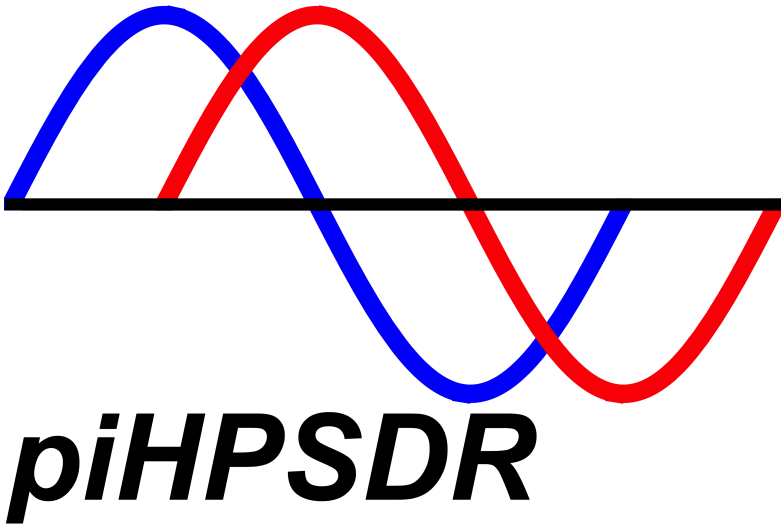

User's Manual

piHPSDR post-2.6 development Version

Christoph van Wüllen, DL1YCF
email: dl1ycf@darc.de

March 27, 2026

Copyright Notice:

Copyright (C) 2023–2026 Christoph van Wüllen, DL1YCF.

This work is licensed under the Creative Commons license CC BY-SA, version 4 or later, so it can be freely distributed. This license also allows re-users to distribute, modify and build upon the material in any medium or format, as long as attribution is given to the creator. The license allows for commercial use. If you modify or build upon the material, you must license the modified material under identical terms.

Disclaimer. The manual has been written with the intention that it is useful. It is quite clear that it still contains errors, therefore it is stressed here that it comes without any warranty. The reader is hereby explicitly warned that through wrong use of an SDR program such as piHPSDR, it is possible to damage the radio hardware.

Trade marks. Registered trade marks are not marked with a sign in this manual. From the absence of a trademark sign, it cannot be concluded that a mark you find in this manual is not registered or not protected.

The author:

Christoph van Wüllen (DL1YCF) has contributed a lot to piHPSDR in the last few years, this manual refers to the code in his GitHub account

`https://github.com/dl1ycf/pihpsdr`

Meanwhile, this code has numerous additions/corrections compared to the piHPSDR code in John Melton's master repository. A version history can be found in Chapter 1.3.

If you think you can improve the manual, you are welcome. Write an email to the author: `dl1ycf@darc.de`. Suggestions for corrections or additions to the piHPSDR code itself can be made with the standard tools on GitHub, but you can equally well write me an email.

Contents

1 Introduction **1**

1.1 What is piHPSDR? 1

1.2 Philosophy of the piHPSDR user interface 4

1.2.1 Screen size 4

1.2.2 Automatic vs. Manual settings 4

1.3 Version history 6

2 Prominent beginner's problems **9**

2.1 I cannot find a ready-to-run executable 9

2.2 No (or too little) RF power out when transmitting 10

2.3 The RX spectrum looks fine, but I hear no RX signal 10

2.4 How do I connect a PTT foot-switch or a Morse key 10

3 Starting piHPSDR for the first time **13**

4 Main window layout **23**

4.1 One or two receivers 23

4.2 The `Hide` button 25

4.3 What is a spectrum? 26

4.4 Spectrum scope options 28

4.5 The Slider Area 29
4.6 The toolbars 31
4.7 Mouse clicks in the main window 32
4.8 VFO bar and status indicators 34
4.9 Meter section 37

5 piHPSDR menus: introduction 39

5.1 piHPSDR menus 39
5.2 Using menus *and* controllers/panels 40
5.3 The main piHPSDR menu 41
5.4 The `Exit` Menu 42
5.5 The `About` Menu 44

6 Radio-related menus 47

6.1 The `Radio` Menu 47
6.1.1 General controls 48
6.1.2 Hardware specific settings 53
6.2 The `Screen` Menu 57
6.3 The `Display` Menu 61
6.3.1 Display Menu: General 61
6.3.2 Display Menu: Peak Labels 63
6.4 The `Meter` menu 64
6.5 The `XVTR` (Transverter) Menu 66
6.6 The `Server` Menu 69

7 VFO-related menus 71

7.1 The `VFO` menu 71
7.2 The `Band` menu 73

7.3 The `BndStack` (Band stack) menu 74
7.4 The `Mode` menu 75
7.5 The `Memory` menu 75

8 RX-related menus 77
8.1 The `RX` Menu 77
8.2 The `Filter` menu (including notches) 80
8.2.1 Choosing filters for non-FM modes 80
8.2.2 Choosing deviation for FM 83
8.2.3 Manual Multi-Notch Filter 83
8.3 The `Noise` Menu 85
8.3.1 Upper half of the Noise menu 86
8.3.2 NB settings 86
8.3.3 NR2 settings 87
8.3.4 NR4 settings 89
8.4 The `AGC` Menu 90
8.5 The `Diversity` Menu 90

9 TX-related menus 93
9.1 The `TX` Menu 93
9.1.1 TX Menu: TX Settings 94
9.1.2 TX Menu: CFC Settings 98
9.1.3 TX Menu: DEXP Settings 99
9.1.4 TX Menu: Peak Labels 101
9.1.5 TX Menu: TUNE and SWR 102
9.2 The `PA` Menu 104
9.2.1 PA calibration. 104
9.2.2 Watt meter calibration. 105

9.3 The VOX Menu 107
9.4 The PS (PureSignal) Menu 108
9.5 The CW Menu 115
9.5.1 Changing CW options 115
9.5.2 Changing CW texts 117

10 Menus for RX and TX 119
10.1 The DSP (Signal Processing) Menu 119
10.2 The Equaliser Menu 120
10.3 The Ant (Antenna) Menu 122
10.4 The OC (Open Collector) Menu 125

11 Controlling piHPSDR 127
11.1 The Toolbar Menu 127
11.2 The Sliders Menu 130
11.3 The CAT/TCI Menu 132
11.4 The MIDI Menu 135
11.5 The Encoders Menu 142
11.6 The Switches Menu 146
11.7 The G2panel Menu 147

12 Client/Server remote operation 151
12.1 Server side: the Server menu 152
12.2 Client side: connecting the server 153
12.3 Encryption and network security 155
12.4 Auto disconnect feature 156
12.5 Network bandwidth 156
12.6 Settings on the Client side 157

13 Latencies **159**

13.1 RF (base band) latencies 160

13.2 CW side tone latency 162

13.3 External solution to CW side tone latency 164

14 RX/TX profiles **167**

15 The TX audio chain **169**

15.1 Start of the TX chain: The MicLvl indicator 170

15.2 Monitoring the TX signal 171

15.3 The downward expander (DEXP), a flexible noise gate 172

15.4 MicGain and peak TX ALC value 172

15.5 The TX equaliser 173

15.6 The auto leveller 173

15.7 The speech processor (CMPR, CFC) 173

15.8 The controlled envelope SSB overshoot control (CESSB) . . . 174

15.9 The automatic level control (ALC) 175

16 Audio recording and replay **177**

17 HermesLite-II and RadioBerry support **181**

17.1 Default PA calibration values 182

17.2 Support for the HL2 I/O board 182

17.3 “TX latency” and “PTT hang time” 184

17.4 HL2 parameters in the panadapter 184

17.5 RadioBerry support 185

A List of piHPSDR Commands **187**

B The MULTI encoder **217**

C Keyboard bindings **219**

C.1 Accessibility for the Blind 219

C.2 Other keyboard shortcuts 221

D piHPSDR CAT commands **223**

D.1 Kenwood CAT commands 224

D.2 Extended CAT commands 236

E Connecting a Morse Key **249**

E.1 CW priorities 249

E.2 How to connect 250

E.2.1 RaspPi GPIO 251

E.2.2 MIDI 251

F piHPSDR and digi mode programs **253**

F.1 piHPSDR and Digi on different computers 253

F.2 piHPSDR and Digi on same computer 254

F.2.1 ALSA: Virtual Audio Cables 256

F.2.2 PipeWire: NullSink Modules 258

F.2.3 MacOS: LoopBack 263

G Compile-time options **265**

H RaspPi GPIO lines **269**

I RaspPi: Activating I2C **273**

J Linux: piHPSDR install from sources **275**

J.1 Fresh install from the internet 276
J.2 piHPSDR and SoapySDR 279
J.3 Upgrading an installation 282

K MacOS: piHPSDR install from sources 285

K.1 Fresh install from the internet 286
K.2 Upgrading an installation 291

L Connecting the Computer and the Radio 295

Chapter 1

Introduction

1.1 What is piHPSDR?

piHPSDR is a program that can operate with software defined radios (SDRs). As a graphical user interface, it uses the GTK-3 toolkit, while the actual signal processing is done by Warren Pratt's WDSP library. Thus, piHPSDR organises the transfer of digitised radio ,,intermediate frequency" (IF) data between the radio hardware and the WDSP library, the transfer of audio input (e.g. from a microphone) or output (e.g. to a headphone) data, as well as the processing of user input (either by mouse/touch-screen, keyboard, or external "knobs and buttons"), and the graphical display of the IF data. piHPSDR is intended to run on different variants of Unix/Linux. It runs on all sorts of Linux systems, including a Raspberry Pi (hence the name piHPSDR), but equally well on Linux desktop or laptop computers, and on Apple Macintosh (Mac OSX) computers which have a Unix variant under the hood. It is possible to run piHPSDR under the Windows operating system, see e.g. `https://github.com/ew8bak/pihpsdr`. There you find instructions on how to set up an ,,ecosystem" under Windows which can then be used to compile piHPSDR. The resource mentioned seems to come with an older version of piHPSDR, but others have reported that using that ,,ecosystem" it is possible to run the version of piHPSDR described in this document as well. Note that this is what has been reported to me, I never tried to use piHPSDR under Windows myself.

Although piHPSDR can be operated entirely by using mouse (or touch-screen) and keyboard as input devices, many users prefer to have physical push-buttons and/or knobs or dials. To this end, piHPSDR can control push-buttons and rotary encoders connected to the GPIO (general purpose input/output) lines of a Raspberry Pi. At least two generations of such controllers have been put on the market by Apache labs, and I know of several projects where home made controllers have successfully been made. As an alternative, MIDI devices can be used for user interaction. For desktop/laptop computers that do not have GPIO lines, MIDI offers an easy-to-use possibility of having push-buttons and dials that control piHPSDR. Apart from home-brew projects in which a micro-controller such as an Arduino Micro controls the actual buttons/knobs and acts as a MIDI device to the computer to which it is connected via USB, there are low-cost so-called "DJ controllers" (DJ stands for disk jockey) from various brands which are sold by music stores and which have successfully been used with piHPSDR. A third possibility to control piHPSDR is via a serial interface through CAT (computer aided transceiver) commands. The CAT model used by piHPSDR is based on the Kenwood TS-2000 command set with lots of PowerSDR extensions. The ANDROMEDA controller uses such CAT commands to communicate with piHPSDR, so using CAT is, in addition to MIDI, the second option to have buttons and knobs for general (non-GPIO) computers.

Using a touch-screen instead of a mouse offers the possibility to put the actual radio hardware together with a Raspberry Pi running piHPSDR and an assortment of buttons/knobs into a single enclosure. This way, one can build an SDR radio which can be operated like a conventional analog one.

The piHPSDR program has been written by John Melton G0ORX/N6LYT. It is free software that is licensed under the GNU (free software foundation) general public license. Many other radio amateurs have contributed to the code. A lot of extensions and improvements have been added by myself, therefore this document refers to the version of piHPSDR that can be found on my GitHub account `https://github.com/dl1ycf/pihpsdr`.

Because piHPSDR can be used on many different types of computers, and because operating systems change rather quickly over time, I do no longer have pre-compiled binaries in the repository. Such binaries usually only work on a specific hardware, and with a specific variant (e.g. 32-bit compared to 64-bit) of the operating system. It is therefore necessary to build piHPSDR *on the*

target system from the sources. I know this sounds daunting to non-LINUX-nerds. But I have put much effort into making the installation/compilation procedure semi-automatic and easy-to-use for "the rest of us". To this end, I provide so-called shell scripts which do all the work behind the scene. The procedure is described in some detail in Appendix J for Linux (including RaspPi) computers and in Appendix K for Mac OSX. Several ,,absolute beginners" have meanwhile successfully downloaded and installed piHPSDR from the sources, so you will manage to do so as well. The installation procedure also builds a Desktop icon such that piHPSDR can be started just by double-clicking this icon. On MacOS, a so-called ,,app bundle" is made (via ,,make app") which can be moved to another place (e.g. the Desktop) but which will not run if transferred to other MacOS machines. Again, it is not overly difficult to bundle everything that is needed inside this ,,app bundle", but this then will only run on certain MacOS versions and not on others. So also on MacOS, it is preferred to build piHPSDR from the sources.

This manual starts in its first chapter with the first invocation of a freshly compiled piHPSDR. Within this manual, we shall use a typewriter font in red colour if we refer to a text or a button within a menu of within the VFO bar. piHPSDR menus and commands are indicated through a typewriter font printed in blue. The author hopes that this improves readability. In some cases, if the name of a command or menu is written on the screen, you may find the same string both typed in red and blue in the description, depending on whether the text refers to the command in an abstract sense or to the string as it can be found on the screen. This may be confusing upon first reading, but I shall try to follow the colouring convention as laid out here.

1.2 Philosophy of the piHPSDR user interface

In this section some of the design principles of piHPSDR are explained, as they are the reason why I sometimes say ,,no" to feature requests.

1.2.1 Screen size

piHPSDR is designed to run on really small screens with sizes down to 800x480 pixels. Although it is perfectly possible to run piHPSDR in large windows (see the `Screen` menu, chapter 6.2), the requirement that is must also work on a small screen restricts the size that menus and submenus can have and the number of control elements that can be put on the screen. When it comes to building a ,,standalone" radio from a small SDR board, a single-board computer and a small screen, piHPSDR is the software of choice, and any new features to be built in are checked whether they possibly destroy the possibility of building such a radio.

1.2.2 Automatic vs. Manual settings

I am often asked to make more controls user-accessible, allowing them to fine-tune the radio's behaviour. Such suggestions are often turned down in favour of making things work automatically. This makes using piHPSDR easier and the results are often better, compared what happens is users have too many controls to play with. So making things automatic does not only allow for a simpler user interface, but also gives decent results with a flat learning curve.

To illustrate this, I give two examples: in piHPSDR, when activating a CW audio peak filter, the only user control is to turn it on or off, and the bandwidth and gain of that filter is automatically calculated, based on the characteristics of the main filter. I do not allow the user to individually choose the width of the audio peak filter since there are certainly unreasonable combinations of the base and peak filter widths. The advantage is, that each time the main filter is changed, the audio peak filter bandwidth is recalculated.

The second example is SSB mode with TX compression. Activating compression automatically activates the auto-leveler in the TX chain (with 8 dB amplification, see chapter 15). Furthermore, if the compression level exceeds 5 dB, the CESSB overshoot control is automatically activated without the user needing to think about CESSB. As an aside, this also excludes the use of low-latency filter for the TX signal since this is not compatible with CESSB. Of course there may be special situations in which, for example, using the compressor *without* auto-leveling is helpful, but this is the exception. So for the benefit of the average user who certainly profits from activating the auto-leveler automatically when activating compression, these functions are tied together in piHPSDR.

1.3 Version history

Here we list the main new featured introduced in each version. **piHPSDR**

development version

- Added a manual multi-notch filter (`Filter` menu).

piHPSDR Version 2.6, released February 2026

- Low-latency CW side tone also for the PulseAudio module.
- One serial port can be used for a PTT-input (DTR/CTS) and a PTT-output (RTS) line.
- Added CW texts that can be edited in the CW menu, and transmitted upon pushing a phyiscal (controller/panel) button or clicking a toolbar (on-screen) button.
- NR3 (RNNnoise) and NR4 (LibSpecBleach) noise reduction models fully integrated.
- Performance and Resilience improvement in Client/Server operation.
- RX1 and TX audio settings (audio device, Stereo/Left/Right) are now part of the RX/TX profile stored in the props file, and automatically switched when changing the mode.
- TX audio monitor option: the audio that goes into the TX chain (before being processed by the compressor or the equalizer) can be routed to the RX audio output.

piHPSDR Version 2.5, released October 2025

- First complete version of the Client/Server model (including TX and CW). Server access is password protected.
- Better HermesLite-II and RadioBerry support.
- When both a scope and a waterfall is running for a receiver, the (relative) height of the waterfall is user-adjustable.
- Text-to-Speech now also in the discovery screen, and Text-to-Speech uses the MacOS native capability if compiled on MacOS.
- Better support for LIME-SDR (2RX) and LIME-mini (1RX) radios.
- Option to have an SWR-dependent side tone while TUNE-ing.
- ZOOM now up to 32x
- User-configurable Slider and Toolbar areas, with 0–3 slider rows and 0–3 toolbars.
- Added local replay of captured audio data (`Replay` command) – useful to check captured audio data before transmitting it.
- Added the NR2 noise reduction post processing introduced with WDSP 1.27 (`Noise` menu)
- GPIO code now works both with the V1 and V2 `libgpiod` API

piHPSDR Version 2.4, released January 2025

- Possibility to label peaks (with dBm values) in the RX/TX panadapters (contributed by Lukasz).
- Dynamically change the font piHPSDR is using (`Screen` menu)
- Dynamically assign functions to G2 Ultra panel knobs/buttons (`G2panel` menu)
- Stripped-down TCI server for use with logbook programs and linear amplifiers (`CAT/TCI` menu)

piHPSDR Version 2.3, released August 2024

- Preliminary "accessibility" support for the blind (see Appendix C.1)
- Added the "trained" NR2 noise reduction method introduced with WDSP 1.25 (`Noise` menu)
- Extended the RX and TX equalisers to 10 bands (`Equaliser` menu)
- Support for HermesLite-II I/O board
- Added audio capture and replay (see Chapter 16)
- Added a multi-function encoder (Appendix B)

piHPSDR Version 2.2, released October 2023

- Support for ANAN-G2 radios
- Added a CW audio peak filter (`Filter` menu)
- A new double-buffering scheme that "survives" short-time CPU stalling in the DSP functions
- Support for ANDROMEDA controllers
- Added an in-depth manual (this document)
- piHPSDR now works both with light and dark GTK themes
- Option for "touch screen friendly" combo-boxes
- Dynamically change the size of the piHPSDR window (`Screen` menu)
- Unified layout for nearly all menus

piHPSDR Version 2.0.8

The last synchronisation with John Melton's (G0ORX) piHPSDR version 2.0.8 took place in January 2022.

Chapter 2

Prominent beginner's problems

Over the years, many problems have been reported to me by those who start using piHPSDR. In this section, I want to address those problems which occurred frequently, and point to the chapter in the manual where this is addressed. My selection of ,,prominent problems" will be extended upon request, or if I get the feeling that a problem not yet mentions pops up too frequently.

2.1 I cannot find a ready-to-run executable

piHPSDR is meant to run on a variety of very different computers, from desktop computer running MacOS or LINUX to small single-board computers such as the RasperryPi. Many different versions of flavours (32-bit vs. 64-bit) of operating systems are around, such that maintaining executables for all those systems is a too high burden for me.

Therefore, piHPSDR must be compiled from the sources which are freely available in the internet. Detailed procedures how to do so are descriped in Appendix J for Linux computers (including the RaspberryPi) and in Appendix K for Macintosh MacOS-X computers.

Meanwhile lots of ,,absolute beginners" have successfully managed to get piHPSDR installed and running this way, and you will also be able to do so.

2.2 No (or too little) RF power out when transmitting

This is actually the most prominent beginner's problem. Read Sec. 9.2 on the PA menu, where one has to adjust the output level up to the nominal power of the PA. By default piHPSDR wants to be on the safe side, therefore many radios produce no or very little output power before you make the proper adjustments in the PA menu.

2.3 The RX spectrum looks fine, but I hear no RX signal

Many radios do not have built-in audio codecs. In this case, you must choose local audio output in the RX menu and connect a head-phone to the sound card selected there. The same applies to the microphone: choose a sound card with a mic jack in the TX menu for audio input and connect your microphone there.

2.4 How do I connect a PTT foot-switch or a Morse key

The standard setup of many users is that they run piHPSDR on a computer placed in close proximity to the radio, and that the radio then provides jacks for headphone, microphone, Morse key, and PTT foot-switch. In this case, there is little need to worry how to connect such gear to the *computer* running piHPSDR (see also the previous section).

This changes if there is a long distance between computer and radio. Imagine radio is placed in the attic and you are sitting in your living room operating the radio through an ethernet cable that connects the computer on your desktop with the radio in the attic. The same applies if you are running piHPSDR in client mode (see chapter 12) and operate a remote station over the internet. Note further that there are radios which do not have any jacks for the mentioned hardware. In all these cases you have to connect things to

your computer. For a headphone and a microphone, this is explained in the previous section, but you also need at least a PTT switch (for SSB operation) and a Morse key (for CW).

Originally piHPSDR was designed to run on small single-board computers which offer lots of general-purpose input/output (GPIO) lines that can be used for connecting e.g. a foot switch and a key (see Appendix H), but if piHPSDR is running on a desktop or laptop computer, then the computer offers no such GPIO lines.

The most versatile way for providing input lines is certainly to use a small micro-controller (uC) which has such lines and which communicates with the computer using MIDI commands over an USB connection. Chapter 11.4 explains in detail how one can ,,teach" piHPSDR to make use of such MIDI commands (see also Appendix E). Personally I use a standard CW keyer software on an ArduinoMicro with a small extension that sends MIDI commands for PTT on/off and Key up/down events. A ,,luxury" version thereof is described in

`https://github.com/softerhardware/CWKeyer`

which does not only implement a CW keyer, but also a sound card in a single USB connection. So you can connect a headphone, a microphone (with PTT contact), and a CW key to that small device consisting of a Teensy4.0 uC and an audio codec. I have also built a prototype thereof using the standard Teensy AudioShield.

Many users are not very familiar with uCs and and a maximally simple way to connect a PTT footswitch to a desktop computer. For those, a dedicated serial port (or USB-serial adapter) can be specified in the CAT/TCI menu (chapter 11.3) which can then be used for PTT input simply by connecting the DTR and CTS serial lines with a foot switch or with an (isolated) microphone PTT contact. In addition, the serial line signals on the RTS line whether piHPSDR is transmitting or not.

Chapter 3

Starting piHPSDR for the first time

Let us assume you have an SDR (say, an Anan-7000 or a HermesLite-II) powered up and connected to an antenna, and you have piHPSDR installed on a computer (say, a Raspberry Pi or an Apple Macintosh). The first thing to do is to establish a proper connection between the computer and the radio. Although advocated at many places, I do highly recommend against a WiFi connection. WiFi routers often use optimisations where they hold back data packets for a given client for a while, to be able to send a collection of them in a burst. While this certainly optimises the through-put because it minimises clear-channel arbitration events, such jitters are disastrous in SDR operation. The easiest way of connecting the radio and the computer is to have a managed switch (or router) with a built-in DHCP server, and to connect both the computer and the radio with a suitable cable to the switch. If the computer has both a RJ45 jack for an ethernet cable, and a WiFi interface, my personal recommendation is to use WiFi to connect the computer to the internet, and use a single direct cable plugged into the RJ45 jacks of the computer and of the radio (see Appendix L).

If the piHPSDR program is started for the first time, it opens a window that looks like Fig. 3.1. Besides stating a version number, and the ID string and the date of latest commit (change) applied to the repository from which piHPSDR was built, the list of optional features is documented (compile-time options, in this case GPIO, MIDI, SATURN) as well as the audio module used

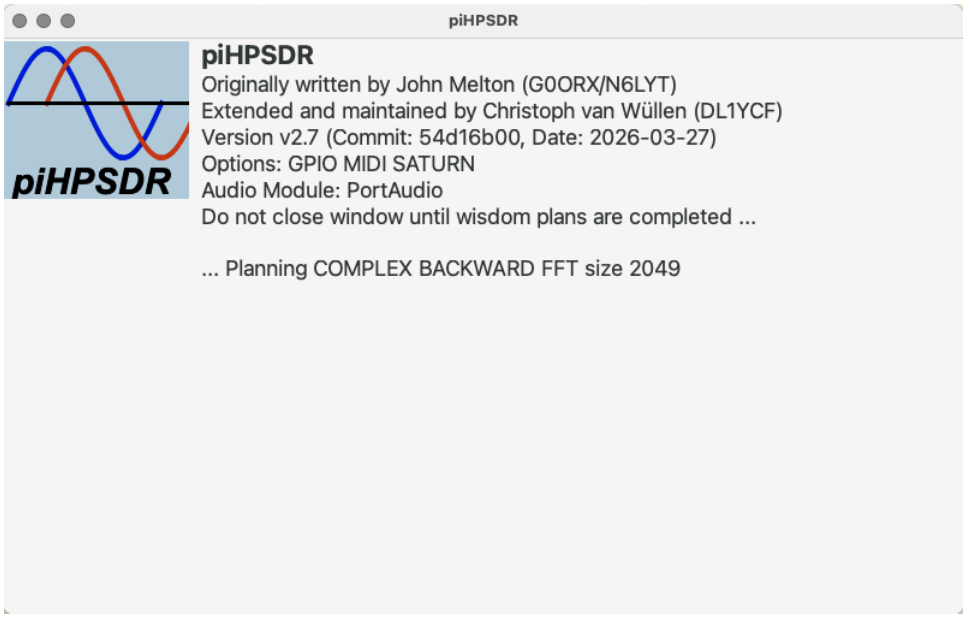

Fig. 3.1: piHPSDR screen while completing the *wisdom plans*.

(here: PulseAudio) for attaching headphones or microphones (via a sound card) to the host computer running piHPSDR. Information about compile-time options and how to add or remove them, and on the available audio modules, are given in Appendix G.

