

"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" Raspbian 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.

Step A: Obtain Raspbian 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 denoted as the "Raspbian 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)

RaspPi: 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 Windows can be found on the internet, see for example

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

Here I document how I did this on my Macintosh computer using standard commands, and the procedure should also work for a LINUX machine: I attached an USB card reader to my Macintosh and inserted a fresh card, then a volume with name „NO NAME“ appeared. With the command `df` (in the terminal window) I obtained a list of all mounted file systems, and the relevant line in the output read

```
/dev/disk3s1      . . . .  100%   /Volumes/NO NAME
```

which shows that the device associated with the SD card is `/dev/disk3`. Note that `/dev/disk3s1` is the device associated with the partition the volume is mounted on. On a LINUX system, this often reads `/dev/sdc1` (partition) from which you deduce the "whole disk" is `/dev/sdc`. Finally, using the commands

```
sudo umount -f /dev/disk3
sudo dd if=2019-09-26-raspbian-buster.img of=/dev/disk3 bs=8192k
```

the volume was un-mounted and the image written to the SD card. Since the IO-speed was about 10 MByte/sec, this took about six minutes.

TinkerBoard:

You do not need an USB card reader. Solely connect the TinkerBoard with a standard USB-C cable to your computer. Then it should behave as an external hard disk (just as an SD card in a reader). Then, proceed as above by typing (provided the TinkerBoard is `/dev/disk3`, and put the third line at the end of the second command)

```
sudo umount -f /dev/disk3
sudo dd if=20190821-tinker-board-linaro-stretch-alip-v2.0.11.img
      of=/dev/disk3 bs=8192k
```

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 piHPSPDR, you need tons of additional libraries. Fortunately, installing a component automatically installs its 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).

```
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 piHPSPDR 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 Broadcom 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 Raspberry 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 download 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_NOVAHI
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: 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).

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 release/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
```

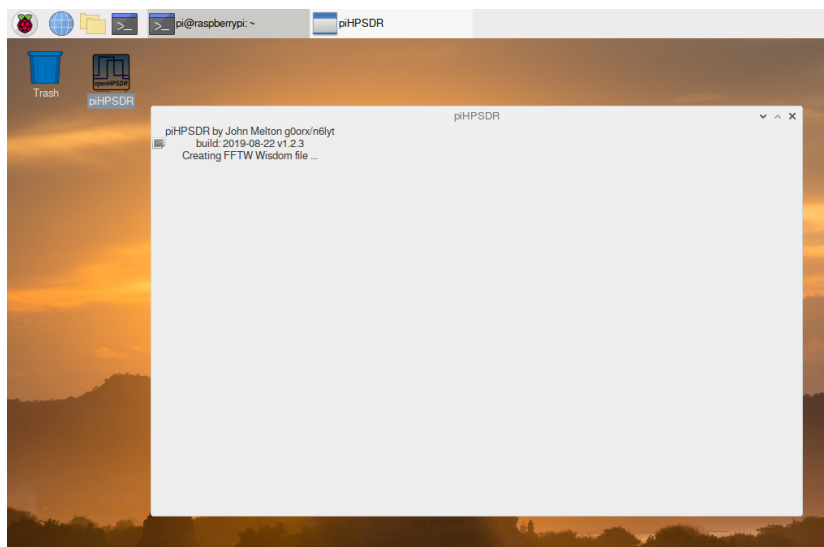
As soon as you have created this file (e.g. using the vi editor), an icon with text piHPSDR and the HPSPDR logo should occur on your Desktop. In order to get rid of being asked each time whether the program should be run in a terminal window, you issue the command (within a terminal window, and in the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

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 wisdom file (`wdspWisdom00`) as well as the configuration files (`*.props`) reside in that directory. This has the advantage that if you start piHPSDR from a terminal window from that directory, the wisdom and props files can be re-used.

