USB stick, then you get a Debian installation screen from which you choose "Graphical Install". Then proceed further choosing your localization etc. Because only a small boot image has been downloaded, additional components are obtained from the internet during installation, so you clearly need internet connection for the installation.

When the "software selection" screen appears, check the boxes "desktop environment", "ssh server" and "standard system tools". For the look-and-feel of the desktop environment, there are several choices, I have checked "LXDE" because this is also the standard desktop on the RaspPi. Since more than 1000 software packages are going to be installed, the process may take some time, mainly depending on the speed of your internet connection.

During the installation, you have to specify the password for the administrator ("root") account as well as choosing the name and the password of at least one regular user.

## Step 3: 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 to the most recent version. When this is complete, the system should be restarted.

**Desktop/laptop:** The system automatically boots after the installation. Because this is a standard Linux system, it is much more restrictive concerning the allowance for users to use the sudo command to perform administrator tasks. Normally the file /etc/sudoers has to be edit to grant the "normal user" such privileges. One possibility is to add the line

```
user    ALL=(ALL:ALL) ALL
```



to the file `/etc/sudoers` where the name of the "normal user" has to be used instead of "user". This gives this user full administrator privileges so the system is potentially insecure.

#### Step 4: Upgrade operating system

The "image file" obtained in step 1 is updated on the internet in regular intervals, but normally you would like to run the most recent version of the operating system. To do so, open a terminal window and type in the two commands

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

On the RaspPi this should not be necessary (since this task is normally automatically performed upon first-time boot from the SD card) but it also can do no harm.

You will get a bunch of output after either of the two commands, and for this step to work you need internet connection. If you connect the RaspPi and your internet router by a standard Ethernet cable, you should automatically get internet connection (here I assume that your router offers DHCP service, but this you also need for all other computers). Remember that after an OS upgrade, the system should always be rebooted to make the upgrade effective, so after the second command has completed, re-boot the system.

## Appendix B: Standard installation in a nutshell

In the guide, we have considered many special cases which are not relevant to the largest part of the users. Therefore here it comes in a nutshell:

a) Setup your RaspPi, preferably with a "virgin" operating system, as described in Appendix A. Transfer the file `scripts.tar` (e.g. via an USB stick) to the Home directory (`/home/pi`) on your RaspPi.

b) Open a terminal window on the RaspPi and type in the following commands. After each command, there may be lots of output lines and some commands take long, but here they are one after the other:

<code>./ip.sh</code>	<b>&lt;=== skip this using a router between Computer and Radio</b>
<code>./gpio.sh</code>	<b>&lt;=== skip this if <i>not</i> using a RaspPi</b>
<code>./pulseaudio.sh</code>	
<code>./packages.sh</code>	
<code>./hamradio.sh</code>	
<code>./desktop.sh</code>	
<code>./compile.sh</code>	

This should produce a "run-able" piHPDSR, Fldigi, WSJTX, and FreeDV program that can be started by double-clicking the corresponding icon on the Desktop. The desktop icons should appear almost immediately after you executed the "desktop.sh" script. Compilation takes most of the time (about 50 min on a RaspPi4). For running piHPDSR with WSJTX/Fldigi/FreeDV, see sec. G) how to set up the programs accordingly.

## Appendix C: install scripts

You should never need this section! Obtain, if possible, the scripts as a tar file from the same source you obtained *this* file. In the unlikely case you have *this* file but cannot locate the file `scripts.tar`, the scripts are printed here (in fine print). It takes some basis knowledge to get this files onto a Linux system with proper execute permissions, to this appendix is "experts only". The scripts are printed here in the order they are mentioned above.

### C.1. Script "ip.sh"

```
#!/bin/sh
#####
#
# Create a fixed IP address
#
# If not connected to DHCP, the computer will use a fixed IP address
# (192.168.1.50 in this example).
# If your radio has a fixed IP address in the same subnet, this
# makes it easy to use the radio with a direct cable between RaspPi and radio
#
#####
#
cd $HOME

cat > etc_network_eth0 << '#EOF'
auto eth0
    iface eth0 inet static
    address 191.168.1.50
    netmask 255.255.255.0
    gateway 192.168.1.1
    dns-nameservers 192.168.1.1
#EOF
sudo cp etc_network_eth0 /etc/network/interfaces.d/eth0

rm etc_network_eth0
```