What is important here that you have to wait. This only applies to the very first time you start piHPSDR. On CPUs with a rather simple instruction set (like the ARM processor in the Raspberry Pi, or the Apple Silicon processor in recent Macintosh computers), this so-called *planning* step is quite fast. For example, on my Apple M2 Mac mini, this step only takes 6 seconds, and you have to wait for 34 seconds on a RaspberryPi 4. On the contrary, on CPUs with complex instruction sets, more planning is necessary: on a Mac mini with a 3 GHz x86 processor, it takes 16 minutes! But note this has only to be done once, in subsequent starts of piHPSDR, the wisdom will simply be read from the file created during the *planning*. These plans contain, for a large number of dimensions, the fastest way to perform fast Fourier transformations (FFTs) on the given CPU. When this ,,wisdom" is secured, piHPSDR tries to detect a radio on the network. If everything went well with the network connection, you then see a screen with a discovery

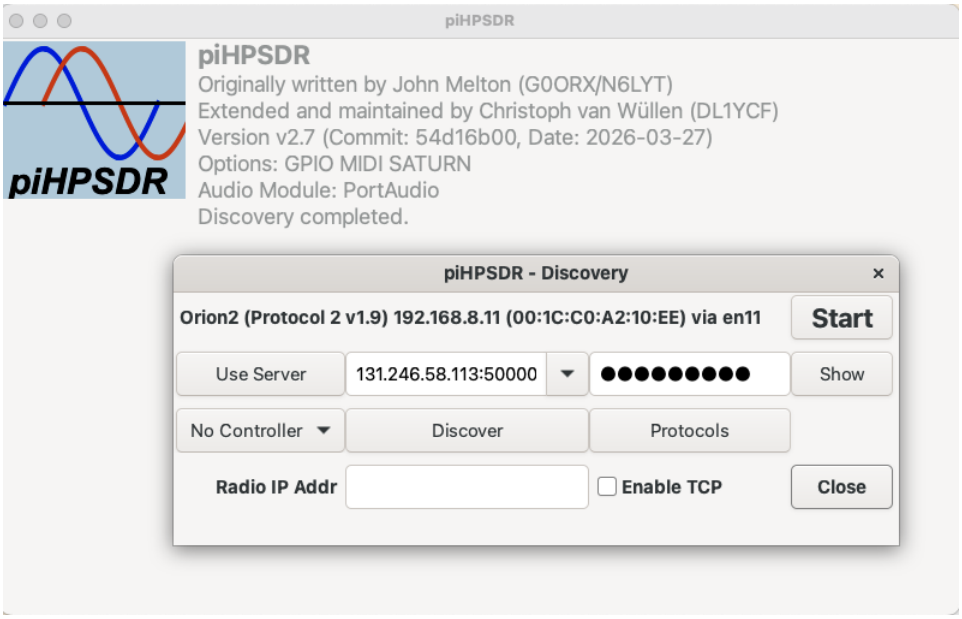

Fig. 3.2: A radio has been discovered. You are ready to start it.

menu (Fig. 3.2).

The Start button

It is possible that the `Start` button is deactivated and shows another text. `In Use` at this point means that the radio is already in use (connected by another SDR application), while `Incompatible` denotes that the radio is not compatible with this version of piHPSDR. This is only the case if piHPSDR runs on the compute module inside the new Anan Saturn/G2 radios and the FPGA firmware is known to be too old (or too new) for direct (XDMA) data transfer between the compute module and the FPGA. Updating both piHPSDR and the FPGA firmware to the latest version should help in this case. Should the start button read `Subnet!`, then the IP address of the radio is outside the range of the network adapter which received the discovery packet. In most cases when you see this, it results from a radio being caught in the act of initialising itself, and pressing the `Discover` button again to re-start the discovery process then usually lets the problem vanish.

Note when using a VPN tunnel: I have heard of cases where VPN is used to attach a remote subnet as if it were a local one, and where the virtual

,,ethernet adapter" that realises the VPN tunnel is assigned a netmask that is too narrow such that the start button is deactivated and shows the text `Subnet!` although a connection is indeed possible. In such cases, the easiest remedy is to manually type in the IP address of the radio (which is shown in the line containing the deactivated button) in the `Radio IP Addr` text field (see below) and then click the `Discover` button.

If more than one radio is available, or a radio can be connected via more than one network interface, you will see several `Start` buttons. If you see at least one `Start` button, you can start the corresponding radio simply by clicking that button.

The line of control elements following the radios discovered begins with the `Use Server` button, and belongs to the client/server feature that is described in chapter 12, it will not be discussed here where we now explain the purpose of the remaining control elements. Easiest to explain is the `Close` button, this will simply terminate the program. Most likely, you may want to go into the `Protocols` menu sooner or later. By default, piHPSDR tries to discover the presence of a radio using all protocols known to piHPSDR. However, if you know that your radio, for example, uses Protocol-2, then trying to discover a Protocol-1 radio is just a waste of time. So if you know which types of radio you want to connect to, you can enable (only) these in the `Protocols` menu. The available protocols are

`Protocol 1` This is the "original" HPSDR protocol.

`Protocol 2` This is the "new" HPSDR protocol.

`Saturn XDMA` This is used to talk to a Saturn FPGA through the internal XDMA interface. Only available if piHPSDR is compiled with the `SATURN` option.

`USB OZY` This is used to talk to a radio using the legacy USB OZY interface. Only available if piHPSDR is compiled with the `USBOZY` option.

`SoapySDR` This is used to talk to a radio through the SoapySDR library, for example to an AdalmPluto. Only available if piHPSDR is compiled with the `SOAPYSDR` option.

`STEMlab` This is used to connect to RedPitaya based SDRs through the WEB interface. Only available if piHPSDR is compiled with the `STEMLAB_DISCOVERY` option. Starting the radio using this protocol is a two-step process: first, the RedPitaya's WEB interface is located, and the `Start` button then starts the SDR app on the RedPitaya. Then, piHPSDR tries to connect to this SDR app and upon success offers a new `Start` button to start the radio. If the RedPitaya is exclusively used as a radio, it is recommended to auto-start the SDR app when the RedPitaya is powered up. In this case, the STEMlab protocol is not used, because the SDR app can be started through Protocol-2.

`Autostart` This is a very useful option. It indicates that if exactly one radio has been found, it is automatically started. So in normal operation, when starting piHPSDR subsequently, and all settings are still valid, the radio is started without user intervention. If this option is activated and one radio is present, you will not see this menu, so in order to make further changes here, you have to disconnect the radio from the ethernet cable, start piHPSDR until you see this menu, and update the `Protocols` settings. Then you can re-discover using the `Discover` button. If you have an Anan-G2 radio and activated `Autostart`, then you have a problem. Since always exactly one radio (the G2) is discovered, you will always step over the discovery menu and start the radio right away, which means you cannot change any setting in the discovery menu. The only way to get of of this trap is to remove the file `protocols.props` in the directory where piHPSDR stores its settings. On a G2, this is usually `$HOME/github/pihpsdr` or `$HOME/pihpsdr`.

Connection to a radio with known IP address

Sometime piHPSDR *needs* to know the IP address of the radio, for example to perform a `STEMlab` discovery as described above. But also if the radio and the computer running piHPSDR are not on the same network segment, a successful radio discovery may only be possible by directly querying the radio by its IP address. To use this feature, the IP address in numerical

form (xxx.xxx.xxx.xxx) can be entered in the box with the label `Radio IP Addr`. If a legal IP address is contained in this box, protocol-1 and protocol-2 discovery queries start with sending a discovery packet to the specified IP address. With a known IP address (and if supported by the radio), one can connect to radios which are not on the same sub-net as the computer, in principle you can connect to any radio on the world (either directly or through a VPN tunnel) provided it is on the internet. My advice is to always use this possibility if the IP address of the radio is known: if a radio is discovered this way, further discovery with broadcast packets is not attempted, so you can simply connect faster specifying the IP address.

In rare cases (the RedPitaya STEMlab and RadioBerry cases are the only one I know of) it is possible to use the TCP protocol rather than the standard UDP one for communicating with the radio. If the `Enable TCP` box is checked, a discovery packet will be sent via TCP to the specified IP address, and then piHPSDR waits up to three seconds for a response. For radios that are not TCP-enabled, it is thus best *not* to enable TCP, to avoid this three-second delay.

The `Discover` button re-starts the discovery process. This is useful if the radio has been powered up too late and was not yet ready when piHPSDR was started. Simply press `Discover` to start another attempt. This is largely equivalent to quitting and restarting piHPSDR but much more convenient.

Specifying a ,,piHPSDR controller"

If piHPSDR is compiled with GPIO support, there is a combo-box (pop-down menu) to the left of the `Discover` button lets you choose which type of GPIO based controller you have attached to the computer. Note this menu does not appear on SATURN (Anan G2) radios, since here the correct choice is made automatically. The possible choices are

`No Controller` Choose this if no GPIO controller is wired to your Raspberry Pi. This is the default when you first start piHPSDR. Some GPIO lines are still used e.g. for PTT or CW. If this conflicts with other hardware connected to the GPIO, it is better to compile piHPSDR without the GPIO option.

`Controller1` Choose this if you have the original piHPSDR controller put to the market by ApacheLabs (or a clone thereof), called the Controller1 in the rest of this manual.

`Controller2 V1` This option is valid for some early prototypes of the "version 2" controller with single encoders. This special case is not covered in this manual.

`Controller2 V2` Choose this if you have a "version 2" piHPSDR controller from ApacheLabs (with double encoders on a single shaft), or a clone thereof. This controller is denoted Controller2 in the rest of this manual.

—— Attention ——

Be sure to choose a controller only if such a controller is actually connected to your Raspberry Pi. Unexpected things can happen if you chose one type of controller here but have totally different hardware connected to the GPIO. For example, if you have a so-called ,,audio hat" attached to the RaspPi GPIO, then it is safest to compile piHPSDR without GPIO support.

All settings (protocols, controller, IP address, Server address) made in this menu are stored in radio-independent) files (`gpio.props`, `ipaddr.props`, `protocols.props`, `remote.props`) and are restored when piHPSDR is started the next time.

If all went well, a radio could be discovered and you hit the `Start` button, the radio is started, and if this succeeds, you see something like shown in Fig. 3.3.

At the top (denoted ,,VFO bar" in Fig. 3.3), just below the window title, you have the VFO bar which contains information on the frequencies of the two VFOs A and B, as well as lots of further information, to be explained later. At the top right, there are two buttons `Hide` and `Menu` which will be explained in the next chapter. To the left of these two buttons, there is the meter bar which by default is a digital S-meter.

Below you see the panels (panadapters) for two receivers (,,RX1 panel" and ,,RX2 panel"), stacked vertically. Both panadapters show both a spectrum and a waterfall. The default setting, if both are shown, is that the spectrum gets 75% and the waterfall 25% of the vertical space of the RX panel. Below the RX panel(s), there is the ,,Slider Area" which has two rows of sliders,

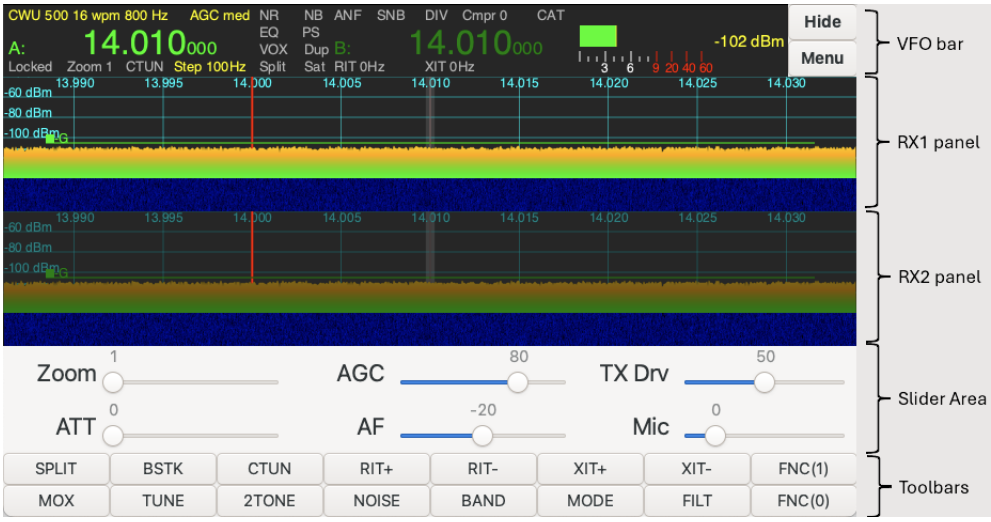

Fig. 3.3: First start of the radio with defaults settings.

each row containing three sliders. At the bottom of the window you see two toolbars, each with eight buttons.

All this is fully user configurable. The user can

- Change the overall size of the piHPSDR window (`Screen` menu). This includes a ,,full screen" option where piHPSDR uses the entire space on the monitor.
- Choose different VFO bar layouts, depending on the size of the piHPSDR window (`Screen` menu).
- Choose whether piHPSDR runs one or two receivers (`Radio` menu).
- Choose (for each receiver) whether a spectrum, a waterfall, or both are displayed in the receiver panel (`Display` menu).
- Set, if both spectrum and waterfall are displayed, how much of the vertical space the waterfall gets (`Display` menu).
- Choose several filling or colouring options for the spectrum (`Display` menu, see Sec. 4.4).
- Choose, if two receivers are run, whether their panels are stacked horizontally or vertically (`Display` menu).
- Choose the number of rows (0–3) to display in the Slider area (`Screen` menu).
- Assign functions to each of the 9 possible sliders (`Sliders` menu).
- Choose the number of toolbars (0–3) to be shown at the bottom of the window (`Screen` menu).
- Assign functions to toolbar buttons (`Toolbar` menu).

In summary, piHPSDR now allows the user to adjust the layout of the window to the needs of the user. For example, CW operators may not need a slider to adjust Mic gain or TX compression but will want a slider to adjust the CW speed. FM operators may need a Squelch slider, but for SSB operators a slider to change the level (and enable/disable) a TX compressor (speech processor) will be more important.

Chapter 4

Main window layout

4.1 One or two receivers

At the end of the previous chapter (Fig. 3.3), there were two receiver panels in the piHPSDR window, and both including a spectrum scope (the green-coloured noise floor) and a waterfall. The waterfall area is completely black in the above picture since there was no RF signal. piHPSDR can be switched between having one or two receivers in the `Radio` menu. If there are two receivers (called RX1 and RX2), one of them is the *active receiver*. If you look closely at the above picture, you will note that the spectrum scope of the lower (RX2) panel is shaded, while it is in bright colour for RX1. This indicates that RX1 is currently the active receiver. By simply clicking into the panel of the other (inactive) receiver, either with a mouse or on a touch screen, the formerly inactive receiver becomes active.

Many conventional rigs with two independent receivers discriminate between the *main* and the *sub* receiver. It is important that this is **not** the case for piHPSDR. In piHPSDR, both receivers are largely equivalent. For example, if you start transmitting in normal (non-split) mode, the TX frequency matches the frequency of the active receiver, no matter whether this is RX1 or RX2. Likewise, in split mode, the TX frequency matches the frequency of the non-active receiver. Most of the receiver-specific controls, for example adjusting the AF volume or the AGC gain, refer to the current active receiver. If piHPSDR runs with two receivers, RX1 is always controlled by

VFO-A while RX2 is controlled by VFO-B. The VFO settings not only include the frequency but also the current mode (e.g. LSB or CWU), the filter setting, the band and band stack setting, whether RIT is enabled or not, and the RIT offset. So changing the RIT value only changes it for the active receiver. If you want to change the RIT value for RX2 while RX1 is the active receiver, you have to make RX2 active, change the RIT value and then make RX1 active again.

RX1 and RX2 are largely independent. They can receive on different bands. They can receive from different antennas provided the radio has two RF front-end with two analog-to-digital converters (ADCs), as most modern radios do. In this case, one usually assigns the first ADC (ADC0) to RX1 and the second ADC (ADC1) to RX2. This can be done in the `RX` menu.

By default, if there are two receivers, they are vertically stacked, with RX1 in the upper part and RX2 in the lower part of the display. This can be changed in the `Screen` menu to horizontal stacking, where RX1 is in the left half and RX2 in the right half of the display. Changing the stacking trades vertical against horizontal resolution, of course.

Fig. 4.1: piHPSDR with a single RX, three rows of sliders and a single toolbar.

Fig. 4.1 picture shows, for demonstration purpose, a piHPSDR window with a single receiver, and three rows of sliders but only a single toolbar. There

is a strong S9 (-73 dBm) signal at 14.1 kHz which has left a spur in the waterfall.

If there is only one receiver, it is controlled by VFO-A. VFO-B then actually only controls the transmitter if using split mode, but the data stored in VFO-B can be quickly used, for example by copying VFO-B to VFO-A (the `A<B` command), or by swapping the two VFOs (the `A<>B` command).

The transmitter spectrum. When transmitting, the RX panels are replaced by the spectrum of the transmitter. This has a fixed width of 24 kHz. It shows the TX signal as sent to the radio, unless PURESIGNAL is active and the monitor function (`MON` button in the `PS` menu) is active. In this case, provided that there is feedback circuitry, the transmitted signal as it shows up at the antenna is shown. If using `Duplex`, then the RX panels are kept while transmitting, and a small separate window containing the TX spectrum (only 6 kHz wide) pops up.

4.2 The `Hide` button

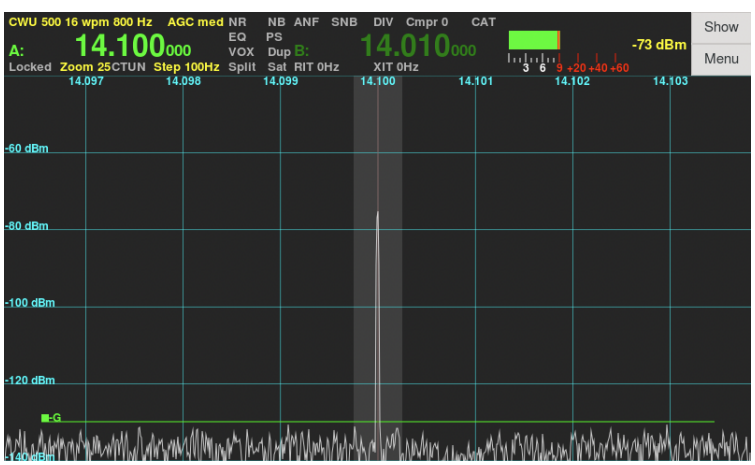

Fig. 4.2: piHPSDR window with the Toolbar and Slider area ,,hidden".

On small screens, space is scarce. This in in particular true for the vertical space if one used two RX panels and both with a spectrum scope and a waterfall. I this case, it may be hard to actually watch the signals if the

screen is small. This is where the `Hide` button comes in. Clicking on this button ,,hides" the toolbar and slider area:

The text on the button then changes to `Show`, and clicking this button again will then return to the previous display.

4.3 What is a spectrum?

The RX panadapter shows the RX spectrum, but what is a spectrum?. Some people get confused if the spectrum shows a noise floor at about −105 dBm, while their S-meter reading ,,without signal" is −100 dBm. As an example, see Fig. 4.3 where there is a calibrated noise floor with −130 dBm/Hz. These people then often ask how these figures are related and which the two figures tells them how much noise there is.

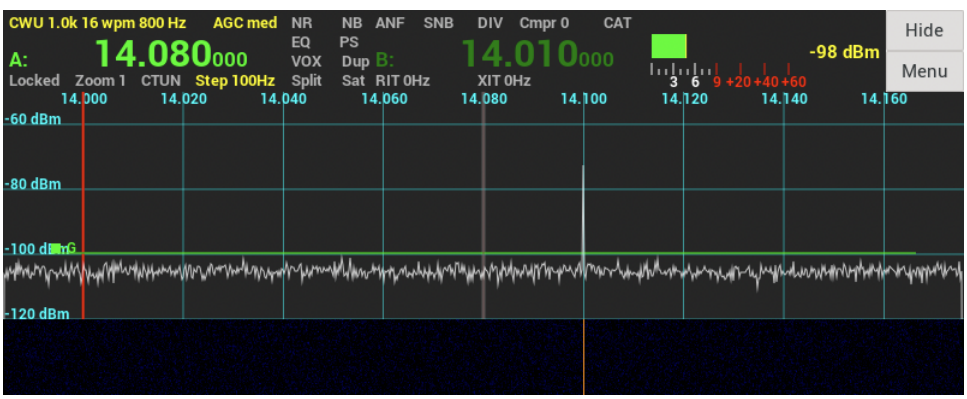

Fig. 4.3: piHPSDR showing a –73 dBm carrier and a noise floor of –130 dBM/Hz.

It is important to note that a spectrum plots an energy *density*, that is an energy (in dBm) per frequency interval. The RX panadapters in piHPSDR normalise the spectrum to a frequency interval that corresponds to the frequency width of one pixel on the screen. In Fig. 4.3 for example, the display is 800 pixels wide and the spectrum encompasses 192 kHz (the RX sample rate here), such that one pixel is 240 Hz wide. Multiplying the noise floor (–130 dBm/Hz) with 240 then gives -106 dBm, and this is indeed the position of the noise floor.

An immediate consequence of this is, that the noise floor moves down when

increasing the ZOOM value, since the frequency width of one pixel gets smaller. So the noise floor reading on the displays varies with display width and Zoom factor. The S-meter reading, on the other hand, integrates the spectral energy density over the filter passband. If you want to measure and compare the noise floor, the best you can do is to get an S-meter reading (in dBm) with a filter that is 1000 Hz wide, and then subtract 30 from the S-meter value to produce a noise floor figure in dBm/Hz. This is shown in Fig. 4.3 where the S-meter reading is about –100 dBm for a filter with of 1000 Hz. Since we have used a relatively wide filter here, this result is not much affected by the filter's shape factor.

Now it is demonstrated that this normalisation of the noise floor is what one actually wants. Consider an unmodulated carrier, e.g. a signal from a (calibrated) signal source. The S-meter should show the correct signal level, but the question is how tall the peak in the panadapter should be. Theoretically, such a peak is infinitely narrow and then must be infinitely high, since the signal level (in dBm) is encoded in the *area* below the peak rather than in its height. In practice, the signal has a natural (minimum) width, depending on the Fourier transformation used to generate the spectrum. The spectrum in the piHPSDR panadapter is normalised to this natural width, such that the height of the peak (roughly) matches the signal level.

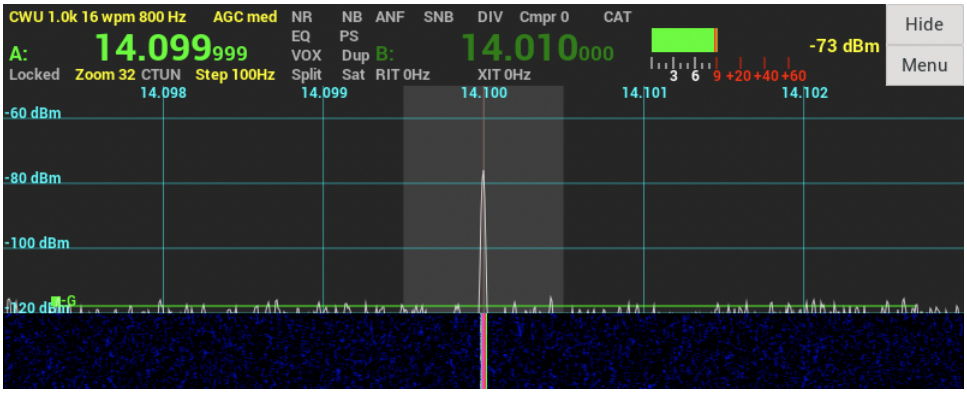

Fig. 4.4: The same as before, but with the Zoom slider moved to 32.

To demonstrate this, I have moved the Zoom slider from 1 to 32, such that Fig. 4.4 results. The noise floor has moved down to slightly less than –120 dBm as expected, since one pixel now only is 7.5 Hz wide. At the same time, the height of the signal peak remained at about –73 dBm. This demonstrates

that the normalisation of the spectrum as chosen in piHPSDR is such that the noise floor changes with screen width and/or Zoom factor, but that the height of narrow signals are not affected.

4.4 Spectrum scope options

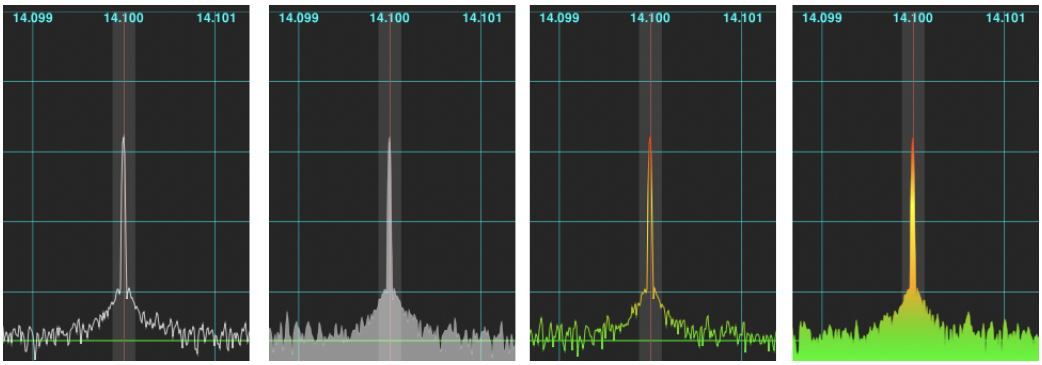

Fig. 4.5: Display options for the spectrum scope.

You have already seen two different spectrum scopes: in the first picture, the spectrum was a filled green area, while in the last picture, there only was a white line (this is similar to what you would see on a spectrum analyser). This can be adjusted to your personal preference in the `Display` menu (see below). There are two options which you can enable or disable, such that there are four different outcomes. The first option is the ,,Filled" option which discriminates between a line spectrum and a spectrum which is filled below the line. In Fig. 4.5, the first and third example have no filling, while the second and fourth spectrum are filled.