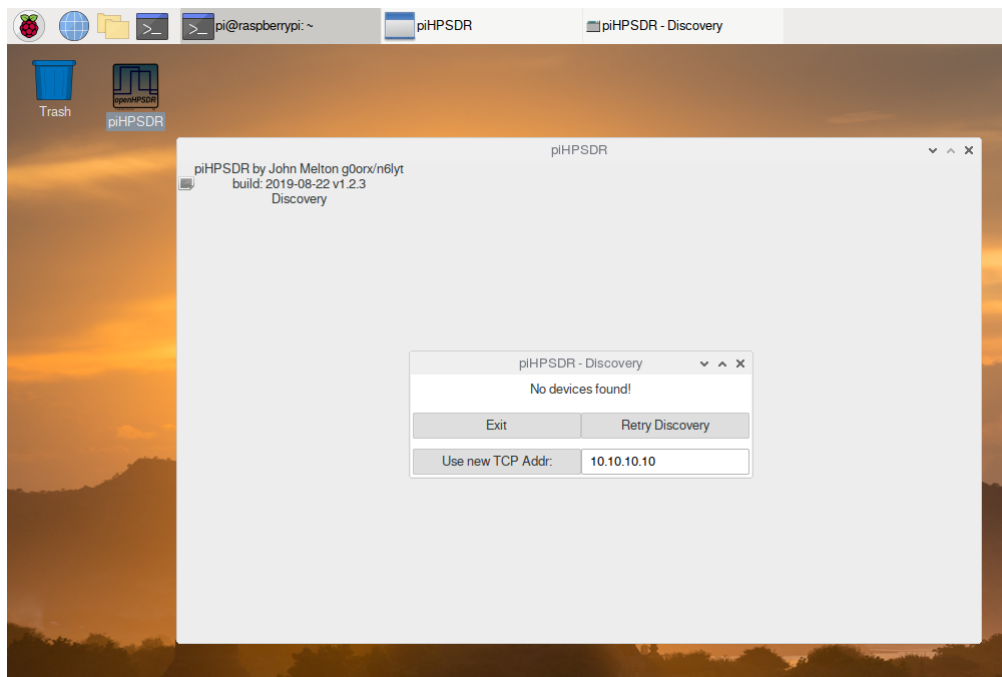
TinkerOS case: For TinkerOS, the procedure is essentially the same. Replace `/home/pi` everywhere by `/home/linaro`.

Step H: starting piHPSDR for the first time

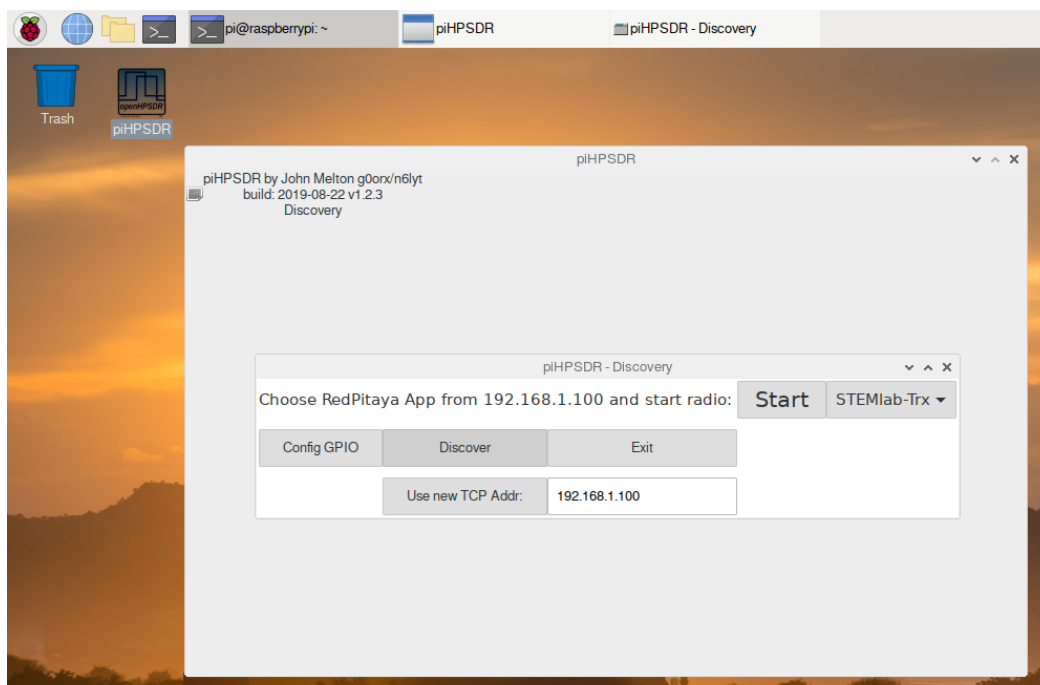
Now you can start piHPSDR by double-clicking on the piHPSDR icon on your desktop. The piHPSDR window should open and look like this:



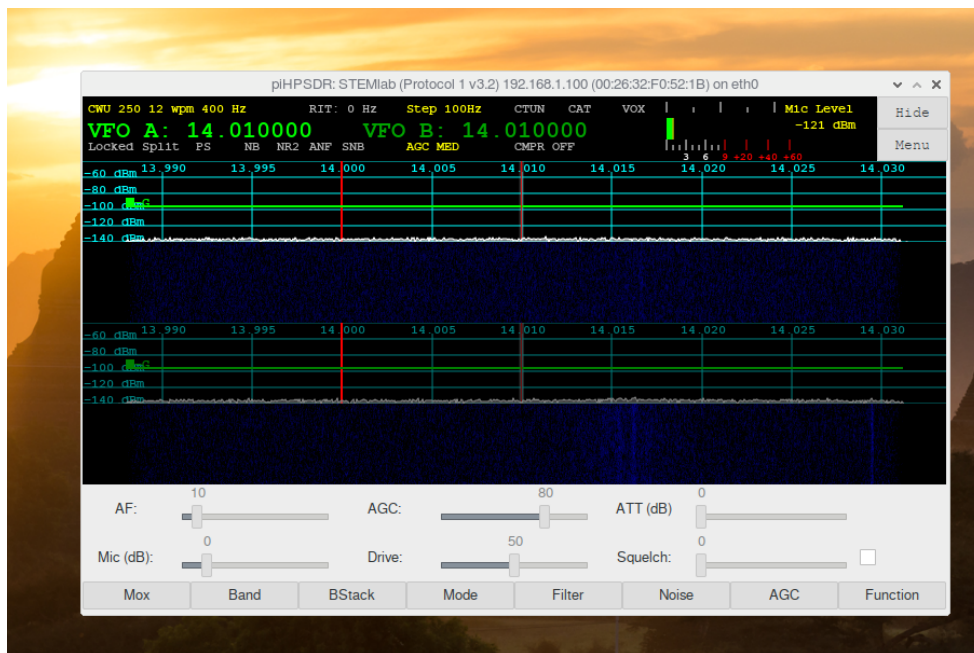
Because it is the first time you started the program, the WDSP library determine (once and for all) the optimum way to do the fast-Fourier-transforms (this will take few minutes). After this time, the piHPSDR window looked like this:



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

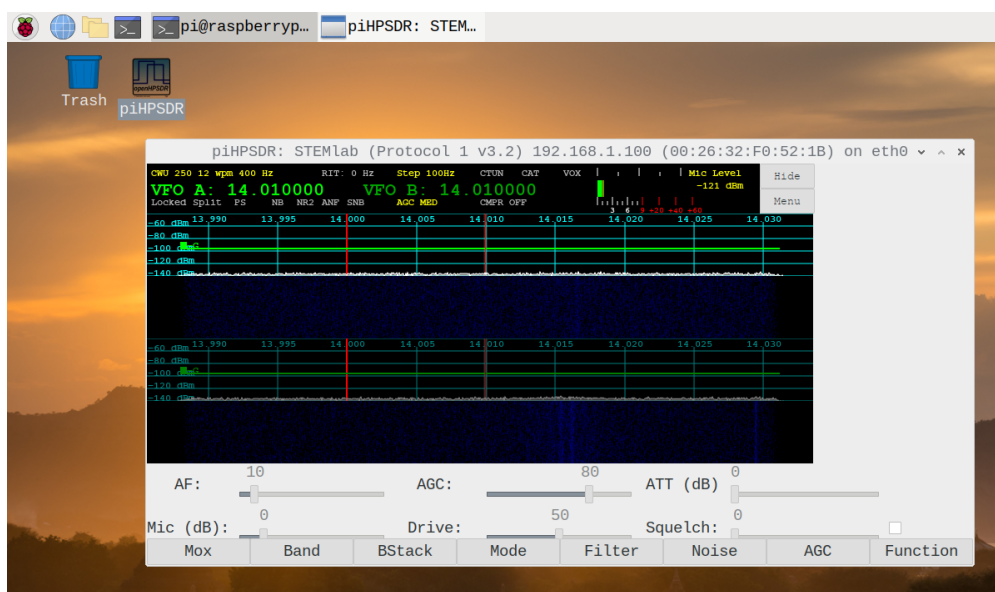


and after clicking the "Start" button, the radio finally started:



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 (192.168.1.50 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.