## C.2. Script "gpio.sh"

```
#!/bin/sh
#####
#
# GPIO-related stuff
#
# Even the latest version of wiringpi does not fully support the RPi 4
#
# Therefore, set all the GPIO pins to "input with pull-up"
#
#####
#
cd $HOME

cat > boot_config.txt << '#EOF'
#####
# setup GPIO pins start
#####
gpio=4-27=ip,pu
#####
# setup GPIO pins end
#####
#EOF
cat /boot/config.txt >> boot_config.txt
sudo cp boot_config.txt /boot/config.txt

rm boot_config.txt
```

### C.3. Script "pulseaudio.sh"

```
#!/bin/sh

cd $HOME

#
# make (if not yet existing) pulseaudio config director
#
mkdir -p $HOME/.config/pulse

cat > $HOME/.config/pulse/default.pa << '#EOF'
#!/usr/bin/pulseaudio -nF

.include /etc/pulse/default.pa

load-module module-null-sink sink_name=SDR-RX sink_properties="device.description=SDR-RX"
load-module module-null-sink sink_name=SDR-TX sink_properties="device.description=SDR-TX"

set-default-sink    SDR-TX
set-default-source  SDR-RX.monitor

#EOF

pulseaudio -k
sleep 2
pulseaudio -D
```

## C.4. Script "packages.sh"

```
#!/bin/sh
#####
#
# Load Raspian packages required to compile + run
# programs such as piHPSDR, fldigi, etc.
#
#####

cd $HOME

#
# -----
# Install standard tools and compilers
# -----
#
sudo apt-get --yes install build-essential
sudo apt-get --yes install module-assistant
sudo apt-get --yes install vim
sudo apt-get --yes install make
sudo apt-get --yes install gcc
sudo apt-get --yes install g++
sudo apt-get --yes install gfortran
sudo apt-get --yes install git
sudo apt-get --yes install pkg-config
sudo apt-get --yes install cmake
sudo apt-get --yes install autoconf
sudo apt-get --yes install automake
sudo apt-get --yes install libtool
sudo apt-get --yes install cppcheck
sudo apt-get --yes install dos2unix
#
# -----
# Install libraries necessary for piHPSDR
# -----
#
sudo apt-get --yes install libfftw3-dev
sudo apt-get --yes install libgtk-3-dev
sudo apt-get --yes install libasound2-dev
sudo apt-get --yes install libcurl4-openssl-dev
sudo apt-get --yes install libusb-1.0-0-dev
sudo apt-get --yes install libi2c-dev
sudo apt-get --yes install libgpiod-dev
sudo apt-get --yes install libpulse-dev
sudo apt-get --yes install pulseaudio
sudo apt-get --yes install pavucontrol

#
# -----
# Install standard libraries necessary for SOAPY
# -----
#
sudo apt-get install --yes libaio-dev
sudo apt-get install --yes libavahi-client-dev
sudo apt-get install --yes libad9361-dev
sudo apt-get install --yes bison
sudo apt-get install --yes flex
sudo apt-get install --yes libxml2-dev
#
# -----
# Install standard libraries necessary for FLDIGI
# -----
#
sudo apt-get install --yes libfltk1.3-dev
sudo apt-get install --yes portaudio19-dev
```

```

sudo apt-get install --yes libsamplerate0-dev
sudo apt-get install --yes libsndfile1-dev
#
# -----
# Install standard libraries necessary for WSJTX
# -----
#
sudo apt-get install --yes libboost-dev
sudo apt-get install --yes libboost-log-dev
sudo apt-get install --yes libboost-regex-dev
sudo apt-get install --yes qt5-default
sudo apt-get install --yes qttools5-dev
sudo apt-get install --yes qttools5-dev-tools
sudo apt-get install --yes qtmultimedia5-dev
sudo apt-get install --yes libqt5multimedia5-plugins
sudo apt-get install --yes libqt5serialport5-dev
sudo apt-get install --yes libudev-dev
#
# -----
# Install standard libraries necessary for FreeDV
# -----
#
sudo apt-get install --yes libspeexdsp-dev
sudo apt-get install --yes sox
sudo apt-get install --yes libwxgtk3.0-gtk-dev
sudo apt-get install --yes libao-dev
sudo apt-get install --yes libgsm-1
sudo apt-get install --yes libsndfile-dev