Then there is the ,,Gradient" option. Without this option, the spectrum is displayed in white colour. With the gradient option, the colour changes from green over yellow towards red depending on the signal strength (red colour is reached for S9). The above picture demonstrates the four possible combinations, and in the `Display` menu, you can make your choice. These settings can be made for both receivers separately. Note that the TX spectrum can be a line spectrum or can be filled (to be specified in the `TX` menu), but that there is no gradient option.

4.5 The Slider Area

Most sliders should be fairly self-explanatory, such as the one labelled `TX Drv` to adjust the TX RF output power, or `AF` to adjust the RX audio volume. All functions that can be mapped to a slider are listed in Sec. 11.2 on the `Sliders` menu. However, the `Zoom` and `Pan` sliders deserve some explanation.

Fig. 4.6: SSB on the 40m band with a large Zoom value.

The width of the RX spectrum equals the sample rate of the receiver. This means that if you use, say, a sample rate of 192 kHz for a receiver (as in Fig. 4.6, its spectrum will be 192 kHz wide. While it is sometimes convenient to have a wide spectrum on display, the part which is relevant to you may look a little bit compressed. This is where the `Zoom` command comes in. The zoom value can adopt integral values between 1 (no zoom) and 32. In the latter case, only 1/32 of the overall spectrum is displayed on the screen.

Fig. 4.6 shows a real-life example, taken while operating SSB on the 40m band with a sample rate of 192 kHz. Using a Zoom factor of 12, the width of the spectrum is only 16 kHz which makes it easy to spot the RX signal on the waterfall and set the RX frequency accordingly. Note the dial frequency (marked by a thin red line in the panadapter) is at the centre of the screen, the signal is to the left of that line because the lower side band is used on

the 40m band.

If only a small fraction of the whole spectrum is actually shown, this leads to the question *which* part is shown. In normal operation, the dial frequency is always at the centre of the screen so then one normally wants to see the central part of the spectrum. In CTUN mode on the other hand, one can receive signals from anywhere in the spectrum: the centre frequency is fixed and one moves the ,,RX window" around. In this case one may want to see a non-central part of a zoomed spectrum. This is where the Pan slider comes in. The Pan value (between −100 and 100) determines which part of a zoomed spectrum is actually displayed. For a Pan value of −100, the leftmost part is on display, and the rightmost part is displayed if the Pan value is 100. A Pan value of zero shows the central part of the spectrum. Note that the Pan value has no effect if there is no zoom, and that the Pan value is automatically adjusted when changing the Zoom value such that the RX frequency is at the centre of the display after changing Zoom. Normally you should not need the Pan slider if you are not using CTUN.

Whenever you change the Zoom or Pan value, the display of the spectrum is re-initialised. So it may take a while until enough data has been collected to be able to display the first spectrum after re-initialisation. This is especially true for small RX sample rates and large ZOOM values.

We will only shortly mention the other functions that can be assigned to sliders. For RX, you can adjust the AF gain, the AGC gain, the RF front end attenuation or gain, or the Squelch level. For TX, you can adjust the TX Drive (RF output power), the TX compressor, the Mic and Line-In gain, the VOX threshold, the CW speed and the lower cut of the RX panadapter (see Sec. 11.2). Together with the sliders for TX compression, Squelch and VOX, there is a small check-box to enable or disable that function.

Pop-up sliders

If using an external controller (GPIO, MIDI, G2 front panel, ANDROMEDA), one can assign piHPSDR functions to knobs (rotary encoders) of that controller. Just to give an example, suppose you can change the RF front end attenuation with such a knob. If you have a slider executing that function on-screen, it will auto-magically move when turning the knob, as to reflect the current value of the attenuation. But normally you do not need an ATT slider if you can change the attenuation with a knob and will not have it on-

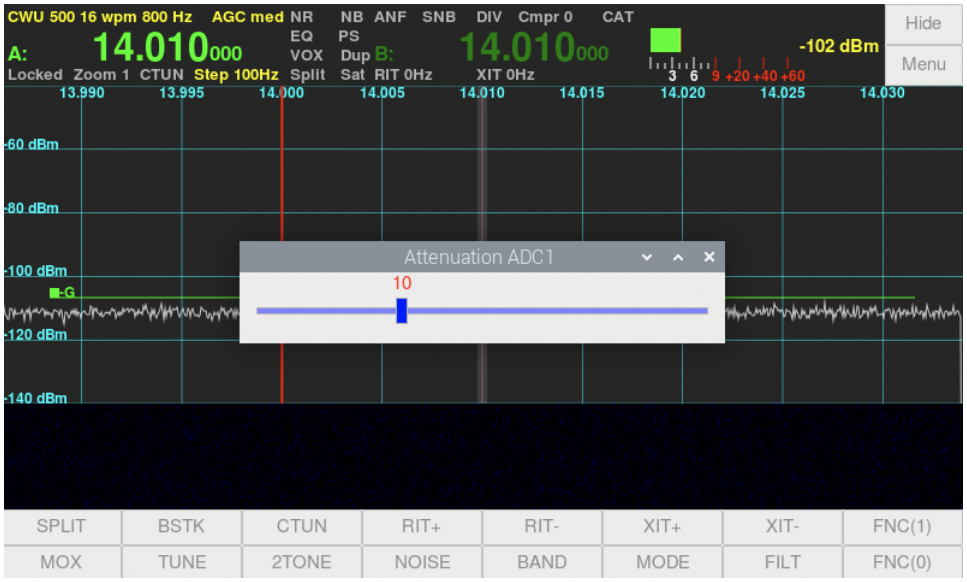

Fig. 4.7: A pop-up slider.

screen. In this case, a sliders pops up in the middle of the screen showing you the attenuation value you are selecting. This is shown in Fig. 4.7. No sliders are displayed and the attenuation is changed via a controller, and the pop-up slider moves while you turn the knob and lets you adjust the attenuation to a specific value, if you want.

In addition to all the function you can map onto sliders, pop-up sliders also appear if you apply temporary changes to filter edges while fighting QRM. For those functions that are shown in the VFO bar (e.g. TX compression), pop-up sliders are not needed and will not be shown.

4.6 The toolbars

At the bottom of the screen there are 0–3 toolbars. piHPSDR maintains six toolbars, each with 8 buttons. The rightmost button has a fixed function (see below) but any piHPSDR function trigger-able by a push-button (see Appendix A) can be assigned to any of the 42 (6*7) programmable buttons of the toolbars in the `Toolbar` menu (Sec. 11.1). The number of toolbars shown at the bottom of the screen (0–3) can be chosen in the `Screen` menu.

The rightmost toolbar button is labelled `FNC(x)` where x is a number from 0 to 5, indicating which of the six possible toolbars we have here. Pushing this buttons cycles that specific toolbar, so a toolbar showing `FNC(3)` will become one showing `FNC(4)`, etc. As a bonus for mouse users, a secondary click (usually a click with the ,,other" mouse button) cycles in the other direction.

If you are using and external controller (GPIO, MIDI, etc.) there are push-button assignments that bind the function of a button to the toolbar. A prominent example for such a controller is the first version of the piHPSDR GPIO controller, referred to as `Controller1` in this manual. In such a case, it is always the function of the toolbar displayed at the very bottom of the screen (if more than one toolbar is on display) that is executed.

4.7 Mouse clicks in the main window

The main window ,,accepts" mouse or touchscreen click events. Some of them come from the standard handlers of the GUI. It is clear, for example, that clicking the `Hide` or `Menu` buttons, as well as clicking one of the toolbar buttons, will activate the function associated with these buttons. Furthermore, the sliders (and the squelch enable/disable checkbox) in the sliders and Zoom/Pan are are operated as usual. But there are additional functions coded into piHPSDR:

If there are two receivers, a mouse click (press and release) into the panel of the non-active receiver makes it active. On the other hand, a mouse click in the panel of the active receiver changes the VFO frequency of that receiver to the value clicked on. This means, if you see a signal in the spectrum scope, click on that signal and your VFO will move (*jump*) to that signal.

The second option to change the VFO frequency of the active receiver is to click (and hold) into its panel, then drag the mouse to the left or to the right, and then release the button. This will shift the VFO frequency by the amount dragged, it makes no difference where the first click actually occurred, only the difference in horizontal position between click and release is used. You must drag at least three pixels so there is clear discrimination between a *VFO jump* (click then release) and a *VFO drag* (click, drag, and release) operation.

Especially if you are using a touch screen, it may be very difficult to click or drag exactly to the desired frequency. This is where the VFO snap option comes in (see the Radio menu). With this option enabled, the frequency chosen by a click or drag will be rounded to the nearest multiple of the VFO step size.

Finally, the VFO frequency of the active receiver can be changed by the scroll wheel of the mouse, if there is any. Using the scroll wheel lets the VFO frequency move in multiples of the VFO step size, while mouse dragging can also be used for finer tuning.

Clicking into the VFO bar opens the FREQ (VFO) menu, for the VFO-A if clicked into the left half of the bar, and for VFO-B if clicked into the right half. This menu not only offers the possibility for direct frequency entry, but also lets you alter the RIT/XIT or VFO step size, or alter the Lock, Duplex, CTUN, or Split states. So a simple click in the VFO bar gets you quick access to often-used functions.

Clicking in the meter section (between the VFO bar and the Hide/Menu buttons) opens the METER menu, where you can change the meter properties (see below).

When operating with a mouse, there are usually two mouse buttons, the primary button (for right-handed mouses, this is usually the left button) and a secondary one. Secondary mouse clicks are difficult to apply with a touch-screen. Although there are touch-screen drivers which convert long presses to secondary clicks, they generate, for a long press, a primary click first and a secondary one later, so it is not possible to generate a single ,,secondary press" event. But for the benefit of mouse users, secondary mouse clicks are handled in a special way:

A secondary click into the VFO bar will open the BAND menu, so a band change can be made with really few mouse clicks. Likewise, a secondary click into the panel of a receiver (no matter if it the active or the non-active one) will open the RX menu for that receiver. This can be used to change the settings of a non-active receiver without making it temporarily active. In the same way, a secondary click in the TX panel will open the TX menu.

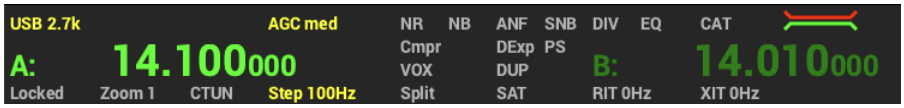

Fig. 4.8: The VFO bar

4.8 VFO bar and status indicators

Fig. 4.8 shows the VFO bar layout in more detail. The example shown is a VFO bar whose width is 745 pixels and thus suitable for screens that are 1024 pixels wide (or more), since the meter area has a fixed width of 200 pixels, and the `Hide`/`Menu` buttons are 65 pixels wide. This layout is denoted `VFO bar for 1024px windows`, as to the choice of VFO bar layouts, see the description of the `Screen` menu.

The large dials indicating the frequencies of VFO-A and VFO-B are easily recognised. The number to the left of the decimal point is the MHz part of the frequency, the three large digits to the right of the decimal point is the kHz part, and the last three (smaller) digits offer sub-kHz resolution. You may wonder why there is so much space to the left of the frequencies. This is so because with the advent of the QO-100 satellite, frequencies above 10 GHz can be used (with the transverter bands) and therefore eleven digits are needed!

Apart from the frequencies, you see a lot of text, most in light grey colour. As a general rule, a text in grey colour indicates a feature that is currently disabled, while features currently active are normally shown in yellow and sometimes in red.

At the top left corner of the VFO bar, the mode and filter width of the currently active receiver is displayed. Note that this is not necessarily the width of the filter which has been selected, if its width has been adjusted e.g. by using controller knobs. In Fig. 4.8, the text is `USB 2.7k` which indicates that the mode is USB and the filter width is about 2.7 kHz. Note the VFO bar reports the *actual* filter width and not the name of the chosen filter, the actual filter width can be temporarily modified for QRM fighting if one has e.g. a console or a penal with knobs assigned to the `Filter Cut High/Low` commands. For the CW (CWU and CWL) modes, the CW speed (in wpm) and the side tone frequency (in Hz) is stated as well. For CW, the filter size

may be appended by a ,,P", which indicates whether the CW audio peak filter (see the Filter menu) is effective on top of the normal filter. For the FMN mode, an indicator of the form C=xxx.y is added if CTCSS is enabled, and then xxx.y shows the CTCSS frequency.

Now we continue line by line, from left to right and find the string AGC med printed in yellow. This means that automatic gain control (AGC) is effective in the active receiver, and that the AGC time constant is intermediate. Possible values for the time constant are Long, Slow, Medium and Fast which can be selected in the AGC menu. Here one can also disable AGC, in this case the VFO bar shows AGC off in grey colour.

Continuing to the right, we see the noise reduction settings, all printed in grey (that is, they are not effective). This can be changed in the Noise menu. We have two different noise reduction capabilities NR1 and NR2, these strings are printed in yellow instead of the grey NR if they are effective. There are also two different noise-blankers NB1 and NB2, the automatic notch filter ANF and the spectral noise blanker SNB. Besides enabling/disabling these functions, there are further parameters you can tweak in the Noise menu.

The following items indicate whether Diversity reception is enabled or disabled (DIV), or whether an equaliser is effective EQ. Since there is a separate equaliser for both RX and the TX, the equaliser indicator, if it is effective, not only turns yellow but reads RXEQ while receiving and TXEQ while transmitting. This means, if only the TX equaliser is enabled, the indicator will show a grey EQ while receiving and a yellow TXEQ while transmitting.

The last indicator in the top row is CAT which indicates if the CAT module (see the CAT menu) has accepted at least one connection. In total, piHPSDR can be CAT-controlled simultaneously by five different sources, two of them using a serial line and three of them a TCP connection.

The indicators in the middle, between the VFO dials, are related to transmitting. CMPR indicates if a speech processor (compressor) is enabled, if so, it prints in yellow, followed by a number between 1 and 20 indicating the compression value in dB. If the speech processor is used simultaneously with the continuous frequency compressor (CFC), there is a yellow CprCfc at this place, and if the CFC is used but not the speech processor, the symbol is CFC on. Dexp states whether the downward expander (DEXP) is active. Due to space restrictions, this indicator is not shown for the smaller VFO bars.

PS indicates whether adaptive pre-distortion (,,PureSignal") is enabled, PS settings can be made in the PS menu. VOX indicates whether VOX (voice control) is enabled. VOX means that if the microphone delivers an amplitude above a certain threshold, the radio is automatically put into TX mode. Enabling/Disabling VOX and setting the correct threshold can be done in the VOX menu. Finally, DUP indicates whether duplex mode is active. In duplex mode, the receiver(s) continue to work during transmit. Duplex mode when using the same antenna for RX and TX is no fun: you not only hear your own signal with a delay (from the cross-talk at the TRX relay), but this cross-talk signal is usually so strong that it leads to ,,AGC pumping", so your receiver is virtually deaf during the first second after TX/RX transition. For satellite operation, on the other hand, duplex mode is very convenient. Here you usually have two separate and well-decoupled antennas for RX and TX.

The bottom line of the VFO bar indicators are related to the VFO status. If the Locked string is red, it indicates that the VFO is locked and will not accept changes. There is a LOCK command which toggles the LOCK status and which can be assigned to a toolbar button or a push-button on a GPIO or MIDI console, but the Lock status can also be set/unset via the FREQ menu, accessibly by the main menu window, or just by clicking into the VFO bar.

The next indicator in the bottom line indicates the Zoom factor. If the Zoom factor is 1, the indicator is grey, otherwise it is yellow and also indicates the factor. Then there is a string CTUN which indicates whether the CTUN (,,click to tune") mode is off or on (the string is yellow in the latter case). The step size of the VFO controlling the active receiver is displayed next, this string is always yellow.

The split status is displayed by the next indicator, which is red in split mode. If split mode is off, transmitting is done on the frequency and the mode of the active receiver (if there are two receivers), or on the frequency/mode of VFO-A (if there is only one receiver). If split mode is on, transmitting occurs on the frequency/mode of the non-active receiver (if there are 2) or on VFO-B (if there is only one receiver). Note that a frequent beginner error is to have, say, SSB mode in VFO-A and CW mode in VFO-B. In this case, nothing is transmitted when doing split and using the microphone. Normally one starts with the A>B command (that is, copy VFO-A to VFO-B), and then adjusts the transmit frequency.

The next indicator shows the SAT (satellite) mode, which can be off (then the indicator reads SAT in grey), or which can be SAT or RSAT (then the indicator displays this string). Once SAT mode is engaged, the two VFOs are tied together such that any frequency change of one of the two VFOs also applies to the other VFO. This is the best way to do cross-band operation with, e.g. the QO-100 satellite which is at a fixed position. In RSAT mode, a frequency change of one of the VFOs is applied to the other VFO with an opposite sign (so if you move up VFO-A by 2 kHz, then VFO-B moves down by the same amount). This is what one needs for low-flying satellites which have inverting transponders which offer some sort of Doppler correction.

Finally there are the RIT (receiver incremental tuning) and XIT (transmitter incremental tuning) indicators. If RIT is off, receiving occurs on the VFO dial frequency. If RIT is on, the indicator becomes yellow and also indicates the RIT offset, that is, the frequency offset used while receiving. RIT is used, for example, if during your CW QSO the frequency of the transmitter of your QSO partner drifts and you want to follow without altering the frequency of your own transmitted signal. The RIT indicator corresponds to the active receiver. If XIT is active, the indicator becomes yellow and shows the offset of the true TX frequency from the VFO dial frequency.

Finally, in the top right corner you see a symbol with a green and a red line which show the actual RX filter edges (in red) compared to the nominal filter edges (in green). Both filter edges normally agree unless the actual filter edge has been temporarily modified, e.g. to cope with QRM.

4.9 Meter section

Fig. 4.9 shows the different designs that exist for the meter. To the left (right) there are the digital (analog) meters, while the top panels show the meter during RX and the lower panels during TX.

By default, the width of the meter section is 200 pixels, but if the current choice of the VFO bar and the window width leaves enough space, the width of the meter is increased to 250 pixels which allows for displaying some information at a larger font size. For example, if the standard VFO bar for a piHPSDR window 800 pixels wide is chosen, and the width of the window exceeds 850 pixels, the meter automatically widens.

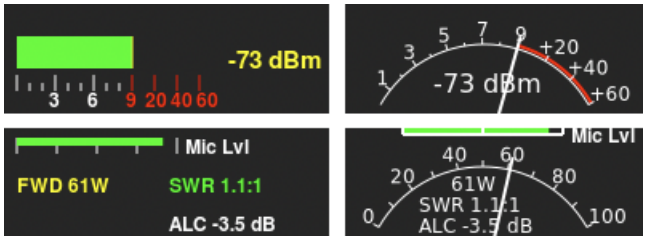

Fig. 4.9: Different designs for the meter.

The design can be switched between digital and analog in the `Meter` menu, which can be accessed quickly just by clicking into the meter area. During RX, an S-meter is shown together with the signal level in dBm. Note that −73 dBm corresponds to S9 for frequencies up to 30 MHz, while above 30 MHz, S9 corresponds to −93 dBm. Normally, one step on the S meter amounts to 6 dB, then a signal level of S1 corresponds to −121 dBm below 30 MHz and to −141 dBm above.. It is possible to choose 3 dB steps in the `Radio`, in this case an S1 signal level indicates −102 dBm below 30 MHz and −122 dBm above. The 3 dB step option has been added because some major hardware producers use such steps in their gear (thus violating a long-standing IARU recommendation).

In the meter menu, one can also choose if the S-meter reading corresponds to a peak or an average value. Further info on the meters (e.g. switching between ,,peak" and ,,average" reporting) is described in the `Meter` menu. During TX, the output power is displayed, provided that the radio actually reports this power. The output power meter can be calibrated (see the `PA` menu). If the SWR exceeds a threshold for SWR warnings (the default is 1:3, but this can be changed in the `TX` menu), the SWR indicator turns red. If, in addition, SWR protection is enabled in the `TX` menu, the output drive will be reduced to zero if the SWR exceed that threshold. Furthermore, the ALC (automatic level control) value of the transmitter is shown. Negative ALC values (in peak mode) indicate that the volume of the TX input audio could be increased to get full output power. During TX (and also during RX when using voice control (VOX), the actual level of the TX audio signal is shown through the small `Mic Lvl` bar. The audio level is in dB from −40 . . . 0 dB. Note that this refers to the peak level of the incoming "TX microphone" audio data, there can be further amplification in the TX chain through the `Mic` slider. All this is explained in detail in chapter 15 on the TX audio chain.

Chapter 5

piHPSDR menus: introduction

5.1 piHPSDR menus

Now we have a series of chapters that discuss all the piHPSDR menus. Many menus can be opened by a button click (or a push-button on an external controller), e.g. hitting the `MODE`, `FILT`, or `NOISE` button on the toolbar you have seen in the last picture. You already know that the VFO and Meter menus can be opened by clicking into the VFO or meter section at the top of the window. When operating with a mouse, a secondary click in the RX or TX pan-adapter opens the RX or TX menu.

Some remarks have to be made about menus in general. Since piHPSDR is optimised for working with small screens, only one menu can be open at a time. If a menu is open and one tries to open another one, the first menu will be destroyed (closed) and the new one will be opened. For example, if you hit the `FILT` button in the toolbar when starting from Fig. 5.1, the main menu closes and the `Filter` menu opens. If you try to open a menu that is already open, then the menu will be closed. So, starting from Fig. 5.1 hitting the `Menu` button again will close the menu. Likewise, when the Filter menu has been opened, either via the Main Menu or with the `FILT` button, then hitting this button again will close the `Filter` menu.

While the menus are looking quite diverse, some effort has been invested to keep some things consistent throughout. For example, at the top left corner of the menu you usually find the "Close" button which closes the menu.

The close button is somewhat emphasised (slightly larger letters, and a thin border) so you will always quickly find it. Of course, it it possible to close a menu by deleting the menu window (on RaspberryPi, this is the small cross at the left of the title bar) but this is neither necessary nor recommended.

5.2 Using menus *and* controllers/panels

Menus in piHPSDR are windows that pop up if a menu is opened. These windows by default are positioned such that they cover a relevant part of the piHPSDR window. If you have a large display, you can move the menu window such that it no longer covers the panadapter(s), if you want.

If a menu window pops open, its controls reflect the status of piHPSDR when the menu opens. One can then operate the menu, usually with a point-and-click device such as a mouse, a trackpad, or a touch screen.

If once changes the state of piHPSDR this way, the change is immediately reflected in the menu. For example, if you open the `Radio` menu and the split state is not active, the split checkbox in the radio menu (see Fig. 6.1) is not checked. If you then click this check box, it changes to the "checked" state and the Split state becomes active, as can be seen e.g. in the VFO bar. It is also possible to change the Split state by assigning the `Split` command to a toolbar button, or to a physical button of an external controller (GPIO, MIDI, ANDROMEDA) or a G2 built-in panel. If you do this while the `Radio` menu is open, its Split check-box will not change although the split state changed, as witnessed by the VFO bar.

This example demonstrates that as soon as one has opened a menu, one should use the point-and-click device to make adjustments and then close it, before continuing operating piHPSDR using a controller or a panel. While piHPSDR does not block the controller or the panel once a menu has popped open, using the panel while a menu is open is not supported. In most cases, piHPSDR will behave as expected, but if you change a piHPSDR setting with a controller or a panel while a menu is open that lets you change just that setting, the behaviour may be unexpected or even undetermined.

5.3 The main piHPSDR menu

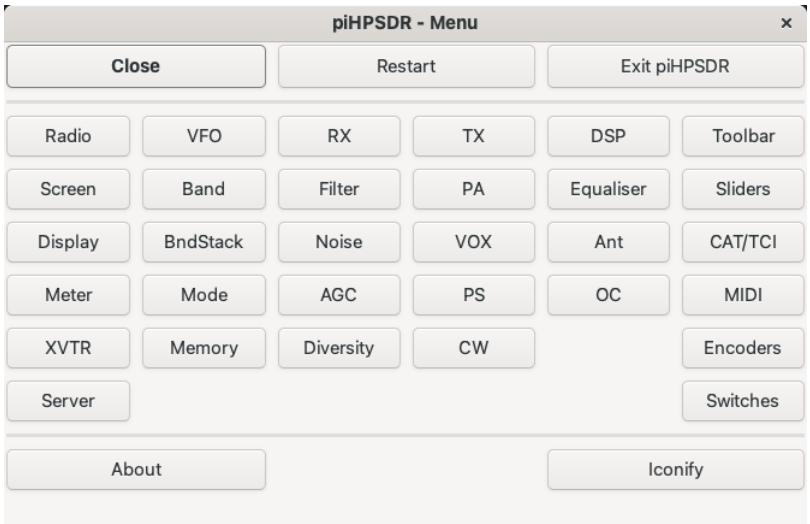

Fig. 5.1: The Main men, opened by the `Menu` button.

There is one place from which *all* piHPSDR menus are at hand, and this is the "Main Menu". It can be opened by clicking into the `Menu` button at the top right corner of the piHPSDR window, the outcome is shown in Fig. 5.1. It is also possible to open the main menu just by hitting 'm' or 'M' on the keyboard. In the Main menu, there are some commands available here that do not directly affect the radio operation, so these commands are found in the top and bottom line of the Main Menu. We first mention the `Restart` button in the middle of the top line. This restarts the radio protocol. While not needed under normal circumstances, it my happen (especially with beta releases of radio FPGA firmware) that the data exchange between piHPSDR and the radio gets out-of-sync. I observed such problems with early versions of the Protocol-2 firmware for Orion2 boards and that is the reason the `Restart` button is there, since this made a quick recovery possible without loosing the QSO. At the bottom right, there is the `Iconify` button which ,,minimises" the piHPSDR window. Normally, if needed, one can do so by standard methods of the operating system in the title bar of the piHPSDR window. If piHPSDR, however, runs in full-screen mode (this is the case on very small touch screens), then the `Iconify` button to make the piHPSDR window temporarily disappear without breaking the connection to the radio,

do some work with the operating system, and get the piHPSDR window back. Note in earlier versions of piHPSDR this function was associated with the "Hide" button in the top right corner of the main window. Then, there are two menus (`Exit` and `About`) which are described in due course and which one can open by clicking either `Exit piHPSDR` or `About` in the main menu.

The other buttons, between the two horizontal separator lines, give access to piHPSDR control and fine tuning. They are organised in six columns, namely radio related menus (first column), VFO related menus (second column), RX and TX related menus (third and fourth column), menus affecting both RX and TX (fifth column) and, finally, menus for adjusting how you can control piHPSDR (sixth column), either via Toolbar, MIDI, GPIO encoders/switches or the new G2-ultra front panel.

Not all menus appear in all cases: For example, the `Encoders` and `Switches` menus shown in Fig. 5.1 are only present if piHPSDR is controlled by a GPIO-based controller. If, instead, a G2-Ultra type front panel is detected, a menu `G2panel` appears in that place which lets you assign non-default functions to buttons and encoders of the front panel. If there is neither a GPIO controller nor a G2-Ultra front panel, the last column of the main menu ends with the `MIDI` menu.

Whenever one of the sub-menus is closed, /pH writes its settings to the props file. This is not done automatically (that is, periodically) because on some systems I/O operations affect network traffic and may cause audio cracks. So restricting this command to situations where ,,much happens" anyway in the GUI seems a good compromise.

5.4 The `Exit` Menu

Fig. 5.2: The `Exit` menu.

Via the `Exit` menu, you can leave the piHPSDR program. When leaving the

program, the radio protocol is stopped and all the settings are written to a preferences file. This file is located in the piHPSDR directory and takes the name XX-XX-XX-XX-XX-XX.props, where the XX are hexadecimal numbers that encode the MAC address for the radio. So the preferences for different radios (if you have more than one) are stored in different files. Note that the preferences are not only written to file when leaving the program, but also each time a menu is opened, so the preferences file should reflect the changes you made even in the case the piHPSDR has crashed for whatever reason.

To leave the program, just click the "Exit" button in this menu. If you decide you want to continue, you can leave the Exit menu by clicking the "Cancel" button. This is the button which closes the menu and has the same position and look as the "Close" buttons in all the other menus.

If piHPSDR runs with administrator privileges, you can even leave the program and either re-boot or switch off the computer via the "Reboot" and "Shutdown" buttons. This makes sense for setups where a Raspberry Pi running piHPSDR, a small SDR radio, a touch-screen and several encoders and switches are built into a single common enclosure. On the other hand, when running piHPSDR on desktop or laptop computers, clicking "Reboot" or "Shutdown" both leave the piHPSDR program but no re-boot or shutdown takes place, due to missing administrator privileges.

5.5 The About Menu

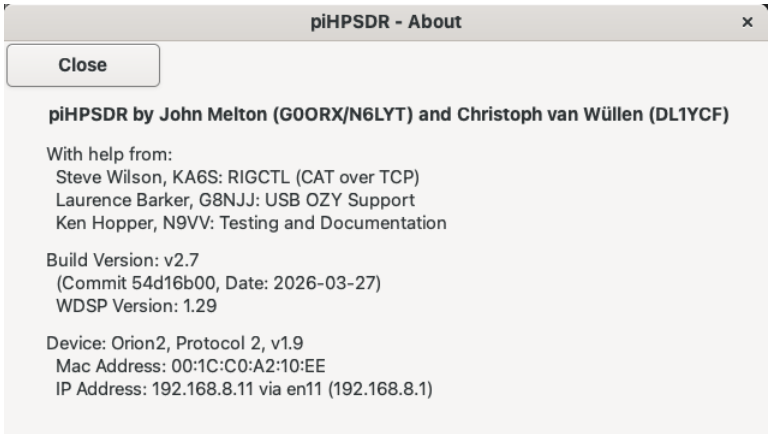

Fig. 5.3: The About menu.

The about menu gives you some information about piHPSDR, first the original author, John Melton, and an (incomplete) list of persons who contributed to the code, and then a statement which version of piHPSDR is working here, and the ID string and the data of the latest commit (change) applied to the repository from which piHPSDR was built. Here you also find the version number of the WDSP library which is the ,,engine" running under the hood, and which does nearly all of the signal processing. If you file a bug report, this is very important information so you should always include a screen shot of the About menu when reporting problems. Of particular importance is the so-called commit, this hexadecimal number (here: b5e780d0) identifies the exact status of the source code repository when the program has been compiled. If a string `-dirty` is appended to the version number (here: v2.6, that is, *not* dirty) this means that one or more files have been locally modified and do not match the commit, and problem reports from a ,,dirty" version are difficult to handle since it is not documented what has been changed locally.

Finally, there is some data on the radio, namely the device type and version numbers, and which protocol is running. For diagnostic purposes, you also see the MAC address of the radio, its IP address (in the example shown, 192.168.8.2) and the IP address of the computer running piHPSDR (here 192.168.8.3, this is the IP address of the network interface that actually does the connection with the radio). The radio's IP and MAC address are also

displayed in the piHPSDR main window title bar, the MAC address is of interest since the radio-specific preferences are stored in a file whose name derived from the MAC address of the radio, replacing the colons by dashes and appending ,,.props". This means you can have separate sets of settings for different radios.

Chapter 6

The Main Menu: Radio-related menus

6.1 The Radio Menu

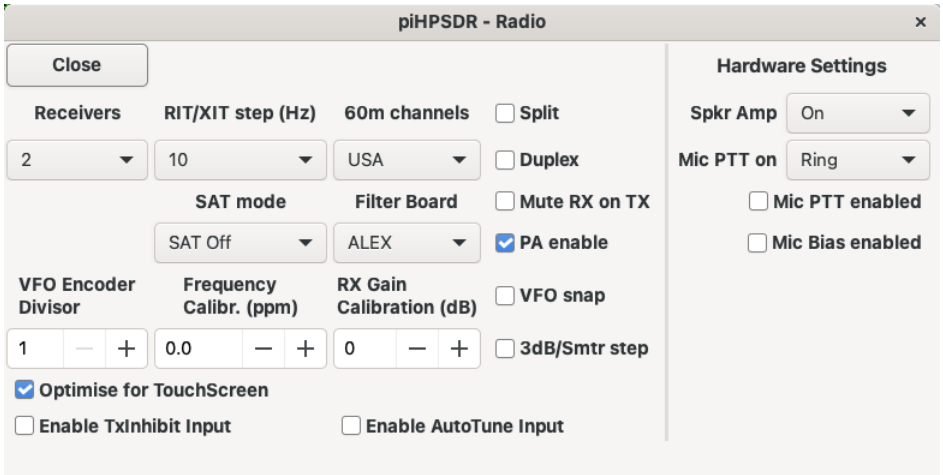

Fig. 6.1: The Radio menu.

The Radio menu lets you make settings which affect the general setting, and the hardware of the radio. The following figure (Fig. 6.1) shows the Radio menu as it appears on my Anan-7000. In the rightmost part of the panel (separated by a thin vertical line), there are hardware-specific settings,

this part will differ between different radio models and this will be discussed below. First we go through all the elements we see in in the left part of Fig. 6.1.

6.1.1 General controls

`Receivers`. In the pop-down menu (GTK combo-box) below this string, you can select the number of receivers that are running (well, you can choose between 1 and 2). When the number of receivers change, the radio communication will shortly be stopped and then resumed, so do not be surprised if the spectrum scope freezes for a second or so.

`RIT/XIT step`. In the pop-down menu you can choose among three (1 Hz, 10 Hz, 100 Hz) RIT step sizes for the VFO of the active receiver. For example, if the RIT step is 10 Hz, then you can change the RIT offset in steps of 10 Hz with the RIT+ or RIT- buttons in the toolbar or a push-button, or with an encoder assigned to the `RIT` command. Note that for the `XIT+` and `XIT-` buttons, the XIT offset changes by 10 times the RIT step size. The RIT/XIT step size is part of the RX/TX profile (see chapter 14) and therefore it is automatically updated if you change the mode.

`60m channels`. When operating in the 60m band, the allowed frequency ranges have a reddish colour in the RX panadapter. Possible choices are `USA`, `UK`, and `WRC15`, The `USA` and `UK` choices implement the channel structure of the 60m band according to the regulations valid in the USA and Great Britain. `WRC15` gives you a small (15 kHz wide) 60m band according to the WRC15 (World Radio Conference 2015) document, which is now implemented in many countries. Note that since the end of 2025, FCC regulations have changed such that one of the five U.S. channels has been replaced by a band with regulations very similar to WRC15. To my knowledge, WRC15 allocation is still not in effect in the U.K. Given to this fairly complicated regulatory situation, please inform yourself which regulations apply in your country!

`Split`. (Only shown on radios with a transmitter) Use this checkbox to enable/disable Split mode. In Split mode, the frequency of the non-active receiver (when using two receivers) or the frequency of VFO-B (when using one receiver) controls the TX frequency. In normal (non-Split) mode, it is the frequency of the active receiver (2 RX) or the frequency of VFO-A (1

RX) that matters. Note that a frequent beginner error is to have, say, SSB mode in VFO-A and CW mode in VFO-B. In this case, nothing is transmitted when doing split and using the microphone. Normally one starts with the A>B command (that is, copy VFO-A to VFO-B), and then adjusts the transmit frequency.

Duplex. (Only shown on radios with a transmitter) Use this checkbox to enable/disable Duplex mode. In Duplex mode, the receiver(s) continue working during TX. In normal setup, this is detrimental since the very strong signal that originates from the cross-talk at the T/R relay will lead to AGC pumping, making your receiver(s) essentially deaf for a short period after TX/RX switching. However, when using different and well-decoupled antennas for RX and TX (this is typical for some satellite operations), Duplex mode gives you important information, as you can see your own down-link signal. In contrast to what is often stated, Duplex mode does not affect the data stream between the computer and the radio, it *only* determines whether the receivers (within the WDSP library) are shut down during transmit or not.

Sample Rate. This box (to the left of the SAT mode controls) does not appear for HPSDR radios running protocol-2, since there each receiver has its own sample rate that is therefore set in the RX menu (chapter 8.1). For HPSDR radios running protocol-1, the available sample rates are between 48 and 384 kHz. For SoapySDR radios the sample rate can usually chosen between 48 kHz and the native sample rate of the radio in steps of two, but little is gained choosing a sample rate that is below the native sample rate because the downsampling is done within piHPSDR.

SAT mode. Here you can choose between `SAT off`, `SAT`, and `RSAT`. In SAT mode, frequency moves applied to one of the two VFOs are applied to the other VFO as well. This is convenient for cross-band operation over satellites with (normal) linear transponders. In RSAT mode, frequency moves applied to one of the two VFOs are applied to the other with the sign inverted, that is, if for example you move the frequency of VFO A up by 3 kHz, the frequency of VFO B moves down by the same amount. This is convenient for cross-band operation over satellites with inverted transponders. Inverted transponders are sometimes find in low and fast moving satellites because this leads to some Doppler correction.

Filter board. Normally SDRs have some sort of built-in PA with a filter board. Filters in the TX path between the PA and the antenna are always

required, and filters in the RX path provide some protection against ADC overloads from strong out-of-band signals. Here you can choose between NONE, ALEX, APOLLO, CHARLY25, and N2ADR. Choose NONE if none of the other cases apply, and hope your radio does things right automatically. ALEX is the most frequent choice and applies to the largest part of current HPSDR radios. APOLLO is an early design of a PA/filter combination for Hermes boards, choose this if you have one. CHARLY25 is a filter board used in some RedPitaya based radios (STEMlab and HAMlab). If you choose this, the Attenuator slider will disappear from the Slider area (because this design does not have a step attenuator), instead, you get a combined Attenuator/Preamp check-box which lets you choose between zero, preamp values of 18 and 36 dB, and attenuation values of 12, 24, and 36 dB. N2ADR, finally, is the filter board usually used in combination with a HermesLite-II radio. It is controlled by the OC (open collector) bits in the HPSDR protocol. This means if you use N2ADR, this will override your OC settings upon program startup. It is possible to change the OC settings in the OC menu, and these settings are saved with the preferences. Upon next program start, however, these preferences will again be overwritten as long as the N2ADR filter board is chosen. This means that if you want to use the N2ADR board with non-standard settings, you have to choose NONE for the filter board and make your settings in the OC menu!

Mute RX when TX. (Only shown on radios with a transmitter) Normally, the receivers are shut down while transmitting, except in duplex mode. This option (checkbox) mutes the RX audio while transmitting even if running Duplex. It is important to note that the RX continue to work, so you can see the signals on the RX panel, the S-meter works, etc. This option is largely equivalent to moving the AF slider to the minimum position while transmitting.

PA enable. (Only shown on HPSDR radios with a transmitter) This enables/disables the PA in the radio. In addition to this global flag, there is a per-band PA enable option for the transverter bands (see the XVTR menu).

VFO Encoder Divisor. This control applies to VFO knobs in front panels, GPIO or MIDI controllers. In some cases, these encoders generate too many ticks per revolution, such that it is difficult to fine tune on a signal. If the VFO Encoder Divisor, as shown in the example, has a value of 15, only every 15th tick will we processed. This value is often used with optical encoders

found in some of the GPIO controllers. For the default value of 1, the divisor has no effect.

Frequency Calibr. (ppm). Here you can apply a small correction to the frequencies. Typically, all frequencies are derived from a common clock, so any correction must be *relative*. For example, if you choose a value of 3.0 here, then the TX/RX frequencies at 7 MHz will be increased by 21 Hz, but this increase is four times larger at 28 MHz. Note that for transverter bands, adjusting the frequency is better done by adjusting the local oscillator frequency (see Sec. 6.5).

RX Gain Calibration Here you can calibrate your RF front end. To this end, you need a highly accurate signal source of, say, −73 dBm. Connect this source to your radio and tune on the signal. If the signal reported by the meter appears too strong (say, at −65 dBm), *decrease* the calibration value with the spin button until the meter shows the correct level. The RX gain calibration value can be viewed as the amplification/attenuation of a virtual device you would need in your RF front end. Therefore, you need a negative calibration value (attenuation) if the signal in the meter is too strong. For most HPSDR radios, the default value is zero. For the HermesLite-II and other radios using the AD9866 chip, the default value is 14, but this is a rough estimate and certainly depends on details of the RF front end.

VFO snap. If this box is checked, VFO frequency changes by clicking or dragging in the RX panadapter are finally rounded to the nearest multiple of the current VFO step size. This option is mostly meant for touch-screens which are usually not very accurate. Note that if this option is checked, you may not be able to exactly tune on a signal by clicking on it with a mouse.

3dB/Smtr step. There is an IARU recommendation that 1 step on the S-meter corresponds to difference in signal level of 6 dB. While the recommendation for the S9 level (−73 dBm for frequencies up to 30 MHz, and −93 dBm for higher frequencies) is well respected, some major amateur radio hardware producers have S-meters where one S-meter step corresponds to a 3 dB signal level difference. piHPSDR normally follows the IARU recommendation but if this box is checked, 3 dB steps are used in the S meter scale, that is, −73 dBm is shown as S9 and −76 dBm as S8, etc.

Optimise for TouchScreen. The normal procedure to make a selection from a pop-down menu (such as the Receivers button on this screen) is to

click (and hold) it with a mouse, then drag the mouse to your choice, and then the selection is made by releasing the mouse button. This is very difficult to achieve on a touch screen. Therefore, if `Optimise for TouchScreen` is checked, the behaviour of pop-down menus is modified as follows: You click and release on the menu button, then it pops down and stays open. Then you make your selection by a second click/release sequence on your choice. While this is (only a little bit) more involved than the normal procedure when using a mouse, it is a great helper when using a touch screen. Therefore this option is set by default, but you can uncheck it here if you prefer normal mouse operation. Note that this option becomes effective when the next menu is opened.

The controls `Enable TxInhibit Input` and `Enable AutoTune input` in the next row only appear for HPSDR (and not for SoapySDR) radios, since it assigns actions to HPSDR user input bits.

`Enable TxInhibit Input`. For protocol 1, TxInhibit is signalled for most radios by the Hermes IO1 bit being cleared (the Hermes IO2 bit is used for Anan-7000/8000). For protocol 2, for most radios the IO4 bit is used (Anan-7000/8000/G2 use the IO5 bit). If the `Enable TxInhibit Input` box is checked, ,,TX Inhibit" will be drawn in the top left corner of the RX1 pan-adapter if signalled. If TxInhibit is signalled while piHPSDR is transmitting, it will induce a TX/RX transition, and any RX/TX transition is suppressed while TxInhibit is active. Some radios (e.g. Anan-7000) have a RCA jack with an active-low input, and the TxInhibit bit follows that input. Note that for effective hardware protection, processing the TxInhibit bit must take place in the FPGA of the radio. This checkbox simply enables piHPSDR to tell the user about such an event.

`Enable AutoTune input`. his box only appears for HPSDR (and not for SoapySDR) radios. The AutoTune state is signalled for protocol 1 by the input bit IO3, and for protocol 2 by the user input bit IO6 (the active state is represented by the bit being cleared). If `Enable AutoTune input` checked, piHPSDR will initiate `TUNE`-ing if the input bit becomes active, and stop `TUNE`-ing if it later on becomes inactive. For the Anan-7000, there is no RCA jack connected with the IO3/IO6 bit, but the input is available on the spread-out board (between the Orion-II and the PA board) and one can easily solder a wire there to have this user input. The AutoTune input is meant to be pulled down if e.g. a "Tune" button on an external automatic tuner is

pressed. For such a setup, it is usually recommended to use a reduced RF output level while TUNE-ing (see the TX menu, chapter 9.1.1).

6.1.2 Hardware specific settings

In the right part of the Radio menu, a large number of settings may appear, but only those for the actual radio are shown.

Microphone/Speaker settings

Most of these sections are relevant for an Anana-7000 and thus shown in Fig. 1. These settings are also valid if piHPSDR is running on a controller with an audio codec (Controller3). The controls are

Spkr Amp. For OrionII and Saturn radios, the headphone output is driven directly by the audio codec chip, but additionally there is an AF amplifier that drives the speaker jacks and possibly built-in speakers. The possible choices in this pop-down menu are On (speaker amp switched on all the time), Mute on Tx (speaker amp switched off only while transmitting) and Off (speaker amp switched off all the time). Note that the option Mute on TX is generally *not recommended*, because the speakers may emit an audible "plop" each time the speaker amplifier is muted or un-muted. Some users have reported RFI problems with the speakers (but not with the headphone) and this option has been added to make it possible to mute the speakers during transmit only.

Mic PTT on This chooses the wiring of the 3.5 mm TRS microphone jack. The possible choices are Ring and Tip. For Ring, the ring connector is used for PTT, while the top connector is used for microphone input (and possibly microphone bias). For Tip, it is the other way round (PTT on tip, Mic/Bias on ring).

Mic PTT enabled This box enables the PTT contact on the front panel microphone jack. This may be either the ring or the tip contact, depending on the choice made with Mic PTT on.

Mic Bias enabled This box enables a bias on the contact of the front panel microphone jack. This may be either the ring or the tip contact, depending on the choice made with Mic PTT on. A bias is needed for electret microphones,

for dynamic microphones one should usually disable the bias.

`Mic Input`. This is only shown for Anan G2 radios (Saturn boards). These radios have two jacks for connecting a microphone, either a 3.5mm TRS jack in the front panel, or an XLR connection in the back panel. The pop-down menu lets you choose between these two options.

ATLAS bus systems

Lots of additional controls are required for ATLAS (first-generation HPSDR) systems, so we show the `Radio` menu for such systems in Fig. 6.2. They are only shown if the radio identifies itself as a METIS or OYZ device. A characteristic of this bus system is that each receiver and the transmitter are built as individual cards that can be plugged into a bus system. Since these radios usually run protocol-2, the `Sample Rate` control appears in the left part of the menu.

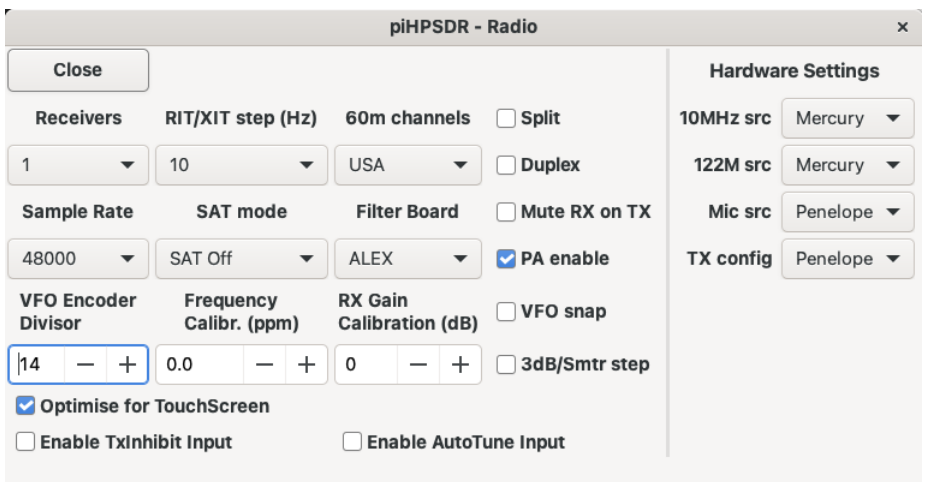

Fig. 6.2: `Radio` menu settings for ATLAS bus systems

Note that I have no access to such legacy radios, so the piHPSDR code to handle these radios is partly built on speculation (that is, studying the specs) and exchanging e-mails with people who still run such hardware. If you meet any inconsistencies, please contact the author.

`10MHz src` This selects the 10 MHz master clock, which can be either `ATLAS`

(the bus itself is the source), Penelope (the transmitter board is the source), or Mercury (the receiver board is the source).

122M src This selects the 122.88 MHz master clock, which can be either Penelope or Mercury.

Mic src This selects where the microphone samples that are sent to the computer originate, that is, where your microphone has to be connected. The default is Penelope, this means the microphone is connected to the transmitter board. The other choice is Janus. The Janus board is simply an ADC/DAC board (not a radio) and used in some very early setups.

TX config This indicates which transmitter board is present on the bus. It can be No TX, if this is a receive-only radio, Penelope or Pennylane. The Pennylane is a later version of the Penelope transmitter board, the essential difference is that it can control the output signal level. In the Penelope case, piHPSDR will scale the IQ samples to provide TX drive control.

Janus Only. This box (not shown in Fig. 6.2) is for ATLAS systems that only have a Janus ADC/DAC card, and will only appear for OZY (USB-connected) boards. While this hardware is not a radio, external hardware such as the SDR-1000 can be connected to the Janus card. If this option is checked, piHPSDR assumes that the radio is controlled outside piHPSDR, and will thus only process the data stream but not try to send any commands to the radio.

Anan radios up to the 200D

For these radios, there are very few settings to be made:

Anan-10E/100B. (Checkbox, only shown on Hermes boards) While the Anan-10E and the Anan-100B identify themselves as Hermes boards, they differ from standard Hermes boards since they have a FPGA with limited resources, and this affects the allocation of PureSignal feedback channels. To make PureSignal work on these machines, you have to check this box in the Radio menu. This also applies to the new ,,revised" version of the Anan-10E that came out in 2025. Note that checking this box for standard Hermes boards will prevent PureSignal from working.

New PA board. (Checkbox, only shown on Hermes/Angelia/Orion boards) This control is relevant for the Anan-100/200 radio family, where the PA

board has been revised such that there is an ,,old" (produced up to about February 2015) and a ,,new" PA board (for radios produced after Feb. 2015). Among other differences, it seems that `Bypass` jack in the rear panel is an output to be connected with either EXT1 or EXT2 for the PURESIGNAL feedback path, while for the new board, `Bypass` is an input preferably used for that feedback.

HermesLite-II

There are only three checkboxes in the Hardware Settings for the HermesLite-II. Note that in addition, the `N2ADR` filter board is chosen by default for this radio, and the RX gain calibration defaults to 14 dB.

`HL2 audio codec`. Check this box if you have the AK4951 companion board installed. piHPSDR then activates the audio codec on that board and includes RX audio samples in the data stream to the HL2.

`HL2 CL1/2`. The HL2 has two SMA jacks denoted CL1 and CL2 in the front panel. When checking this box, you can feed a 10 MHz reference clock to the (input) CL1 jack, and the (output) Cl2 jack then has an internally re-generated 10 Mhz output signal with about 3.3 V_{pp}.

`HL2 AH4 ATU`. The standard HL2 firmware includes support for an Icom AH4 automatic antenna tuner. As a consequence, when one starts TUNE-ing, the HL2 stops RF output and first looks for the presence of the tuner and then activates it. If no such tuner, or another tuner e.g. controlled by the HL2 IO-Board, is connected, this interruption of the RF output may have unwanted side effects. So the standard HL2 tuner support will only be active if this box is checked, while the HL2 IO-Board tuner register is written upon TUNE-ing if this box is un-checked.

SoapySDR

Few check boxes appear in the `Radio` menu that are specific for SoapySDR radios:

`Swap IQ` This box only appears for radios connected via the SoapySDR library. If checked the I and Q samples are exchanged, both in the receivers and in the transmitter. An indication that this is necessary is if you see

signals with a frequency above your dial frequency in the left half of the RX panel, or if you have to go to LSB to receive USB signals. If you observe this behaviour, check this box.

HW AGC RX1. If checked, automatic gain control (AGC) that is implemented in hardware in the radio is enabled. Note that in this case S meter readings may be significantly off.

For SoapySDR radios running two receivers, there is an additional check box HW AGC RX2.

SoapySDR gain elements at the bottom of the menu

If one of the receiver of transmitter channels of the SoapySDR device features more than a single gain elements, these elements are displayed at the bottom of the Radio menu, separated by a thin horizontal line. Normally the RF gain and TX drive sliders should be enough. However, it can occur that the automatic distribution of the overall gain on the individual gain elements, as done by the SoapySDR driver module for that radio, is not optimal. In this case, the individual gain elements can be controlled by spin-boxes at the bottom of the Radio menu. If such gain elements are changed, the new resulting overall gain is calculated and the RF gain slider is re-adjusted if it is on display.

6.2 The Screen Menu

The Screen menu lets you dynamically change the font used by piHPSDR and the size of the piHPSDR main window, it also lets you choose between different VFO bar layouts that are designed to fit on a screen with a given width. Furthermore, one can select whether the Zoom/Pan, the Sliders, or the Toolbar area are shown or hidden. The menu is opened via the main menu and the Screen button and is shown in Fig. 6.3.