```

## C.5. Script "hamradio.sh"

```
#!/bin/sh
#####
#
# Download source code (do not yet compile) of some
# ham radio software packages, namely
#
# needed for piHPSDR:
# =====
# WDSP                FFT library for SDR programs
# piHPSDR             SDR program for HPSDR and Soapy radios
#
# only needed if piHPSDR is compiled with the SOAPYSDR option:
# =====
# SoapySDR core       core of the SoapySDR layer
# libiio              needed for Soapy PlutoSDR module
# PlutoSDR            SoapySDR module for AdaM Pluto
#
# if you want to run fldigi/wsjtq/freedv on the RaspPi:
# =====
# hamlib              TRX control library
#                     (needed for fldigi and wsjtq)
# fldigi              digimode program for RTTY, PSK, etc.
# wsjtq               digimode program for FT8, FT4, JT65, etc.
# freedv              digital voice program
#
#####
#
# DELETE all directories we are going to clone
#         this will delete all your personal setups there!
#
#####

cd $HOME

yes | rm -rf wdsp
yes | rm -rf pihpsdr
yes | rm -rf SoapySDR
yes | rm -rf libiio
yes | rm -rf SoapyPlutoSDR
yes | rm -rf hamlib
yes | rm -rf fldigi
yes | rm -rf wsjtq
#
# -----
# Download WDSP library
# -----
#
cd $HOME
git clone https://github.com/dl1ycf/wdsp
#
# -----
# Download piHPSDR
# -----
#
cd $HOME
git clone https://github.com/dl1ycf/pihpsdr
#
# -----
# Download SoapySDR core
# -----
#
cd $HOME
git clone https://github.com/pothosware/SoapySDR.git
```



```

# -----
# Download libiio (needed for Soapy Pluto)
# -----
#
cd $HOME
git clone https://github.com/analogdevicesinc/libiio.git
#
# -----
# Download the SoapySDR Pluto module
# -----
#
cd $HOME
git clone https://github.com/pothosware/SoapyPlutoSDR
#
# -----
# Download hamlib (use 4.4 release version)
# (needed for fldigi, GUI rig controller, and wsjtx)
# -----
#
cd $HOME
git clone https://github.com/hamlib/hamlib
cd hamlib
git checkout 4.4
#
# -----
# Download fldigi
# -----
#
cd $HOME
git clone https://git.code.sf.net/p/fldigi/fldigi
#
# -----
# Download wsjtx
# -----
#
cd $HOME
git clone https://git.code.sf.net/p/wsjt/wsjtx
#
# -----
# Download FreeDV
# -----
#
cd $HOME
rm -rf freedv-gui
git clone https://github.com/drowe67/freedv-gui.git

```

## C.6. Script "desktop.sh"

```
#!/bin/sh
#####
#
# Create Desktop Icons and startup scripts
# for piHPSDR, fldigi, wsjtx, FreeDV
#
#####

cd $HOME
#
# We invoke pihpsdr through a "shell script wrapper".
# This way we can take care the HPSDR logo is found
# and combine the stdout and stderr output into a single
# log file.
#
cat > $HOME/pihpsdr/pihpsdr.sh << '#EOF'
#!/bin/sh
cd $HOME/pihpsdr
rm -f hpsdr.png
cp release/pihpsdr/hpsdr.png hpsdr.png
./pihpsdr >pihpsdr.log 2>&1
#EOF
chmod 755 $HOME/pihpsdr/pihpsdr.sh

#
# The desktop files are created "the pedestrian way"
# then we can insert the actual home dir
#
FILE=$HOME/Desktop/pihpsdr.desktop
echo "[Desktop Entry]" > $FILE
echo "Name=piHPSDR" >> $FILE
echo "Icon=$HOME/pihpsdr/release/pihpsdr/hpsdr_icon.png" >> $FILE
echo "Exec=$HOME/pihpsdr/pihpsdr.sh" >> $FILE
echo "Type=Application" >> $FILE
echo "Terminal=false" >> $FILE
echo "StartupNotify=false" >> $FILE

FILE=$HOME/Desktop/fldigi.desktop
echo "[Desktop Entry]" > $FILE
echo "Name=Fldigi" >> $FILE
echo "Icon=$HOME/fldigi/data/fldigi-psk.png" >> $FILE
echo "Exec=$HOME//fldigi/src/fldigi" >> $FILE
echo "Type=Application" >> $FILE
echo "Terminal=false" >> $FILE
echo "StartupNotify=false" >> $FILE

FILE=$HOME/Desktop/wsjtx.desktop
echo "[Desktop Entry]" > $FILE
echo "Name=wsjtx" >> $FILE
echo "Icon=$HOME/wsjtx/icons/Unix/wsjtx_icon.png" >> $FILE
echo "Exec=$HOME/wsjtx/bin/wsjtx" >> $FILE
echo "Type=Application" >> $FILE
echo "Terminal=false" >> $FILE
echo "StartupNotify=false" >> $FILE

FILE=$HOME/Desktop/freedv.desktop
echo "[Desktop Entry]" > $FILE
echo "Name=FreeDV" >> $FILE
echo "Icon=$HOME/freedv-gui/contrib/freedv128x128.png" >> $FILE
echo "Exec=$HOME/freedv-gui/build_linux/src/freedv" >> $FILE
echo "Type=Application" >> $FILE
echo "Terminal=false" >> $FILE
echo "StartupNotify=false" >> $FILE
```