In the first line, starting with Font used, there is a combo-box for choosing the font. On MacOS, the default font is system-ui as shown, which is Apple's default user interface font. Other choices for MacOS are Helvetica Neue, Monaco, and Arial which should be available (installed) on virtuall all Ma-

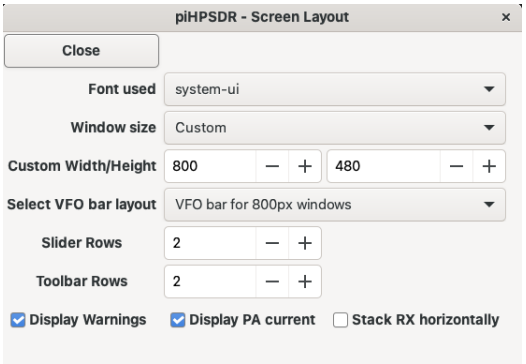

Fig. 6.3: The `Screen` menu.

cOS systems. For LINUX systems, the fonts are `Open Sans` (default), `Roboto Mono`, `FreeSans`, `Roboto`, and `NunitoSans`, the last one is now the default user interface font on RaspberryPi systems. Some people prefer mono-spaced fonts, therefore `Monaco` (MacOS) and `Roboto Mono` (LINUX) have been included in the list. The LINUX installation procedure takes care of installing (most of) these fonts. Changing the font affects all menus, the slider, zoom and tool bars, as well as the VFO bar.

In the second line, starting with `Window size`, one can choose the size of the piHPSDR window. Currently the choices are `Full Screen`, `Custom`, `640*400`, `800*480`, `1024*600`, and `1280*720`. For the `Full Screen` choice, the piHPSDR will occupy the entire screen space (in a multi-monitor setup, this is the area of the monitor on which the piHPSDR window was opened upon program start). Note that all sorts of problems with the transition to `Full Screen` have been reported, that mostly originate in the different window managers. The `Full Screen` option has been added for convenience, but it is not guaranteed to work since piHPSDR must rely on the window manager doing things correctly.

For `Custom`, piHPSDR uses the screen dimensions specified by the spin buttons in the third line. Note these buttons are only active if a custom layout is chosen. The other choices set the screen dimensions as indicated. One can, for example, set and use a preferred custom size, and temporarily switch to a small pre-defined geometry if one needs screen space for other work. Switching back to `Custom` then restores the preferred window size.

If the window width is decreased such that the VFO bar currently chosen

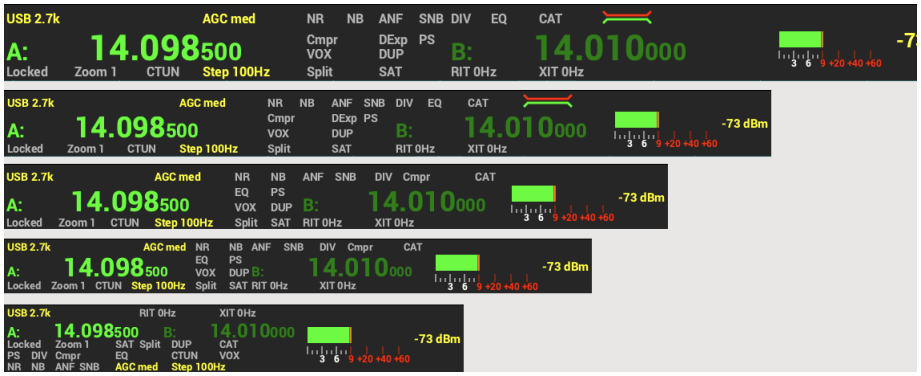

Fig. 6.4: Five choices for the VFO bar built into piHPSDR.

does no longer fit, piHPSDR automatically switches to a smaller VFO bar, the current choice shown in the pop-down menu `Select VFO bar layout` is updated. This menu lets you choose the layout of the VFO bar. In Fig. 6.4 five pre-defined layouts are shown.

These layouts require a screen size of 1152, 1024, 896, 800, and 640 pixels (from top to bottom). Screens slightly wider (about 50 pixel) than these minimal values then allow for a larger meter area with increased readability, this enlarged meter is automatically used if the display width permits. There is one even larger VFO bar (for a screen width of 1960 pixels) available that is not shown but does not look different from the topmost one in Fig. 6.4. The VFO bar has been described in detail in chapter 4.8. If you choose a VFO bar layout that is wider than the current window width allows, the window width is automatically adjusted (increased) when using a custom dimension, and one advances to a larger pre-defined size if using a non-custom size. On the other hand, choosing a VFO bar layout that is smaller than before does not affect the screen dimensions, but there will then be some empty space between the VFO-B frequency and the meter. Note that the smallest VFO bars do not offer enough space to display all controls: for example, the indicator for the downward expander and the actual filter edges are missing in the smallest layouts, but if one is restricted to such small screen one has to live with it.

Fig. 6.5 shows, as an example, piHPSDR running in a window as small as 640*400 pixels. It is admitted that this looks rather squeezed, and this will probably look over-crowded when running two receivers. However, for

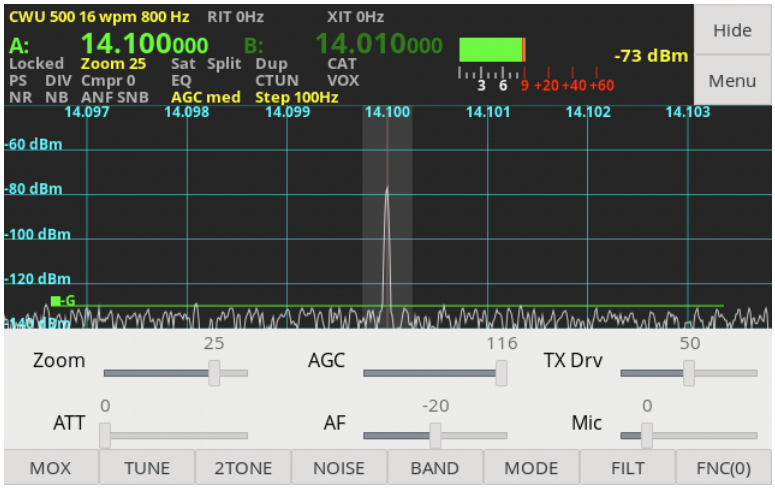

Fig. 6.5: piHPSDR running in a 640 x 400 window.

portable operations such small windows are often desired, if piHPSDR is to be run alongside with a logbook and digi mode program on a laptop. Note that the piHPSDR menus are designed to fit on a window 800*480 pixels large and might not fit into a piHPSDR window smaller than that.

`Stack receivers horizontally`. If checked, this puts the panels of the two receivers (if two receivers are used) side-by-side instead of on top of each other.

`Slider Rows`. This spin-button lets you choose how many rows of sliders (0–3) are shown. Choosing zero rows means that no sliders are shown at all. This however, makes little sense unless you operate an external controller that has knows with which you can change, for example the RX audio volume or the AGC gain. For temporarily gaining vertical space, use the `Hide` button at the top right of the main window.

`Toolbar Rows`. This spin-button lets you choose how many toolbars (0–3) are shown. Choosing zero rows means that no toolbar is shown at all.

`Display Warnings`. There are several non-fatal conditions that can be displayed either on the pan-adapter of the first receiver or, in non-duplex mode, on the TX pan-adapter. Normally these warnings are no reason to worry if you see them only occasionally. These conditions are

Sequence Error. Packets from the radio arrive in the wrong order, or

packets are missing. If this occurs frequently, check the connection between the radio and the host computer.

ADC overload. The RF input level is too high for one of the analog-to-digital converters. If you see this, increase attenuation in the RF front end.

TX FIFO under run/over run The TX IQ data packets sent to the radio while transmitting either arrive too fast or too slow, such that the first-in/first-out queue of TX samples inside the FPGA of the radio either overflows or drains.

High SWR During TX, an SWR above the SWR threshold has been detected. If this occurs frequently, check your antenna. The SWR warning threshold (default: 3.0) can be set in the TX menu. Because the forward and reflected power used to calculate the SWR have possibly been measured at slightly different points in time, the calculated SWR may be much too high at the beginning or the end of an RF pulse. piHPSDR has been instrumented to avoid such false warnings by using power readings averaged over some time, but it still cannot be excluded that a too high SWR is reported for a short amount of time.

Display PA current If this is checked, the PA supply voltage and the PA current are shown while transmitting. Such data is only available for Orion-II and SATURN boards (Anan-7000/8000 and Anan-G2) and the HermesLite-II. For the HermesLite-II, the PA temperature and the PA current are shown.

6.3 The Display Menu

The Display menu is used to customise the overall layout of the piHPSDR window and the pan-adapters of the active receiver. Adjustments for the TX pan-adapter must be done in the TX menu. The menu has two sub-menus, namely the general settings and the peak label settings, the selection is done in the topmost row.

6.3.1 Display Menu: General

For the general settings, the Display menus shown in Fig. 6.6.

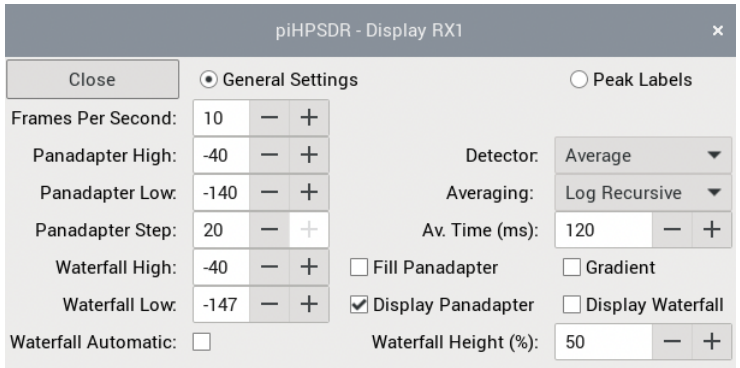

Fig. 6.6: The `Display` menu (General settings).

`Frames Per Second` This adjust how often the RX display is re-drawn. 10 frames per second (the default) is a good value.

`Panadapter High` This value is the dBm value of the RX signal strength at the top of the RX spectrum scope. A value of -40 dBm corresponds to S9 + 33 dB for HF signals.

`Panadapter Low` This value is the dBm value of the RX signal strength at the bottom of the RX spectrum scope. A value of -140 dBm is usually low enough such that the noise floor is still on display.

`Panadapter Step` This value is the spacing of the horizontal lines on the spectrum scope. Lines are drawn at dBm values that are multiples of the step size.

`Waterfall High` This is the RX dBm value that will lead to the brightest colour (yellow) in the waterfall. If the `Waterfall Automatic` box is checked (see below), this spin button has no effect.

`Waterfall Low` This is the RX dBm value below which the waterfall will be black. If the `Waterfall Automatic` box is checked (see below), this spin button has no effect.

`Waterfall Automatic` If this box is checked, the `Waterfall High` and `Waterfall Low` parameters are not used. Instead, the waterfall high/low values are determined automatically in each update of the waterfall, and these min/max values are then used instead of the waterfall High/Low control values to determine which colour belongs to which signal strength.

Detector Here one can choose between Peak, Rosenfell, Average and Sample. The Rosenfell detector is probably closest to what one knows from a spectrum analyser. The Average detector is usually preferred since it is less ,,nervous".

Averaging Here the possible choices are None, Recursive, Time Window and Log Recursive. For the details, see the WDSP manual.

Av. Time (ms) If averaging is used for the spectrum scope, the time constant involved in averaging can be set here.

Fill Panadapter This is used to enable/disable the ,,Filling" option for the RX spectrum scope (see chapter 4.4).

Gradient This is used to enable/disable the ,,Gradient" option (colour coding) for the RX spectrum scope (see chapter 4.4).

Display Panadapter. This option enables/disables the pan-adapter display on the RX panels. With the pan-adapter enabled, you can have the waterfall alone.

Display Waterfall This option enables/disables the waterfall display of the RX panels. Note if both the waterfall and the pan-adapter is disabled, there will simply be a large ,,hole" in the piHPSDR window. This makes little sense except on machines with very weak CPUs, since *not* displaying neither waterfall nor pan-adapter saves some CPU time in the graphics back-end.

Waterfall Height (%) This parameter can be chosen between 20 and 80 (percent) with the spin button to the right of this label, and only has an effect if *both* the panadapter and the waterfall are displayed. In this case, the parameter determines how much vertical space the waterfall gets, as a percentage of the total height of the panel for the receiver. The default value is 50 and means that panadapter and waterfall are of equal height.

6.3.2 Display Menu: Peak Labels

The "Peak Labels" sub-menu allows one to control if and how labels (values in dBm) are printed at the most prominent peaks on the RX panadapter (Fig. 6.7):

Label Strongest Peaks This enables peak detection and labelling.

Label in Passband Only If the box is checked only the peaks within the

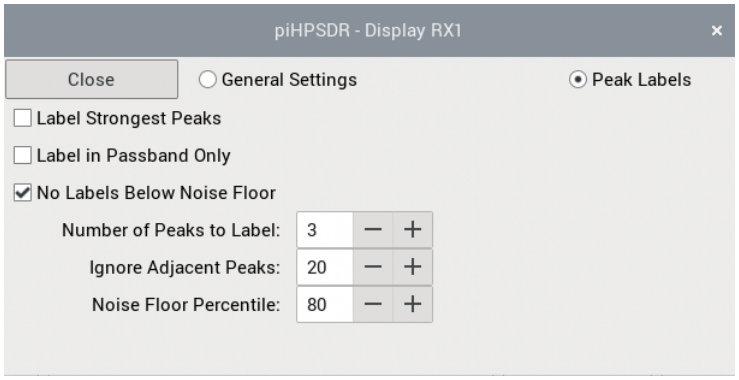

Fig. 6.7: The `Display` menu (Peak label settings).

current filter pass-band are shown.

`No Labels Below Noise Floor` This suppresses labelling of peaks below the noise floor.

`Number of Peaks to Label` Set maximum number of peaks to be shown. There can always be less, but no more than this number. The strongest peaks will be labelled first.

`Ignore Adjacent Peaks` Determine how close two peaks can be such that both are labelled. The higher the number the closer the peaks can be to each other. It's a divisor of the full window width so setting it to 1 will mean only one peak can occupy the entire window,

`Noise Floor Percentile` This is the percentile setting that defines the noise floor. If the percentile is 50, then the noise floor is the level of that pixel that is in the middle of the list if all pixels are sorted by their dBm value (a safety margin of 3 dBm is added). If the `No Labels Below Noise Floor` box is ticked peaks below this value will not be labelled.

6.4 The `Meter` menu

The `Meter` can either be opened simply by clicking in the meter area, or through the main menu. Only few choices can be made here.

`Meter Type` Here you can select between a digital and an analog meter. The

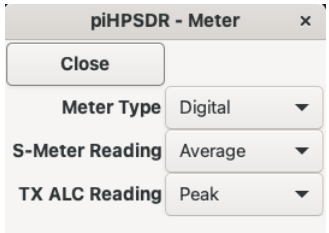

Fig. 6.8: The Meter menu.

four different designs (either analog or digital, and during RX or TX) have already been shown in Sec. 4.9.

In both cases, there is a choice between Peak and Average reading, which refers to peak envelope power and average power. Here averaging is done over relatively short times. For a two-tone signal for example, the peak reading is 3 dB above the average reading.

S-Meter Reading Here you can choose whether the S-meter reports a Peak or an Average value (default is Average). Note, however, that in order to make the display less ,,nervous" a moving average with a rather long time constant (about 0.5 sec) has been implemented on top of both the Peak and Average S-meter readings.

TX ALC reading Here, the possible values are again Peak and Average. Normally one uses the peak ALC value. If it is negative, the level of the TX audio signal or the Mic gain slider should be increased. This is very important since, for example, PureSignal only works if the TX audio input has maximum amplitude, so you can put the drive slider to zero, then put the radio into TX mode, speak into the microphone and slowly increase the Mic gain until the ALC value shown is about zero (if the ALC value is positive, you have to reduce the Mic gain). The average TX ALC value only is of interest if you search for settings for the speech processor or the continuous frequency compressor. For normal speaking into the microphone, the average ALC value will be negative even if the Mic gain slider is adjusted for zero peak ALC value. A compressor will distort the TX audio such that this gap gets smaller. More details can be found in chapter 15 discussing the entire TX audio chain.

For RX-only radios, the TX ALC setting will not be shown in the menu.

6.5 The XVTR (Transverter) Menu

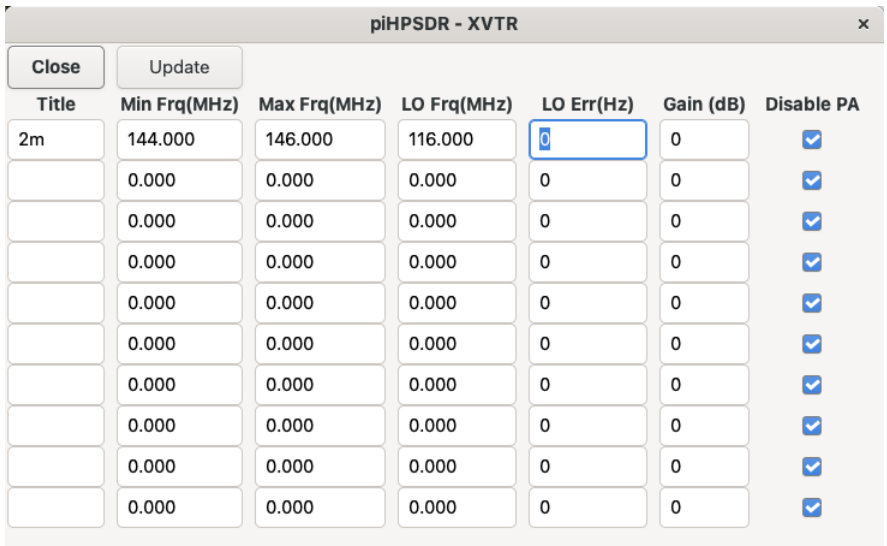

Fig. 6.9: The XVTR (transverter) menu.

The XVTR menu lets you define up to ten additional bands that you can work on using transverters. The bands should normally be beyond the standard frequency range of the radio, otherwise the calculation of a band from a given frequency will sometimes not work. Fig. 6.9 shows the XVTR menu with data for an example for a transverter which you can drive with frequencies between 28 and 30 MHz and which will convert them to the frequency range 144 to 146 MHz, and which will receive frequencies in that range and mix them down to the 10m band. The data you have to enter in the XVTR menu (use the first free entry) are as follows:

Title In this column, enter a name for your band. You can choose whatever name you want, this is the one that will be displayed in the Band menu. In the present example, use "144" or "144 MHz" or "2m". If the title string is empty, all transverter data for this band will be cleared.

Min Frq Enter the lowest frequency of the transverter band in MHz, in the present case, 144.

Max Frq Enter the highest frequency of the transverter band in MHz, in the present case, 146.

LO Frq This is the frequency offset (in MHz) between the radio frequency and the operating frequency. In this case, use 116. From this offset, radio frequencies between 28 and 30 MHz will be used for operating frequencies between 144 and 146 MHz.

LO Err This entry can be used for a fine calibration of the frequency. The value (in Hz) is added to the local oscillator (LO) frequency in MHz.

Gain For each transverter band, the RX gain of the transverter can be specified here. If the transverter has a positive gain, the signal will appear too strong in the meter and the pan-adapter, so entering a positive number here will *reduce* the dBm reading in the meter area. This setting affects the meter reading, and the RX pan-adapter and waterfall.

Disable PA This checkbox is only present for HPSDR (Protocol-1 and Protocol-2) radios and indicates that the PA of the radio should be disabled when using the transverter band. This implies that the radio has some sort of low-power output that is used to drive the transverter.

Update When entering data in the text fields, it does not become effective immediately. The data is stored if you leave the menu, but also if you push the Update button. After pressing this button, the text fields will be regenerated, so if you have typed in, for example, "144" as the minimum frequency, this text field will change to "144.000". Consistency checks are performed as well: the minimum and maximum frequencies are recalculated from the local oscillator frequency and the frequency range of the radio, if something does not fit. It is therefore recommended to push Update and review the text fields before leaving the XVTR menu.

To give an example of how to setup transverter operation, a sample setup for QO-100 operation with an AdalmPluto is given. The Pluto is operated via the SoapySDR interface. For receiving in the 10 GHz band, one uses a so-called LNB (low noise block) in the focus of the parabolic antenna which converts the 10 GHz signal down to about 740 MHz. The setup for defining the "QO100" transverter band is shown in Fig. 6.10. Since the Adalm Pluto is operated via the SoapySDR interface, there are no check boxes for disabling the PA.

The LO (local oscillator) frequency is always chosen such that it is below the RX frequency, and the difference between the RX frequency and the LO frequency is the frequency the radio is operating on. The name (here:

piHPSDR - XVTR

Close | Update

Title	Min Frq(MHz)	Max Frq(MHz)	LO Frq(MHz)	LO Err(Hz)	Gain (dB)
QO100	10489.500	10490.000	9750.000	0	0
	0.000	0.000	0.000	0	0
	0.000	0.000	0.000	0	0
	0.000	0.000	0.000	0	0
	0.000	0.000	0.000	0	0
	0.000	0.000	0.000	0	0
	0.000	0.000	0.000	0	0
	0.000	0.000	0.000	0	0
	0.000	0.000	0.000	0	0
	0.000	0.000	0.000	0	0

Fig. 6.10: XVTR setup for QO-100 operation.

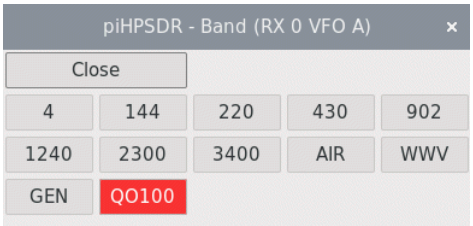

Fig. 6.11: The Band menu after defining the QO100 band.

QO100) can be chosen as one likes, but it cannot be left empty. Once the transverter band has been defined, it shows up in the BAND menu (see Chapter 7.2), as shown in Fig. 6.11. Note this is a screen shot from operation with an Adalm PLUTO, where the 70 MHz (4 m) band is the lowest one.

To make QO-100 Operation easy, the working frequencies, the CTUN mode etc. should be stored in the first band stack entry of the QO100 and the 13 cm band. To this end, click on the QO100 in the band menu (Fig. 6.11 and adjust the VFO-A frequency until the display reads 10489.500 MHz. Now swap VFO-A and VFO-B (A<>B command) and enter 2400 MHz into VFO-A using the VFO menu (Chapter 7.1). and swap the VFOs again. SAT mode will now ensure the 10.489 GHz Receive Frequencies and the 2.4GHz Transmit frequencies track correctly when changes are made to the Receive frequency. piHPSDR will now remember these settings when you select the bands. The complicated swapping was necessary since band stack entries

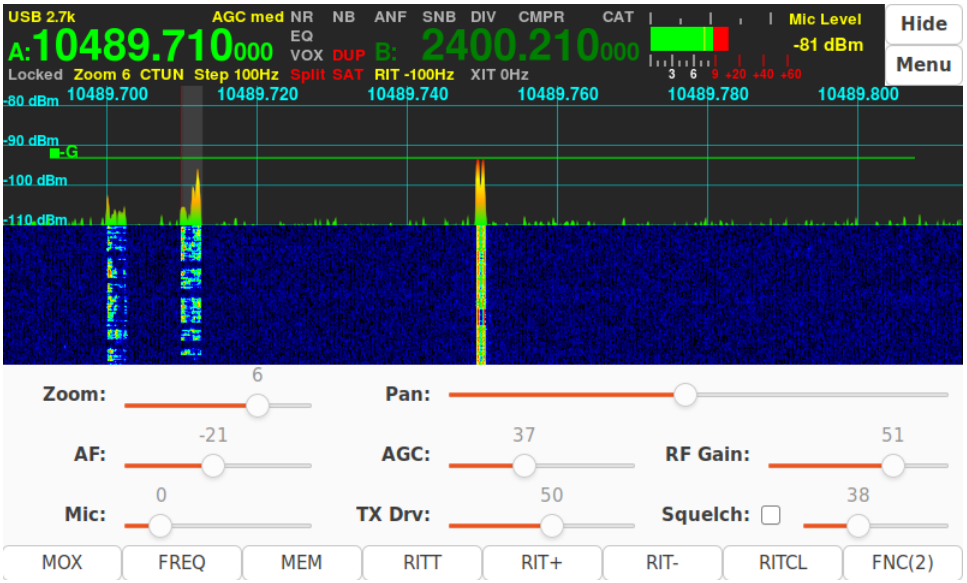

Fig. 6.12: Working QO100 with piHPSDR and the Pluto.

will only be stored from VFO-A.

Fig. 6.12 (a contribution from a ham using the Adalm Pluto for QO-100 operation) gives an impression of how this actually works. Note that the default setup is working Split, Duplex and SAT. Split mode implies that VFO-B is used for transmitting. Duplex mode implies that the receiver continues to work while transmitting so you can watch and hear your own signal. The TX spectrum scope then appears during transmitting in a small separate window.

6.6 The Server Menu

This menu is part of client/server remote operation, which is explained in chapter 12.

Chapter 7

The Main Menu: VFO-related menus

In this chapter we discuss the menus from the second column of the main menu. These are all VFO-related menus.

7.1 The VFO menu

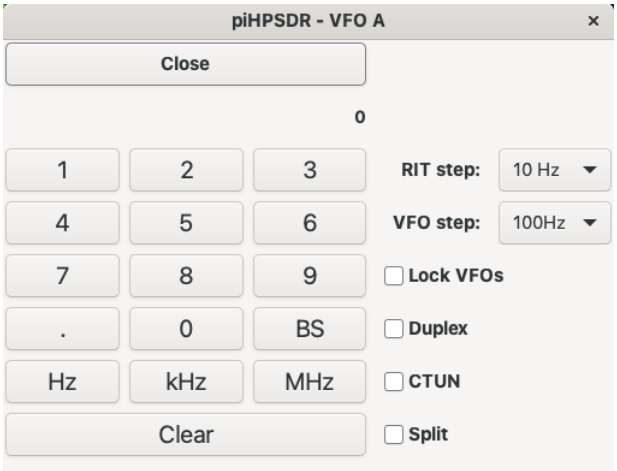

Fig. 7.1: The VFO menu.

The VFO menu can be used for direct frequency entry and to enable/disable some frequently used options. If the menu is opened, it refers either to VFO-A or VFO-B. If opened via the main menu, it automatically refers to the VFO controlling the active receiver. The easiest (and therefore recommended) way to open the VFO menu is just to make a mouse click (or a touch screen press) into the VFO bar. If clicked in the left half of the VFO bar, the menu is opened for VFO-A and if clicked in the right half, it is opened for VFO-B. The VFO menu is shown in Fig. 7.1.

The ,,keypad" is used for direct frequency entry. You can enter digits and a decimal point. While entering a number the string entered so far is not only shown in the upper part of the VFO menu, but also (in yellow digits) in the VFO bar. The other buttons of the ,,keypad" have a special meaning:

BS Backspace. This cancels the last entered character (digit or decimal point).

Hz This enters the frequency ,,as is".

kHz This multiplies the frequency just entered with 1000 and enters it. This means, the number entered is interpreted as a frequency in kHz.

MHz The string typed in so far is interpreted as a frequency in MHz and this frequency is transferred to the VFO.

Clear The string typed in so far is deleted, the VFO frequency is not updated.

The commands entered by clicking the buttons of the keypad in the VFO menu can also be entered by push-buttons from a GPIO or MIDI controller, see the NumPad commands in Appendix A.

In addition to frequency entry, the VFO menu offers a convenient way of changing some piHPSDR settings, simply because the VFO menu can be opened by a simple mouse click into the VFO bar.

RIT Step In this pop-down menu, the RIT step size can be chosen (1/10/100 Hz).

VFO Step In this pop-down menu, the VFO step size can be chosen. The VFO step sizes range from 1 Hz to 1 MHz.

Lock VFOs With this check box enabled, VFO frequencies cannot be changed by turning a VFO dial (GPIO or MIDI controller), or by clicking/dragging in the RX panel. Band changes (via the Band menu) and other VFO related functions still work.

Duplex and Split. With these check boxes, you can put the radio in Duplex or Split mode, see the Radio menu.

CTUN. With this check-box, you can put the VFO this menu is referring to into CTUN mode. In CTUN mode, the spectrum scope does not move when changing the frequency, rather, the RX ,,window" moves. CTUN mode does not affect TX operation.

7.2 The Band menu

The Band menu lets you change the band of the active receiver. It is shown in Fig. 7.2. When the menu opens, the button of the current band is highlighted.

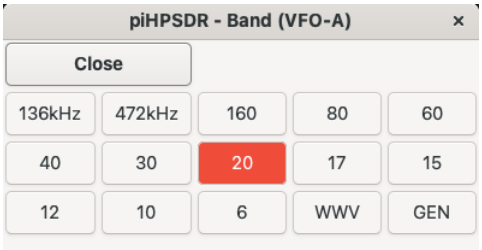

Fig. 7.2: The Band menu.

Pressing a button corresponding to another band, two things happen: first, if the active receiver is controlled by VFO-A, the current frequency is stored in the current band stack (which is thus updated). Then, the new band is chosen, the frequency and mode are from the active band stack entry of the new band. This means that if you switch to another band and shortly thereafter switch back to the original band, the frequency and mode is restored to what you had before. This also involves restoring a RX/TX profile (see chapter 14) associated with that mode.

If you hit the highlighted button, you will not change the band (since you hit the button of the current band) but instead will cycle through the band stack of that band (see the BndStack menu).

Note that the band menu may look different from the one shown here: there are many bands (24 bands plus up to 8 transverter bands) defined in piHPSDR. However, the bands that are outside of the radio's frequency limits

are not shown. For example, a radio whose maximum frequency is 30 MHz will not show the 6m band. The GEN (General) band encompasses the whole frequency range of the radio. If you set the frequency (e.g. via the VFO menu) to a frequency outside of any of the other bands, you will end up in the General band. If you have defined transverter bands (see the XVTR menu) they will be shown, with the title you have chosen, in the Band menu.

7.3 The BndStack (Band stack) menu

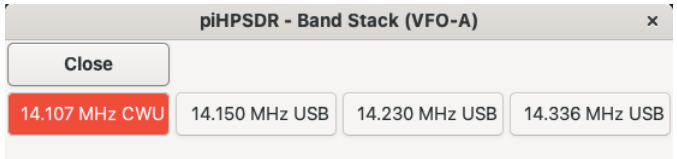

Fig. 7.3: The BndStack (Band stack) menu.

For each band, the band stack is collection of operating frequencies/parameters. The idea is that you can have preferred or recently visited frequencies to which you can easily come back. The parameters that are actually stored are the frequency, the mode (e.g. USB or FMN), the filter, and whether CTUN is enabled or disabled. The band stack parameters also encompass some FMN-specific parameters, namely the deviation and the CTCSS setting. If you open the BndStack menu (Fig. 7.3), the buttons tell you about the frequency and the mode, and the band stack entry currently selected is highlighted.

If you press the highlighted button, the parameters which are currently effective are stored in that band stack entry. If you press another (non highlighted) band stack button, then first the current parameters are stored in the highlighted band stack entry, and then the parameters of the new entry become effective. Note that parameters in band stack entries are only changed if the active receiver is controlled by VFO-A. Note further that when changing the mode as a consequence of moving to another bandstack entry, the RX/TX profile associated with the new mode (see chapter 14) is restored.

7.4 The Mode menu

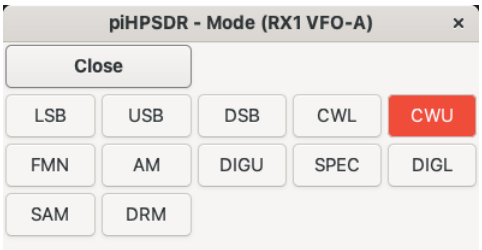

Fig. 7.4: The Mode menu.

The Mode menu lets you change the mode of the active receiver, so you can switch, say, from LSB to CWU or DIGU. The mode menu simply lists the available modes, the current one is highlighted (Fig. 7.4). As detailed in chapter 14 on RX/TX profiles, a lot of settings are changed "behind the scenes" if you change a mode.

7.5 The Memory menu

The Memory menu gives you access to ten memory slots. The menu is shown in Fig. 7.5. You can store the current operating frequency of the active receiver in any of the five slots by clicking a button in the left column (e.g. "Store M2"), or you can restore data from any slot by clicking on one of the entries in the right column which shows the frequency, mode and filter width stored in that slot. In addition (not shown in the right column) the FM deviation and the CTCSS setting are stored in the memory slots. So if you have some often used frequencies (e.g. for a net), the Memory menu allows you to become QRV there with only few mouse clicks.

Special behaviour in SAT/RSAT mode. Normally (that is, not in SAT or RSAT mode), a memory store operation stores the frequency/band/mode/filter settings of the active receiver in the memory slot, and a memory recall operation retrieves this data and modifies the active receiver accordingly. This would be too little to become QRV if one operates one of the satellite modes. So when SAT or RSAT is active, the settings of both VFO-A and VFO-B are stored in the memory slot, together with the SAT/RSAT, Split and Duplex

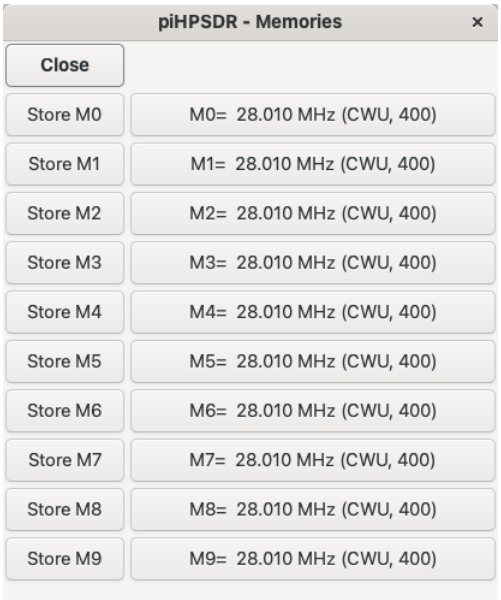

Fig. 7.5: The `Memory` menu.

states. When recalling a memory slot where the `SAT` or `RSAT` state is set, then (and only then!) both the VFO-A/B frequencies/modes etc. are restored together with the `Split` and `Duplex` states. In the menu, the SAT or RSAT flag is indicated with the mode.

Chapter 8

The Main Menu: RX-related menus

The second column of the main menu contains menus which allow you to change receiver settings.

8.1 The RX Menu

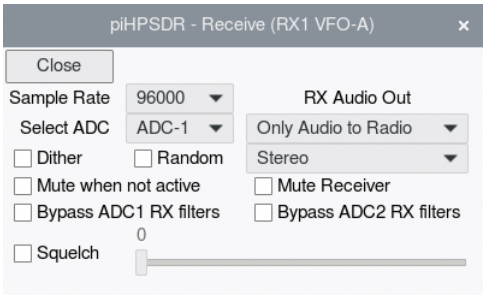

Fig. 8.1: The RX menu.

Invoking the RX menu through the main menu always implies that the settings of the active receiver are to be modified. Using a mouse, you can also open the menu by a secondary click (using the right mouse button) into the receiver panel. This way, if piHPSDR is running two receivers, you can open the RX menu for both receivers (the active receiver as well as the other one),

depending on in which panel you have right-clicked. Note secondary clicks are usually not possible with a touch screen. The menu is shown in Fig. 8.1.

`RX Audio Out`. In the drop-down menu below, you can check the audio output device associated with the receiver. `Only Audio to Radio` means that no sound output device (sound card) is used, and RX audio is only included in the data stream to the radio (if supported). HPSDR radios featuring an audio codec typically have jacks where a headphone and a microphone can be plugged in, but there are also radios such the barefoot HermesLite-II and all SoapySDR radios where `Only Audio Radio` effectively means ,,no audio". If the computer running piHPSDR has audio output devices (either built-in or USB sound cards), one of them can be selected from the drop-down menu. Note that if such a device is selected, the RX audio will additionally also be sent to the radio.

Below the drop-down menu selecting the output device, there is a second one where one can choose between `Stereo`, `Left` and `Right`. While `Stereo` effectively means ,,do nothing", choosing `Left` means the the audio samples for the right channel are muted.

If running two receivers, it depends on the audio output module whether it is possible to use the *same* audio output device for both receivers. With PulseAudio (and PipeWire), this is possible which gives you an additional bonus: Choose the same output device for RX1 and RX2 and activate only the left channel for the first and only the right channel for the second receiver. With this setup, one gets the audio output of the first receiver on the left ear and the audio output of the second receiver on the right ear. This can be very convenient for hunting DX in split mode, because one then hears the hounds and the fox on different ears.

The audio settings (output device and stereo settings) of RX1 (only RX1!) are part of the RX/TX profile (see Sec. 14). Typically, you will choose an audio output device connected to a headphone for SSB and CW, but a virtual audio cable or a sound card connected to another computer running a digi program for digi modes. piHPSDR then automatically switches the audio output device when switching modes. RX2 audio settings are not part of the RX/TX profile: changing RX2 audio settings will not change the profile, and changing the mode will not change the RX2 audio settings.

`Sample Rate`. This box is only shown for radios running Protocol-2, since

only there the receivers can have an individual sample rate. For radios running Protocol-1 or radios accessed through the SoapySDR library, the sample rate is a global quantity that is modified through the Radio menu (see above).

Select ADC. This box is only shown if the radio has more than one analog-to-digital converter (ADC), such as Orion, Orion-II and Saturn boards. These radios have two ADCs so you can choose whether the receiver gets data from ADC0 or ADC1. For these radios, nearly all antenna jacks go to ADC0, while there is a jack denoted "RX2" (or similar) that is connected with ADC1. In most cases, ADC0 is used for normal operation, while ADC1 can be used for connecting a dedicated RX antenna.

Note: Diversity. When using Diversity reception, the ADC setting is overridden, since there data streams from ADC0 and ADC1 are combined (mixed).

Dither. When checked, the ,,dither" bit is set which affects the operation of the ADC converter in some HPSDR boards.

Random. When checked, the ,,random" bit is set which affects the operation of the ADC converter in some HPSDR boards.

Note: On radios where the ,,dither" and ,,random" bits have no effect, these two check boxes do not appear. Examples for such radios are Red-Pitaya based radios or the RadioBerry, or all radios connected through the SoapySDR library. For the HermesLite-II, there also is no such function, but these bits are hi-jacked for some custom functions, so they appear here in the menu.

Preamp. This checkbox is not shown in Fig. 8.1, it only occurs for some legacy HPSDR boards which had a switch-able RX preamp.

Mute when not active. If checked, local audio (e.g. to a sound card) from this receiver is muted when it is not the active receiver.

Mute Receiver. If checked, the audio from this receiver is muted. This applies both to the audio data stream to the radio and local audio (sound cards or virtual audio cables).

Bypass ADC0 RX filters. This box is only shown if the radio is an HPSDR radio and an ALEX filter board is selected (see the Radio menu). If checked, the filters in the RF front-end for ADC0 are by-passed during receive. This option will normally only be used for radios that have band-pass filters in the

front end, and if one operates two receivers both running on ADC0 data on different bands. Without checking this option, only the signals for the band of the active receiver will pass the RF front-end filters. Older radios (up to Anan-100/200) have a combination of a low-pass and a high-pass filter in the RX path to ADC0. If these radios run two receivers, the low-pass filter is selected based on the higher of the two RX frequencies, and the high-pass filters is selected based on the lower one. This ensures that the signals for both receivers fall into the filter pass band.

`Bypass ADC1 RX filters`. This box is only shown for HPSDR radios with the ALEX filter board selected (see the `Radio` menu), and if the RF front-end for ADC1 features a filter board (Anan-7000/8000 and G2 radios). If checked, the filters in the RF front-end for ADC1 are by-passed during receive.

`Squelch`. This check-box lets you enable or disable squelch for this receiver, and the slider to the right lets you adjust the squelch level. Use these functions if you want to use squelch and do not have the squelch slider on display.

8.2 The `Filter` menu (including notches)

With the `Filter` menu, you can change the filter of the active receiver. In addition, up to three notches can be added manually to the filter passband.

8.2.1 Choosing filters for non-FM modes

For each mode (except FM), there are ten fixed filters and two variable filters, see Fig. 8.2. It depends on the current mode which filters are at your disposal, and Fig. 8.2 is what you see for USB and LSB modes. The lower part controls the manual multi-notch filter and will be discussed below. The filter currently active is highlighted, and you can choose another filter simply clicking the button. For USB and LSB, the filters are such the low-frequency cut (in the audio domain) is at 150 Hz, so a 2.7k filter actually encompasses audio frequencies from 150 to 2850 Hz. With the two variable filters (`Var1` and `Var2`) you can be more flexible in the low audio frequency range. Here you can individually select the low- and high frequency cut (both frequencies refer to the audio domain, and are thus both positive value for USB and LSB). There is one pair of variable filters for each mode.

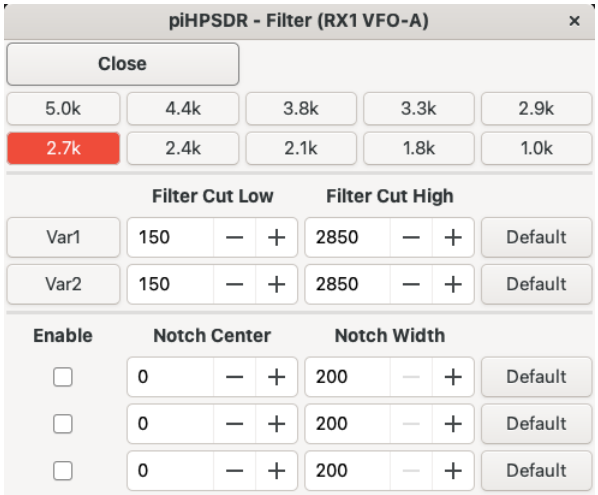

Fig. 8.2: The `Filter` menu (single side band modes).

The pre-defined filters for the digital modes DIGU and DIGL are a little bit different. For filter widths up to 3 kHz, the filter is centred around 1500 Hz. For example, a 1.0k filter for DIGU/DIGL passes audio frequencies between 1000 and 2000 Hz.

For modes such as CW and AM, low/high cutoff frequencies have little meaning, so the `Filter` menu looks slightly different (Fig. 8.3). The fixed filters are designated by their width, they are centred around zero (for AM) or around the CW side tone frequency (for CWU and CWL). For the variable filters `Var1` and `Var2`, the spin buttons can set the filter width and the filter shift. Normally you will not want to change the filter shift, but it may help in special cases.

If you click a filter that had already been selected, seemingly nothing happens. But if filter edges of the active receiver haven been changed through controller or panel buttons, then clicking the filter seemingly already in use (and this includes the `Var1` and `Var2` filters!) will restore the nominal filter edges, which for the variable filters are those displays here in the `Filter` menu. Restoring nominal filter edges also happens on each filter change and thus also on each mode, band, or band stack change, and on recalling a memory location. This is so because a band/band stack/memory change restores the mode stored for that case, and each mode change also sets a new filter because it loads the RX/TR profile associated with that mode.

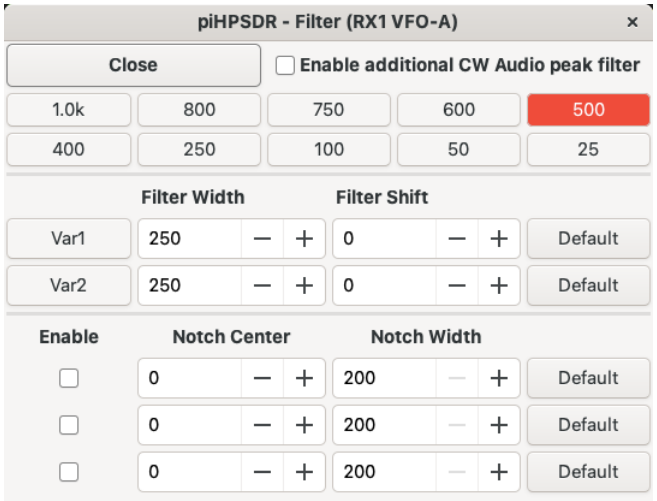

Fig. 8.3: The `Filter` menu for CWL/CWU.

`Enable additional CW Audio peak filter`. If the mode of the active receiver is CWL or CWU, there will be an addition check box in the top row of the menu. Here you can enable/disable an audio peak filter that is applied to the final audio output of the receiver, that is, on top of the regular filtering. The audio peak filter will only be effective in the CW modes, its centre frequency is given by the CW side tone frequency and its width is automatically calculated, depending on the width of the primary filter. The audio peak filter can be used to dig out the CW signal from the noise (making the regular filter narrower also does this job). The audio peak filter can also help to tune to the correct frequency: the regular filters have a flat pass band so the received signal equally loud as long as it is in the pass band. The audio peak filter has a marked peak at the side tone frequency so you can tune for maximum signal volume to adjust your frequency to the received signal.

There is the function `CW Audio Peak Filter` that can be mapped on toolbar buttons or GPIO/MIDI buttons so you can quickly enable/disable the audio peak filter.

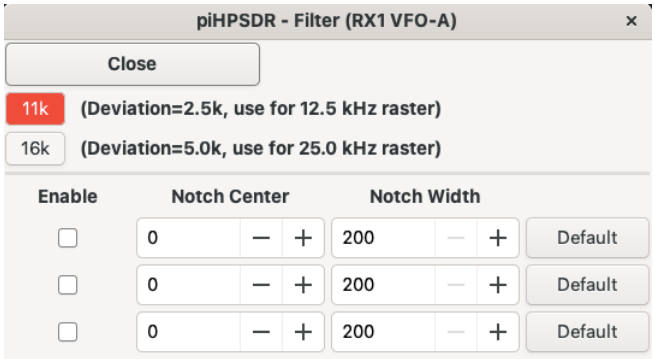

Fig. 8.4: The Filter menu for FM.

8.2.2 Choosing deviation for FM

In FM mode, the Filter menu only lets you choose between a deviation of 2500 Hz or a deviation of 5000 Hz (see Fig. 8.4). Irrespective of whether the box Use RX Filter in the TX menu (see Chapter 9.1.1) is checked, the deviation setting is used both for RX and TX. Filter edges (both for TX and RX) are then calculated according to Carson's rule. Assuming a maximum audio frequency of 3000 Hz, a filter width of 11 kHz and 16 kHz result for deviations of 2500 and 5000 Hz.

8.2.3 Manual Multi-Notch Filter

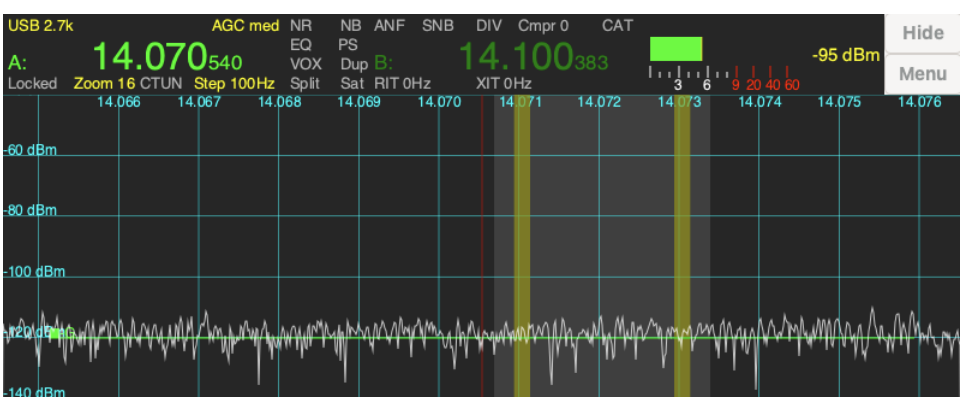

Fig. 8.5: RX panadapter indicating two notches.

Now we come to the lower part of the filter menu, where up to three notch filters can be engaged. The `Enable` column contains checkboxes for enabling/disabling any of the three filters, and the spin buttons in the columns `Notch Center` and `Notch Width` define the center frequencies and widths of the notch filters.

The center frequencies are bound to the baseband frequencies, that is, a frequency of zero corresponds to the center of the RX panadapter (the VFO dial frequency in non-CTUN mode), negative frequencies are in the left, and positive frequencies in the right half of the spectrum. So if some source of QRM is notched out, it remains notched out if you change the RIT value or if you change the frequency in CTUN mode. But when you change the frequency in non-CTUN mode, the notch will move away from the QRM. The active notches are indicated in the RX panadapter by transparent yellow areas, as shown in Fig. 8.5. In this example, the first notch has a center frequency of 500 Hz and a width of 200 Hz, the second notch a center frequency of 2500 Hz and also a width of 200 Hz. Note that in LSB mode, notch frequencies will usually be negative. Notches can be outside the filter passband (the area shaded grey in Fig. 8.5) but then they have no effect.

The `Default` buttons associated with each notch reset its width to the minimum width and its center frequency to the center of the filter passband. The main use of resetting the notch is to quickly get the notch in the middle of the filter passband when using CTUN or large RIT values, since then one needs notch center frequencies very far from zero.

Changing the center frequencies and width, and enabling/disabling a notch, can also be accomplished with GPIO- or MIDI-based controllers, see the commands starting with `NOTCH`.

Depending on the parameters of the digital signal processing, a notch has a minimum width to be effective, and piHPSDR will not set a notch width below that value. For the default RX filter tap length (see the `DSP` menu) of 2048, the minimum notch width is 200, and this can be decreased by choosing a larger number of taps. This will also increase the RX latency.

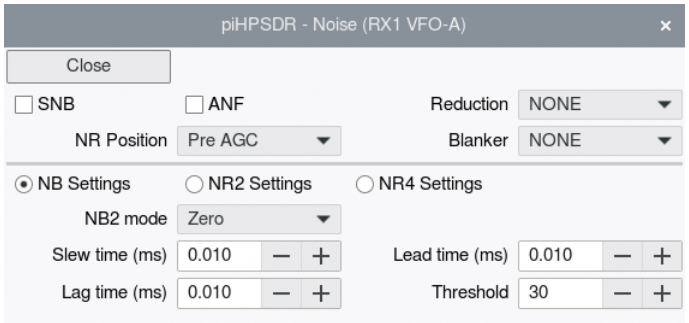

Fig. 8.6: The Noise menu (with NB settings).

8.3 The Noise Menu

With the Noise menu you can select a variety of noise reduction and/or noise blanker capabilities (Fig. 8.6). The upper part of the menu always looks the same, the lower part lets you fine-tune parameters for the noise blanker (NB) or the noise reduction models NR2 and NR4. For an in-depth explanation of the NR/NR2/NBcapabilities, the reader is referred to the WDSP manual. The noise reduction models NR3 (`RNNnoise`) and NR4 (`libspecbleach`) more information can be found starting at their github repositories

RNNoise: `https://github.com/xiph/rnnoise.git`
SpecBleach: `https://github.com/lucianodato/libspecbleach`

It should be noted that noise reduction in general has two goals, namely to reduce fatigue (that is, quieting the background) and to increase readability (that is, digging otherwise unreadable signals out of the noise). This implies that which settings are judged to be ,,best" very much depends on the beholder, and that you have to play around with the parameters until you find a setting that suits your needs.

Most noise reduction models only work reasonably well for phone. In my experience, NR4 does a very decent job also for CW.

You see a thin horizontal line in Fig. 8.6. Choosing one of the three buttons just below the horizontal line determines whether the lower part of the menu offers fine-tuning of NB, NR2, or NR4 settings. The set up for changing the noise blanker settings is shown in Fig. 8.6, below (Fig 8.7) you find the set up for changing the NR2 settings and the NR4 settings (Fig. 8.8). Note there

are no settings for NR3: the default ,,full" RNNnoise model is hard-wired into the code and used.

8.3.1 Upper half of the Noise menu

`SNB`. This check box lets you enable/disable the spectral noise blanker.

`ANF`. This check box enables/disables the automatic notch filter. The ANF is very good at eliminating a single-tone QRM carrier in SSB modes. It goes without saying that activating the ANF in CW is detrimental rather than beneficial, because here the signal is of the type the ANF tries to eliminate.