## C.7. Script "compile.sh"

```
#!/bin/sh
#####
# This cleans up everything in the source code
# directories, updates all downloaded source code
# trees, and recompiles everything.
#
# NOTE: the OS is not upgraded since the system
#       should be re-booted at least if the kernel
#       has been upgraded.
#####

cd $HOME

# number of CPUs to use in parallel make
export NPROCS=4

#####
#
# clean up everything
#
#####
#
make -C $HOME/pihpsdr    clean
make -C $HOME/wdsp      clean
make -C $HOME/hamlib    clean
make -C $HOME/flldigi   clean

rm -rf $HOME/wsjetx/build
rm -rf $HOME/wsjetx/bin
rm -rf $HOME/wsjetx/share

rm -rf $HOME/SoapySDR/build
rm -rf $HOME/libio/build
rm -rf $HOME/SoapyPlutoSDR/build

rm -rf $HOME/freedv-gui/codec2
rm -rf $HOME/freedv-gui/LPCNet
rm -rf $HOME/freedv-gui/build_linux

#####
#
# Update, compile and install SoapySDR core
#
#####

cd $HOME/SoapySDR
git pull
mkdir build
cd build
cmake ..
make -j $NPROCS
sudo make install
sudo ldconfig

#####
#
# Update, compile and install libio
#
#####

cd $HOME/libio
git pull
```

```

mkdir build
cd build
cmake ..
make -j $NPROCS
sudo make install
sudo ldconfig

#####
#
# Update, compile and install the
# SoapySDR Pluto module
#
#####

cd $HOME/SoapyPlutoSDR
git pull
mkdir build
cd build
cmake ..
make -j $NPROCS
sudo make install
sudo ldconfig

#####
#
# Update, compile and install WDSP
#
#####

cd $HOME/wdsp
git pull
make clean
make -j $NPROCS
sudo make install
sudo ldconfig

#####
#
# Update and compile piHPSDR
#
#####

cd $HOME/pihpsdr
git pull
make clean
make -j $NPROCS

#####
#
# Update, compile and install hamlib
# (no pull since we are at a fixed version)
#
#####

cd $HOME/hamlib
autoreconf -i
./configure --without-libusb
make -j $NPROCS
sudo make install
sudo ldconfig

#####
#
# Update and compile fldigi
#
# Note: I always have problems with the

```

```

#      national language support
#      therefore I switch it off
#
#      "confdialog" needs MUCH memory to compile
#      so use only one CPU otherwise compilation
#      may fail due to memory shortage!
#      (this is the case on 64-bit systems with gcc9)
#
#####

cd $HOME/fldigi
git pull
make clean
autoreconf -i
./configure --disable-flarq --disable-nls
make

#####
#
# Update and compile wsjtx
#
# Skip generation of documentation and man pages
# since we need TONS of further software
# (asciidoc, asciidoctor, texlive ...)
# to do so.
#
#####

cd $HOME/wsjtx
git pull
export CC=gcc
export CXX=g++
export FC=gfortran
mkdir bin
TARGET=$PWD
CMFLG="-DWSJT_GENERATE_DOCS=OFF -DWSJT_SKIP_MANPAGES=ON"
mkdir build
cd build
cmake -DCMAKE_INSTALL_PREFIX=$TARGET $CMFLG ..
cd ..
cmake --build build --target install -j $NPROCS

#####
#
# Update and compile FreeDV
#
#####

cd $HOME/freedv-gui
git pull
./build_linux.sh

```