`NR Position`. In the RX chain, the noise reduction can be placed before or after the automatic gain control (AGC). The choice made here refers to *all* noise reduction capabilities (ANF, NR, NR2, NR3, NR4). Having the noise reduction before AGC (box unchecked) normally works better, but with sudden noise changes, having noise reduction after AGC (box checked) may be preferred.

`Reduction`. With this pop-down menu, you can choose the type of noise reduction namely no noise reduction, LMS noise reduction (NR), spectral noise reduction (NR2), RNNnoise reduction based on a trained neural network (NR3) and noise reduction using the `libspecbleach` library (NR4).

`Blanker`. With this pop-down menu, you can choose the type of noise blanker (no noise blanker, the preemptive wide band blanker NB or the interpolating wide band blanker NB2).

8.3.2 NB settings

A noise blanker works very different from noise reduction, since noise blanking is applied to the original RX IQ samples before any frequency shifts and down-sampling take place. Noise blanking is targeted at impulse noise, that is, small but strong impulses in the RX signal that are almost exclusively man-made noise. If you have a single source of noise (e.g. a Plasma TV) that drives you crazy, it is worth the effort to play around with the NB2 parameters, especially the timings. Different QRM sources will require different noise blanker parameters! The default parameters have been proven

useful for many situations, but for you a different setting may produce better results! The options to control the noise blanker algorithms are (for an in-depth discussion, see the WDSP manual):

NB2 Mode. The available choices for the interpolating NB2 noise blanker here are Zero, Sample&Hold, Mean Hold, Hold Sample, and Interpolate.

Slew time, Lag time, Lead time and Threshold. These parameters apply both to NB and NB2. piHPSDR currently does not allow to have a separate set of parameters for NB and NB2. The times are in the range 0 . . . 0.1 msec, and 0.01 msec is a good starting point. The Threshold can vary from 15 to 500. If normal signals are blanked out (erroneously treated as impulse noise), then this threshold should be increased. On the other hand, a too high value will prevent the noise blanker from detecting (and blanking) impulse noise.

8.3.3 NR2 settings

Of course it is perfectly possible to ,,play" with the settings here, hoping to find a combination that suits your needs. It is, however, recommended that you have read (and understood) the discussion of NR2 in the WDSP manual.

The noise menu with the NR2-specific control is shown in Fig. 8.7:

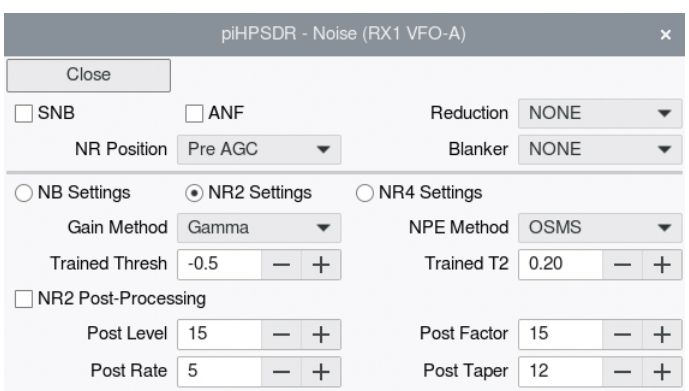

Fig. 8.7: The Noise menu (with NR2 settings).

NR2 Gain Method. The available choices for the NR2 gain-per-bin calculation are Linear, Log, Gamma, and Trained (Gamma is the default).

NR2 NPE Method. The available choices for the NR2 noise power estima-

tion are OSMS (Optimal Smoothing Minimum Statistics), MMSE (Minimum Mean-Square Error), and NSTAT (Non-stationary Noise Method), where OSMS is the default and normally preferred. MMSE and especially NSTAT are better adapted to cases where the noise changes fast. With rapidly changing noise, NSTAT is recommended especially with the ,,trained" gain method.

`NR2 Trained Thresh`. This spin-box sets a threshold (between −5 and 5) that is only used in the NR2 `Trained` method. The default value, −0.5, should be suitable in most cases.

`NR2 Trained T2`. The so-called T2 value of the trained noise reduction may vary between 0.02 and 0.3, and the default value 0.2 should be OK in most cases. Sometimes it happens that very weak signals are eliminated (that is, treated as noise). In this case one can try to *reduce* the T2 value.

`NR2 Post-Processing`. After noise reduction, processed speech often sounds harsh to listeners, and this can be alleviated by mixing in a little bit of white noise at the end, and tapering the frequency response of the signal. All of these post-processing features can be enabled/disabled by this check box, while the settings (level, factor, rate, and taper) can be specified with the spin buttons below this check box. For details, consult the WDSP manual.

`Post Level`. This controls the amount of ,,white noise injection". The range is from 0 to 100, the default being 15.

`Post Factor`. This controls the level of the white noise injected. The range is from 0 to 100, the default being 15.

`Post Rate`. This is a time constant (in seconds) and controls how fast injected white noise fades out. The range is from 0 to 100 secondes, and the default is 5 seconds. With a setting of 0, white noise is only injected during the syllables of the speech.

`Post Taper`. This can be used to suppress high-frequency audio that may be generated by the noise reduction. The range is from 0 to 100, a value of 100 corresponds to ,,tapering" beyond 24 kHz. The default value is 12 (taper beyond 2800 Hz) which is good for normal SSB. For ,,wider" modes (ESSB or AM), this value need be increased.

8.3.4 NR4 settings

The noise reduction menu with the NR4-specific settings is shown in Fig. 8.8.

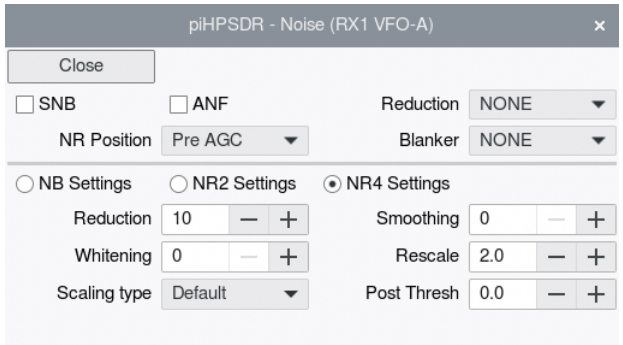

Fig. 8.8: The Noise menu (with NR4 settings).

NR4 involves the algorithms from `libspecbleach` which perform short-time Fourier transforms on the audio data.

Reduction. This value is in dB and can be between 0 and 20 (the default is 10). It is the amount of attenuation applied to the noise.

Smoothing. Percentage of smoothing to apply. Averages the reduction calculation frame per frame so the rate of change is less resulting in less musical noise but if too strong it can blur transient and reduce high frequencies. It goes from 0 to 100 percent.

Whitening. Percentage of whitening that is going to be applied to the residue of the reduction. It modifies the noise floor to be more like white noise. This can help hide musical noise when the noise is colored. It goes from 0 to 100 percent.

Rescale. Strength in which the reduction will be applied. It uses the masking thresholds of the signal to determine where in the spectrum the reduction needs to be stronger. This parameter scales how much in each of the frequencies the reduction is going to be applied. It can be a positive dB value in between 0 dB and 12 dB.

Scaling type. Type of algorithm used to scale noise in order to apply over or under subtraction in different parts of the spectrum while calculating the reduction. The pop-down menu can choose beween Default (a-posteriori

snr scaling using the complete spectrum), `CriticalBands` (a-posteriori snr scaling using critical bands) and `MaskingThresh` (using masking thresholds).

`Post Threshold`. Sets the SNR threshold in dB in which the post-filter will start to blur musical noise. It can be a positive or negative dB value in between -10 dB and 10 dB.

8.4 The AGC Menu

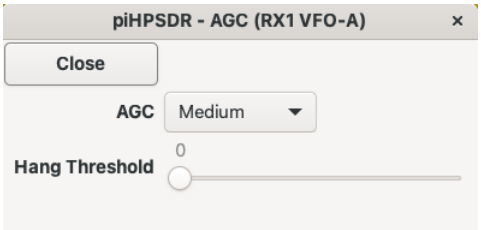

Fig. 8.9: The `AGC` menu.

Only few parameters can be controlled via the automatic gain control (AGC) menu. The first is the AGC time constant, which can be Off (no AGC), Long, Slow, Medium, and Fast. A very long AGC time constant protects your ears, but it also means that the receiver becomes ,,deaf" for a rather long time after a strong QRM burst. This phenomenon is known as "AGC pumping". Generally, if you do SSB on a quiet band, the AGC time constant can be longer, for CW on the other hand, I personally prefer short time constants (Medium or Fast).

The `AGC Hang Threshold` is only effective if the AGC time constant is Long or Slow, since the AGC hang time is turned off for Medium and Fast. In this case, the RX spectrum scope not only shows the ,,normal" AGC line in green, but also the hang threshold line in orange.

8.5 The Diversity Menu

`Diversity` is a very powerful tool to improve reception by using two different antennas and two ADCs. To explain how it works, suppose you live in a house

which produces a lot of local QRM. Your ,,normal" antenna will pick up the wanted DX signals, but also a lot of noise that originates somewhere in your house. Now suppose you have a second receive-only antenna placed in your house that will predominantly pick up your local QRM and only very little DX signal.

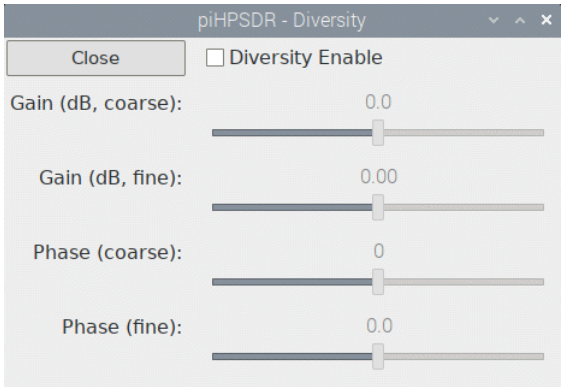

Fig. 8.10: The Diversity menu.

Of course, this RX-only antenna does not deliver anything useful at first sight. But, imagine you could shift the phase and the amplitude of the signal of the in-house antenna such that it exactly opposes the local QRM picked up by your DX antenna! Adding this (phase shifted and amplitude adjusted) signal from your in-house antenna to what comes from your DX antenna will produce a signal where the local QRM is largely eliminated while the DX signal is only weakly affected. This is what Diversity is all about.

Chapter 9

The Main Menu: TX-related menus

Note that for RX-only radios, only the CW menu will be shown here because there one can set the pitch of the CW side tone, which also affects the RX ,,BFO frequency".

9.1 The TX Menu

The TX menu can be opened from the main menu, or just by a secondary mouse click into the TX pan-adapter (while transmitting). The menu is shown in Fig. 9.1. This menu has five sub-menus, called the TX Settings for general transmitter settings, the CFC settings for controlling the continuous frequency compressor (CFC), the DExp Settings for controlling the downward expander (DEXP), the Peak Lables settings for controlling if and how peaks are labelled with their dBm values, and finally the TUNE and SWR settings with options for TUNE-ing and SWR control. With the radio buttons in the top row of the menu you can switch between the sub-menus. When you open the TX menu for the first time after starting piHPSDR, the TX settings are shown. Subsequently opening the TX menu always shows the sub-menu that was on display when the TX menu was closed the last time.

9.1.1 TX Menu: TX Settings

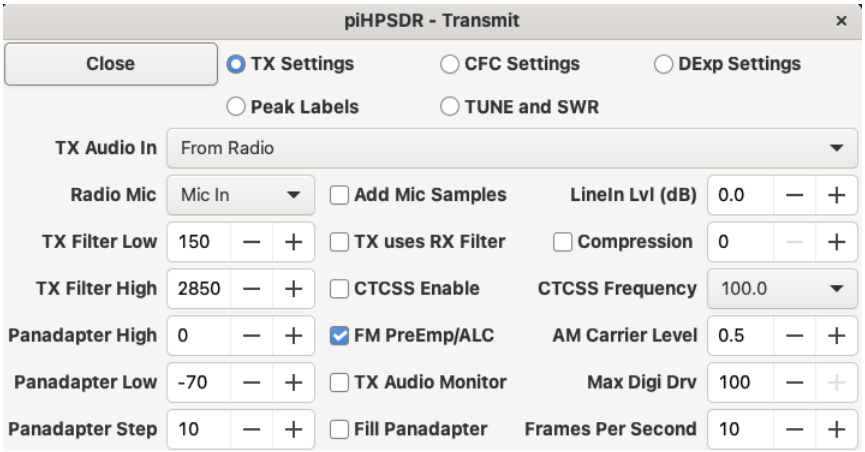

Fig. 9.1: The TX menu with the TX settings.

TX Audio In. This selects the audio input device used for the transmitter. The choice From Radio means that audio samples sent from the radio are used. This is only possible for HPSDR radios which have an audio codec, that is, HPSDR radios which have a jack for plugging in a microphone. All other radios must use a sound device of the computer running piHPSDR (usually, a USB sound card or head-set). Note that with PulseAudio/PipeWire, the names of audio input devices can be quite long, therefore much horizontal space has been allocated to the drop-down menu.

This setting is part of the RX/TX profile (see Chapter 14). Most users will choose a device connected to a microphone for SSB, while choosing a virtual audio cable or a sound card connected to another computer running the digi mode program when doing digi. piHPSDR then automatically switches the audio input device when switching modes.

The next line only appears for HPSDR radios which have a built-in audio codec and thus can send either microphone or line-in audio in the HPSDR data stream.

Radio Mic. The pop-down menu to the right of this text lets you choose between Mic In which means that a microphone can be connected to the microphone input jack, Mic Boost, which additionally switches on a hardware 20 dB mic amp, and Line In which means that the "Line In" jack

of the radio is used for the audio samples transferred from the radio to the host computer. This is part of the HPSDR protocol, it may well happen that your radio has a microphone jack but no line-in input. The optional 20 dB preamp may be necessary when connecting a dynamic microphone whose input level (few mV) is considerably lower that that of a condenser (electret) microphone, or a dynamic microphone with built-in preamp.

`Add Mic Samples`. This option only has an effect if the TX input audio comes from a soundcard or a virtual audio cable, as selected in the `TX Audio In` pop-down menu. Checking `Add Mic Samples` then *adds* the audio samples from the HPSDR data stream to the TX audio in sample, but only when there is a hardware PTT signal from a switch connected to the radio, e.g. the PTT button of a microphone or a PTT foot switch. This option has been added to support voice keyers, that is, to allow logbook or contest programs to send TX audio, usually via a virtual audio cable. For such a setup, choose that virtual audio cable in the `TX Audio In` menu. As long as you press the PTT button at a microphone connected to the radio, you can use that microphone for transmitting. It is assumed that in this setup, you do not use the voice keyer for transmitting while you press PTT. Note that VOX cannot be used with this setup, since the audio data from the microphone is not used if PTT is not pressed. This is to prevent your microphone picking up noise and adding this to the TX audio data while running digi mode.

`LineIn Lvl (dB)`. Here you can adjust the line-in level of the radio, if it has one and if `LineIn` is selected with the `Radio Mic` check box.

Note: The controls in the second line (`Radio Mic` and `LineIn Lvl`) only apply to (and are only shown for) HPSDR (Protocol-1 and Protocol-2) radios. If such a radio does not have an audio codec, the controls are shown but will not have any effect.

`TX Filter low`. With this spin button you can set the low cut of the TX filter. The frequency refers to the audio domain.

`TX Filter high`. With this spin button you can set the high cut of the TX filter. The frequency refers to the audio domain.

`TX uses RX Filter`. If this check box is enabled, the TX filter low/high cuts are ignored, and the filter edges of the current RX filter are used instead. The `TX Filter Low` and `TX Filter High` check boxes will be inactive in this case.

Note:TX filter settings have no effect in FMN mode. All the filter characteristics are then calculated from the deviation for a maximum audio frequency of 3000 Hz.

`Compression`. With this box the TX compressor can be enabled/disabled. The compression level (0-20 dB) can be chosen in the spin button to the right. Setting TX compression on/off automatically engages/disengages the auto leveller (with a maximum amplification of 8 dB), and using the compressor with a compression level > 5 automatically activates the CESSB overshoot control.

Note: The compressor on/off flag, as well as the compression level, is part of the RX/TX profile (see Chapter 14). So it is possible to have the compressor enabled for SSB (LSB/USB) and disabled for digi modes (DIGL/DIGU), and when switching modes, the compressor settings for the new mode are automatically restored. The compressor can be used (although this might not be recommended together with the CFC).

`CTCSS Enable`. (FM only!) This checkbox enables/disables CTCSS (continuous tone coded squelch system). If enabled, a low-frequency tone is transmitted together with the normal TX audio. This can be used to trigger repeaters, or any other function implemented on the other side. The frequency itself can be chosen with the following menu point:

`CTCSS Frequency`. This pop-down menu lets you choose the CTCSS frequency. This choice has no effect if CTCSS is disabled. The frequency list includes 38 standard TIA/EIA-603-D CTCSS frequencies between 67.0 and 250.3 Hz.

`FM PreEmp/ALC`. When transmitting FM, the audio input signals are ,,emphasised". This means that from 300 to 3000 Hz (the usual range of audio frequencies in amateur radio FM), there is a frequency-dependent (6 dB per octave or 20 dB per decade) attenuation that leads to a 20 dB damping of an input signal at 300 Hz (8 dB damping at 1200 Hz, and no damping at 3000 Hz). This of course distorts the audio, but the reverse process is built into FM demodulators to correct for this. Because there is a lot of damping of the audio signal, piHPSDR automatically applies a 15 dB boost to the TX audio input samples when the mode is FMN.

This boost is nearly ineffective if the FM pre-emphasis takes place *after* the TX ALC stage, since the ALC will cancel most of the extra boost and the

FM modulation sounds ,,thin". If the FM PreEmp/ALC box is checked, FM pre-emphasis takes place *before* the TX ALC, such that the ALC ,,sees" the TX audio input after applying both the boost and the damping of the pre-emphasis. This gives the transmitted signal a little more ,,punch". It is generally recommended to have this box checked if doing FM.

AM carrier level. This sets the AM carrier level for the AM modulator. If set to zero, there is no carrier and the signal is a DSB signal. A reasonable value is 0.5 which leads to 100% modulation. Values larger than 0.5 have less than 100% modulation. This means too much power goes into the carrier.

TX Audio Monitor. When this box is checked, the TX audio input (e.g. from the microphone) is mirrored on the audio output of the active receiver. The volume setting of the active receiver is applied. This does not work in CW (where the audio output is reserved for the side tone) or during TUNE-ing when the SWR dependent TUNE side tone is activated. Note that this only monitors the TX audio at the beginning of the TX audio chain (see Chapter 15).

Note: The TX audio monitoring does not work when operating DUPLEX, in this case the active receiver is running during TX and feeds the audio output.

Max Digi Drv. This spin button restricts the range of the drive slider from 0 to the chosen value for the DIGU and DIGL modes. If the value is 100, this has no effect. The primary use of this menu point is PA protection, since many digital modes (unlike SSB voice) are constantly transmitting at full power.

Panadapter High. This spin button sets the upper edge (in dBm) of the TX pan-adapter.

Panadapter Low. This spin button sets the lower edge (in dBm) of the TX pan-adapter.

Panadapter Step. This spin button determines how many horizontal lines are drawn on the TX pan-adapter. If set to 10, for example, there will be a horizontal line for every multiple of 10 dBm.

Fill Panadapter. This is used to enable/disable the ,,Filling" option for the TX spectrum scope (see chapter 4.4). No ,,gradient" option is available for the TX scope.

Frames Per Second. This spin button determines how many frames per

second are drawn for the TX pan-adapter. The default value, 10, is a good choice.

9.1.2 TX Menu: CFC Settings

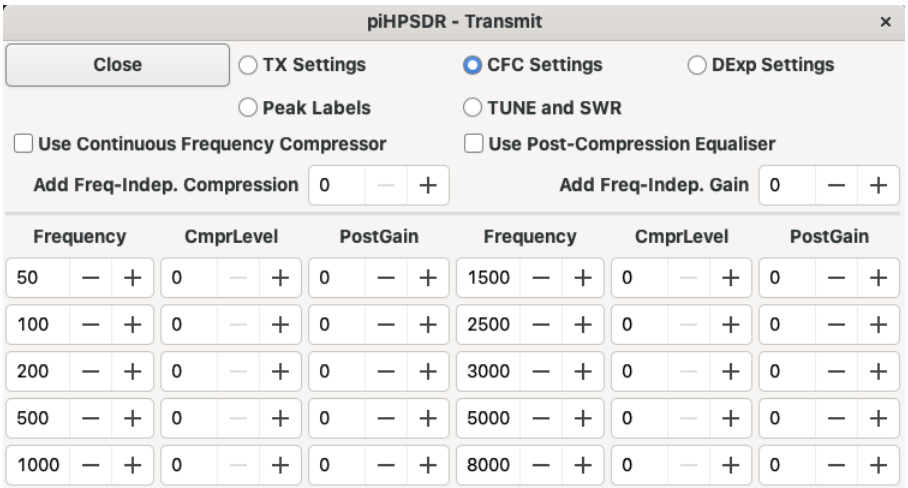

Fig. 9.2: The TX menu with the CFC settings

In the CFC sub-menu, the settings for the continuous frequency compressor can be made. As the name indicates, the compression can be chosen frequency dependent. As for the equaliser, one can specify corner frequencies with a given compression, between corner frequencies the compression is interpolated linearly. In addition, there is an equaliser that is applied after compression, so one give a gain (or attenuation) for each corner frequency, with linear interpolation in-between. Gains in this menu are in dB and can be chosen between -20 and +20 dB, compression levels are also in dB and can be chosen between 0 dB (no compression) and +20 dB.

`Use Continuous Frequency Compressor`. This checkbox is used to enable/disable the CFC. As with the speech processor, using the CFC automatically enables the TX auto-leveller but note CESSB overshoot control is *not* enabled automatically.

`Use Post-Compression Equaliser`. This checkbox is used to enable/disable the post-compression equaliser.

`Add Freq.-Indep. Compression`. With this spin button, one can specify an *additional* frequency-independent compression that applies to all frequencies.

`Add Freq.-Indep. Gain`. If using the post-compression equaliser, this spin button specifies an *additional* frequency-independent gain applied to all frequencies.

In the lower part of the menu, you find ten groups of three spin buttons that specify a frequency/level/gain triple. The compression level and gain given (if the post-compression equaliser is used) applies to that corner frequency while linear interpolation is used between corner frequencies.

9.1.3 TX Menu: DEXP Settings

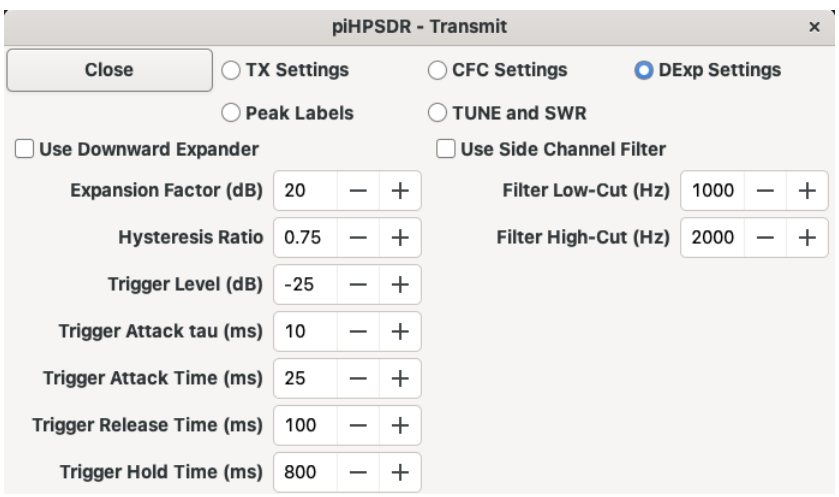

Fig. 9.3: The TX menu with the DEXP settings

In the sub-menu for the downward expander, you can control the settings for the downward expander (DEXP). This can be viewed as a configurable noise gate that applies to the TX audio input. Its main use is to suppress low noise (e.g. from fans) that are taken up by the microphone in speech pauses. The downward expander damps such noise as long it is below a trigger level. Once the ,,noise" exceeds the trigger level (that is, the operator speaks the next word or so) this damping is switched off. Sometimes, the ,,noise" is rather loud, such that noise alone leads to triggering, and in many cases such noise has rather low frequencies. This is where a side channel filter comes in: this

filter specifies a range of frequencies used for triggering. For normal human voice for example, one expects that frequencies between 1000 and 2000 Hz are contained therein so one can only use these for triggering.

`Use Downward Expander`. With this check box, the DEXP can be enabled or disabled.

`Use Side Channel Filter`. With this check box, the side channel filter can be enabled of disabled. With the side channel filter enabled, only TX audio in the defined frequency range will trigger the noise gate.

`Expansion Factor (dB)`. This factor (in dB!) determines how strongly the noise will be damped as long as it is below the trigger. The default (20 dB) should be OK in most cases.

`Hysteresis ratio`. If this value is smaller than 1 (say, 0.75), then the hold time (the time in which the noise gate is open starts when the input level drops below the trigger level, multiplied with the hysteresis ratio.

`Trigger level (dB)`. This spin button specifies the trigger level. It is given in dB relative to full-amplitude audio samples. The default level (-25 dB, corresponding to an amplitude of 0.056) should be OK in most cases.

`Trigger Attack tau (ms)`. This is the time constant (in ms) for averaging the input signal before it goes to the detector.

`Trigger Attack Time(ms)`. This is the time used for opening the noise gate. To avoid plops in the audio, the gate is not opened instantly but with a raised cosine characteristic. This time should not be overly long, since otherwise parts of the first word spoken by the operator can get rejected by the gate.

`Trigger Release Time (ms)`. This is the time used for closing the noise gate. To avoid plops in the audio, the gate is not opened instantly but with a raised cosine characteristic. The time for closing the gate can usually be chosen much larger than for opening.

`Trigger Hold Time (ms)`. After the operator has finished his speaking, the noise gate remains open for some time (e.g. waiting for the next word). Typical hold times are of the order of 1 second, when no new triggering audio arrives within that time, the gate is closed and the noise thus suppressed.

`Filter Low-Cut (Hz)` and `Filter High-Cut (Hz)` specify the frequency edges of the side channel filter (if it is enabled). The defaults given here

(1000–2000 Hz) are well suited for distinguishing human voice from humming noise coming from fans etc.

9.1.4 TX Menu: Peak Labels

The "Peak Labels" sub-menu allows one to control if and how labels (values in dBm) are printed at the most prominent peaks on the TX panadapter (Fig. 9.4):

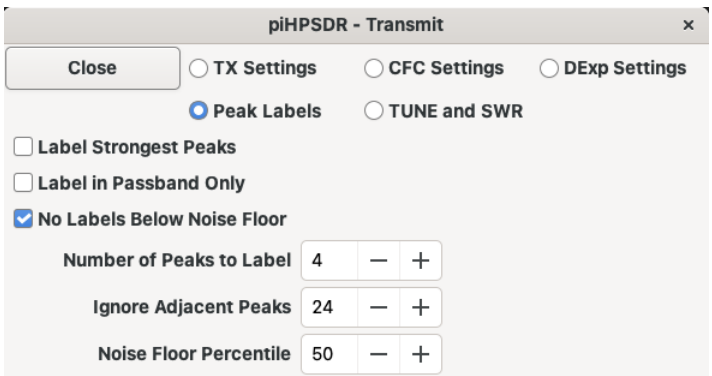

Fig. 9.4: The TX menu with the peak label settings

`Label Strongest Peaks`. This enables peak detection and labelling.

`Label in Passband Only`. If the box is checked only the peaks within the current filter pass-band are shown.

`No Labels Below Noise Floor`. This suppresses labelling of peaks below the noise floor.

`Number of Peaks to Label`. Set maximum number of peaks to be shown. There can always be less, but no more than this number. The strongest peaks will be labelled first.

`Ignore Adjacent Peaks`. Determine how close two peaks can be such that both are labelled. The higher the number the closer the peaks can be to each other. It's a divisor of the full window width so setting it to 1 will mean only one peak can occupy the entire window,

`Noise Floor Percentile`. This is the percentile setting that defines the noise floor. If the percentile is 50, then the noise floor is the level of that

pixel that is in the middle of the list if all pixels are sorted by their dBm value (a safety margin of 3 dBm is added). If the `No Labels Below Noise Floor` box is ticked peaks below this value will not be labelled.

9.1.5 TX Menu: TUNE and SWR

This sub-menu allows to specify the TX drive level while TUNE-ing. For HPSDR radios which report forward and reverse power (and thus the SWR), some SWR-dependent controls are added as well.

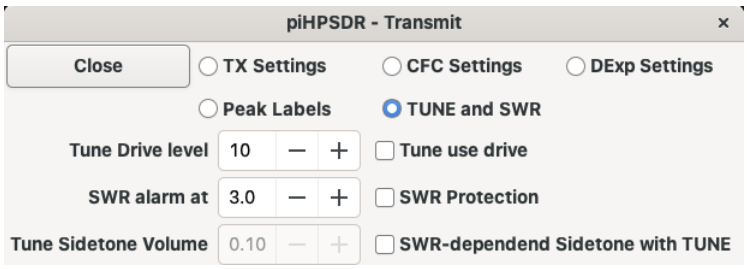

Fig. 9.5: The TX menu TUNE/SWR settings

`Tune use Drive`. If this box is checked, TUNE-ing will be done with the power that corresponds to the current position of the drive slider, and the tune drive level is ignored.

`Tune Drive Level`. The value that can be adjusted with this spin box is the virtual position of the drive slider while TUNE-ing. This value is ignored (and the spin-button made inactive) if the `Tune use Drive` box is checked.

`SWR protection`. If this box is checked, a very simple SWR protection is enabled. If the SWR exceeds the threshold value (see next point), the drive slider is set to zero. The SWR protection is disabled while TUNE-ing.

`SWR alarm at`. The spin button to the right determines the SWR threshold. If the SWR is beyond the threshold, the SWR reported in the meter turns red. If SWR protection is enabled, the drive slider is set to zero if the SWR exceeds the threshold.

`SWR-dependent Sidetone with TUNE`. If this box is checked, there will be a side tone while TUNE-ing. This side tone is similar to a string of dots or dashes, separated by pauses which are 50 msec long. The SWR is encoded

both in the length and the frequency of the "dashes". The higher the SWR, the higher (with a limit of 5000 Hz) the pitch, an ideal 1:1 SWR results in a minimum pitch of 500 Hz. Likewise, the "dot/dash" length is 50 msec for this ideal SWR and becomes longer as the SWR gets larger. So for ideal SWR, the side tones sounds like a string of CW dots at 24 wpm at a pitch of 500 Hz. This "dot length" increases to 125 msec (with a pitch of 725 Hz) if the SWR is 1:1.5, and to 500 msec (with a pitch of 1400 Hz) for an SWR of 1:3. The idea of this feature is that one can operate a manual antenna tuner without looking at the screen.

`TUNE Sidetone Volume`. With this spin-button, one can adjust the volume of the SWR-dependent TUNE side tone (allowed range: $0 \ldots 0.5$).

Attention! Always start with small values to protect your ears. This spin-box is inactive is the `SWR-dependent Sidetone with TUNE` is not active.

9.2 The PA Menu

In the PA menu, you can adjust the output level of your HPSDR board to the PA being used, and you can establish a calibration of the power that is displayed in the meter section while transmitting. The menu presents itself as shown in Fig. 9.6.

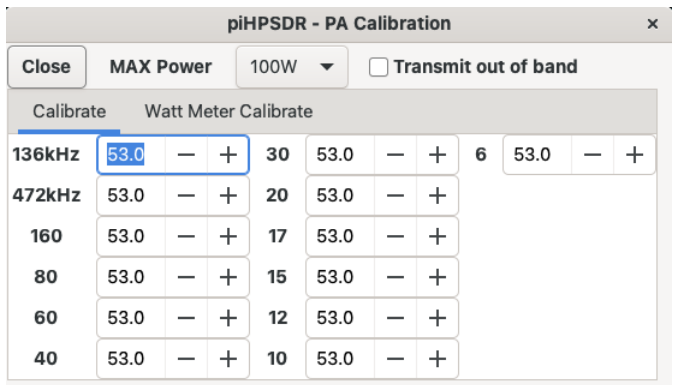

Fig. 9.6: The PA menu, PA calibration screen

In the first line, you can choose the maximum PA power of your radio. The available values are 1, 5, 10, 30, 50, 100, 200, and 500 Watt. If your radio has a different maximum power, choose the next largest value. The choice of this value only affects the watt meter calibration (see below). If the box Transmit out of band is checked, this allows piHPSDR to go TX if you are outside of the amateur radio bands.

9.2.1 PA calibration.

If the Calibrate sub-menu is active (as shown in Fig. 9.6) you can adjust your HPSDR board to your PA. This has to be done for each band separately, and you need a dummy load and a watt meter to do so. Most watt meters used by radio amateurs are not highly accurate, so if you can borrow an accurate one, do so. The PA calibration values are the fictive amplification of the PA. If the value is *increased*, piHPSDR assumes a higher amplification and will thus *decrease* the output power of the HPSDR board. Thus, *increasing* the PA calibration value will *decrease* the output power. A calibration value

of 38.8 dB corresponds to the maximum RF output of the HPSDR board, so the allowed range of values starts at 38.8. The default is a calibration value of 53.0 dB, which usually leads to a low-power RF output. The #1 problem of new piHPSDR users is that they do not do the PA calibration as described below and therefore get too low or even almost no RF output power.

To start calibration, go to the TX menu and check the box Tune use Drive. Then, hit the rightmost toolbar button until one of these buttons reads TUNE. This way, when TUNE-ing, you send a carrier with the power according to the drive slider. For each band, go to the middle of the band, open the PA menu, put the drive (TX Drv slider) at 50 and hit the TUNE button. If the output (measured with the Watt meter) is higher than half of your nominal PA power, increase the PA calibration value of that band, otherwise decrease it. Choose a value such that your Watt meter reads half the nominal output power. For fine adjusting, move the drive slider to 100 and adjust the PA calibration value until your Watt meter shows the nominal output power. The calibration values will (slightly) differ from band to band, often one needs smaller values for the higher bands since the amplification of the PA is smaller there.

Of course, what you do here is not to change the amplification of your PA, but to change the level of the low-power SDR board RF output that goes into the PA. Therefore, PA calibration is also effective for the transverter bands have been defined in the XVTR menu (Chapter 6.5), and which also show up in the PA menu. This is especially useful if the transverter is driven directly from the low-power RF output from the Xvtr port (in this case, you should disable the PA anyway, see the XVTR menu.

9.2.2 Watt meter calibration.

Note first that not all radios sense the RF output power and report it in the HPSDR data stream. I this case, RF output power in piHPSDR will be displayed constantly as zero, and also the SWR sticks at 1:1. In this case, you can skip this section.

Now you can calibrate the power reading within the meter section of the piHPSDR window. If you open the PA menu and click on the text Watt Meter Calibrate, the menu changes and looks like in Fig. 9.7.

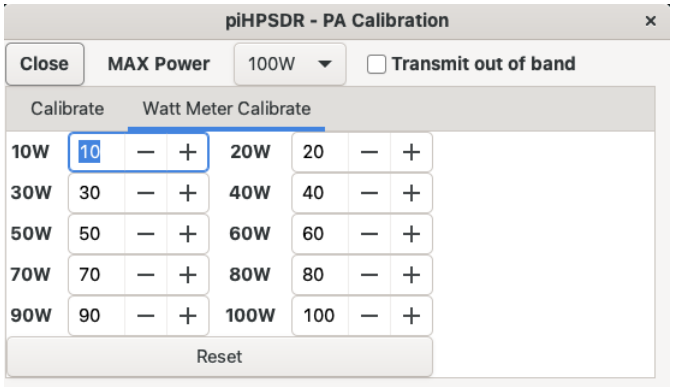

Fig. 9.7: The PA menu, Watt meter calibration

Note that for calibrating the Watt meter as well, `Tune use Drive` in the `TX` menu must be checked to allow for high RF output power while TUNE-ing.

You have 10 values from $\frac{1}{10}$ to the full nominal power. Initially, the values of the spin buttons beside the Watt ratings have the nominal value. The calibration values can always be re-set to these nominal values by hitting the `Reset` button. Watt meter calibration is not done separately for all bands, so it is suggested to perform the following procedure on the 20m band. piHPSDR will convert the ,,measured" into the ,,reported" value by linear interpolation between two adjacent calibration values.

Start with resetting the value by hitting the `Reset` button. Then, move the drive slider to 100 (the unit of the drive slider is per cent, not Watt!) and hit `TUNE` in the toolbar. After the PA calibration described above, your (external) Watt meter should show the nominal PA output power (e.g. 100 Watt for an Anan-7000). Now look at the forward power reported in the meter section (top right of the piHPSDR window). Suppose you read "250 W" there although your output is 200 Watt. Then simply insert the number 250 in the spin button to the right of the string `200W`. Now your watt meter reading should be close to 200W, you can fine-tune it with the spin button. Note that *increasing* the calibration value with the spin button will *decrease* the power indicated in the meter section.

You will observe that the calibration values for the lower powers also have changed. This only happens if you start from nominal calibration values and change the calibration value of the highest power. For example, if you

have entered 250 in the 200W spin button, then the value in the 100W spin button will read 125. So in a single shot, you have roughly calibrated the Watt meter.

A finer calibration only makes sense if you have a highly accurate Watt meter, since the (uncalibrated) reading in piHPSDR may actually be more accurate than your Watt meter. Using your highly accurate Watt meter, you can now move the drive slider until your Watt meter exactly reads one of the lower power values, and use the corresponding spin button to change the calibration until piHPSDR exactly reports the correct power. The procedures is virtually the same if our nominal output power is different. The only complication arises if your radio has a nominal power that is not in the menu, for example 150 Watt.

In this case, choose 200W (the next largest value) in the top line of the PA menu. TUNE and move the drive slider until your Watt meter reads the largest value possible that occurs in the Watt meter calibration menu (in this example, it is 140W). Adjust the 140W spin button until piHPSDR reports 140 Watt. Then go to full power (150W) and adjust the 200W spin button until piHPSDR reports 150 Watt. Then, proceed with 120, 100, 80, etc. Watts.

9.3 The VOX Menu

VOX (voice control) means that you can just speak into the microphone and the radio goes TX, without the need to press a PTT button. VOX can also be used in digital modes, if there is no possibility that the digi-mode program can put piHPSDR into TX mode via CAT commands or hardware lines. The VOX menu is shown in Fig. 9.8.

With the VOX Enable check box, you can enable/disable VOX. For VOX operation, there are two parameters, namely the VOX threshold and the VOX hang time. The VOX threshold is the microphone amplitude required to trigger a RX/TX transition. If the radio goes TX when the neighbour's hound starts barking, then the VOX threshold is too small. If the radio does not go TX although you speak loudly into the microphone, the threshold is too large. The VOX menu features an indicator which can be green or red (in Fig. 9.8, this is the green bar). This indicator flashes red if the microphone

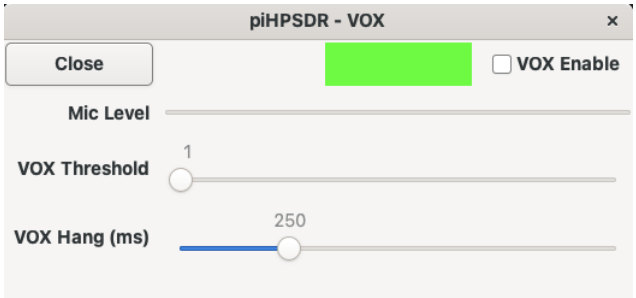

Fig. 9.8: The VOX menu

amplitude is above the VOX threshold. Adjust the threshold with the slider such that the indicator becomes red if you speak into the microphone, but stays green if you don't speak.

The VOX hang time determines how long the radio stays in TX mode after the last time the microphone delivered a signal that was above the VOX threshold. Typical values are 250 to 500 ms. If your radio produces relay chatter because it goes RX between your words, increase the hang time. However, this will also increase the turn-around between you finished your message and go RX.

VOX is very nice for rag-chew phone QSOs, I won't recommend it for contest operation.

9.4 The PS (PureSignal) Menu

PureSignal is the ,,street name" for adaptive pre-distortion. What this means is, that the signal from the *output* of the PA (the ,,antenna signal") is coupled back (through an attenuator of typically 40-60 dB) to the radio and is analysed whether is looks like it should. If it is distorted (e.g. by non-linearity of the PA), then the PureSignal algorithm calculates how an input signal to the PA should look like to produce the desired output. This is usually measured and calibrated with a so-called two-tone experiment. In this experiment, two constant carriers, for example 7100 kHz and 7101 kHz, are transmitted. If both carriers contain 25W power, this is a 100W PEP signal. Non-linearity of the PA first lead to the occurrence of harmonics (in this case around 14.2, 21.3, and 28.4 MHz). This is not a problem because such harmonics are

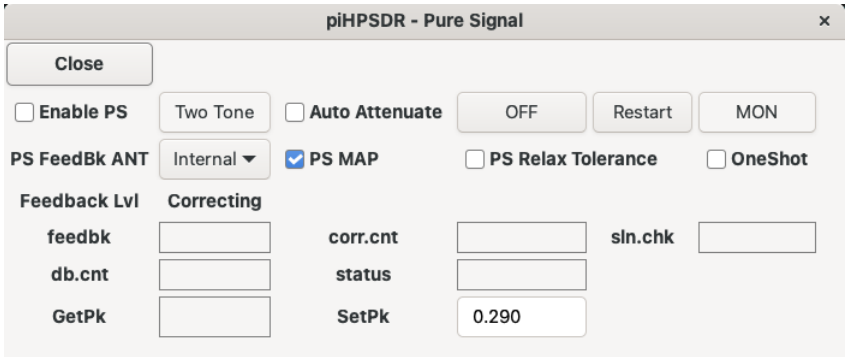

Fig. 9.9: The PureSignal (PS) menu

damped by the TX low-pass filters. Higher-order non-linear effects, however, lead to additional *in-band* signals that cannot be filtered out. In our example they occur at 7102/7099, 7103/7098 etc. kHz. Since these signals cannot be filtered out, they lead to unwanted signals („splatter") that disturb QSOs on neighbouring frequencies. With PureSignal, you can greatly reduce these unwanted signals. If you open the PS menu for the first time it looks like shown in Fig. 9.9. The elements have the following function:

Enable PS. With this check box, PS can be enabled/disabled.

Two Tone. With this button, a two-tone experiment can be started/stopped. The button will be highlighted as long as the two-tone signal is transmitted. In the lower side band modes (LSB, DIGL, CWL), an RF two-tone signal with frequencies 700 and 1900 Hz *below* the dial frequency are transmitted, in all other modes the two RF frequencies are 700 and 1900 Hz *above* the dial frequency. The same two-tone experiment can be started/stopped via the toolbar, or by a GPIO- or MIDI-monitored push button. At the beginning of a two-tone experiment, the diagnostic fields are cleared and re-populated once a valid calibration has been obtained.

Auto Attenuate. This enables/disables automatic adjustment of the RF input attenuator to give the feedback level the correct strength. It is highly recommended to use this option.

OFF. With this button, the PS correction can be stopped (the status will then change to RESET).

Restart. With this button, the PS correction can be resumed, for example

after it has been stopped.

MON. With this button, it can be chosen whether the TX spectrum scope shows the signal sent to the PA (MON button not highlighted) or whether the feedback signal from the antenna is shown (MON button highlighted). Note that feedback data is only available if PURESIGNAL is enabled. Without PS being enabled, the TX pan-adapter will show the TX signal as it leaves piHPSDR no matter if MON is checked or not.

PS Feedback ANT. Here it must be specified which antenna jack is used for the PS feedback signal. It can be Internal which means internal feedback (for example as built into the Anan-7000 or simply the cross-talk from the TX/RX relay), or it can be Ext1 or ByPass which refers to the auxiliary antenna jacks.

PS MAP. This box controls the PURESIGNAL ,,Map mode" and is normally checked. According to the WDSP manual, changing the Map mode allows easier calibration in situations where a very poor PA is driven into heavy gain compression. The state of this box has no effect if there is no compression. *PS is stopped and re-started if this box is changed.*

PS Relax Tolerance. This box is normally unchecked which means that the default value 0.8 is used for the PURESIGNAL calibration tolerance. If checked, this tolerance is reduced to 0.4. According to the WDSP manual, relaxing the tolerance my be helpful for PAs with a very poor load regulation in the power supply such that there are severe and slow memory effects. *PS is stopped and re-started if this box is changed.*

OneShot. This box is unchecked by default. In the default case, the PURESIGNAL algorithm constantly compares the transmitted and the feedback signal and thus constantly updates the calibration. If OneShot is checked, the calibration is kept when the two tone experiment is finished. There are cases, especially if CPU power is lacking, where the PURESIGNAL algorithm from time to time fails to obtain a valid calibration, and when this happens during transmitting, a bad signal may be transmitted for a short amount of time. Checking OneShot can help in such situations because the calibration, once obtained, is kept and not changed. When a successful "one shot" PURESIGNAL calibration has been achieved, the status field in the menu becomes stable and reads STAYON.

However, if something changes (e.g. output power, or the finals get warm,

etc.) the calibration is probably no longer valid but still kept. So unless you have good reason, do not use the OneShot option.

SetPk. This field shows the currently assumed value of the peak value of the TX DAC feedback signal. piHPSDR chooses is automatically, depending on the radio hardware. The standard value for Protocol-1 and Protocol-2 radios are 0.407 and 0.290. Notable exceptions are the HermesLite-II running Protocol-1 (SetPK=0.240) and the Anan-G2 running Protocol-2 (SetPK=0.612). The SetPK value is determined by the FPGA firmware of the radio can can experimentally be determined by comparing the outgoing TX and the incoming TX feedback signal. It should match the value reported by the calibration algorithm in the GetPk field. You can change the value in the SetPk field, the changed value is then also written to the props file and then restored upon the next program start. This should only be done if the radio hardware has a non-standard peak TX DAC value. One case I know of is the "Brick2 SDR", which identifies itself as a HERMES board but needs a quite large SetPk value of about 0.7800. But such cases are easily identified when performing a two-tone test by a large difference between the values in the SetPk field and the GetPk value which is obtained from analysing the TX DAC data.

Attention! Specifying a wrong value in the SetPk field will effectively disable PURESIGNAL, and since this value is stored in the props file, this adverse effect also shows up in the future, that is, until either the props file is deleted or the correct value is again entered in the SetPk field. This *caveat* also applies if you upgrade your radio firmware from protocol-1 to protocol-2, since then your props file still contains the correct protocol-1 value (0.4067) and you have to manually modify the SetPk value (in this case, to 0.2899).

TX ATT. This element can occur both as a text field or as a spin button. If PURESIGNAL is enabled while Auto Attenuate is *not* enabled, it is a spin button with which you can manually adjust the RF attenuation. For normal HPSDR radios, this is a value between 0 and 31, other radios such as the HermesLite have an extended range from -29 to 31. If the feedback level is too strong, this value must be increased, if it is too strong, it must be decreased. It is, however, recommended to enable Auto Attenuate. In this case, the TX ATT field is a text field that shows the current attenuation. Note that if PURESIGNAL is not enabled, the RF input attenuators will automatically be set to maximum attenuation while transmitting with the PA enabled.

All other fields are not meant for user input, they just give information on the status of the PURESIGNAL engine.

Feedback Lvl. When doing a PS calibration through a two-tone experiment, this string turns green if the feedback level is good. It turns yellow if the feedback level is slightly to weak and red if it is too weak. A blue colour indicates a too strong feedback level. The feedback level reported by the PS calibration algorithm is further reported in the feedbk field. The optimum value is about 154.

Correcting. When doing a PS calibration through a two-tone experiment, this string is green if calibration was successful and PS correction takes place, and the string is red if no good calibration could be made.

The other fields give more information on the PURESIGNAL engine, consult the WDSP manual. Noteworthy is the GetPk field, that reports the maximum amplitude seen in the TX feedback samples. This value remains blank or zero until PS starts working. It should be close (ideally, slightly below) the value in the SetPk field. If not, then you either have non-standard hardware where piHPSDR does not have a good default SetPK value, or the default value has been modified (probably upon startup when reading the props file). The field corr.cnt shows a running count of corrections made during the two-tone experiment. When PS is working, this number should be constantly increasing. The status field shows the current states of the PS engine, which is cycling during normal operation. If the PS engine is halted, the status becomes Reset.

If you want to use PURESIGNAL, it is required to enable PURESIGNAL by checking the Enable PS box, and it is recommended to use auto calibration (check the Auto Attenuate box). PURESIGNAL is then activated and calibrated by performing a two-tone experiment. This can, but need not be, done by hitting the Two Tone button. PURESIGNAL restarting and calibration is also done if a two-tone experiment is started via a GPIO/MIDI or toolbar button. It is necessary to repeat such a two-tone experiment each time you change the RF output power, since most likely a new TX attenuation value is needed. I also recommend to repeat a two-tone experiment after each band change. If monitoring the feedback signal (MON button, this setting remains after closing the PS menu) is enabled, a two-tone experiment also quickly shows how effective the adaptive pre-distortion is. If everything works well, you only should see two peaks on the TX pan-adapter during the

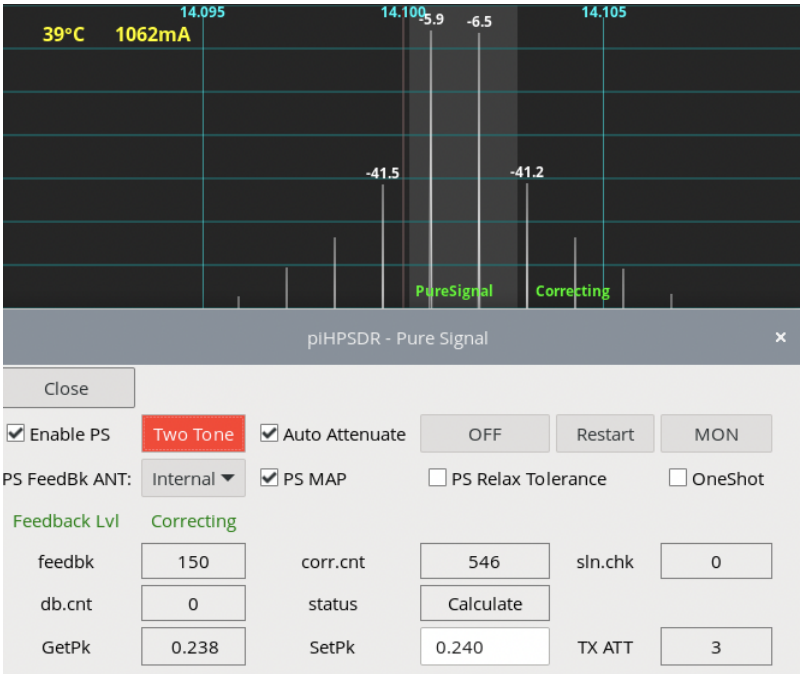

Fig. 9.10: PS: TwoTone without MON

two-tone experiment.

To demonstrate how PURESIGNAL works, I show an example performed with a HermesLite-II radio running Protocol-1. The radio runs at a sample rate of 192 kHz, since it is my experience that PURESIGNAL does not work too well in Protocol-1 when it is run at 48 kHz (with 96 or 192 kHz I never see such problems). Note that for Protocol-2 radios the PURSESIGNAL engine runs at the fixed TX sample rate (192 kHz) and is completely independent on the sample rate of the receiver(s). I also activated the optional "peak labelling" in the `TX Menu` such that the four main peaks heights are also printed (in dBm) on the screen. Checking both `Enable PS` and `Auto Attenuation`, and hitting the `Two Tone` button, it needs only few seconds to stabilise and then Fig. 9.10 results, where the TX spectrum scope and the PS menu window have been arranged such that they do not cover each other. Although both the PS menu and the spectrum scope state that PS is working and correcting, the signal does not look good: a two-tone signal should only have two peaks, but here one sees the two main peaks at −6 dBm and two IM3 satellites at about −41 dBm (this amounts to an IM3 value of only −35

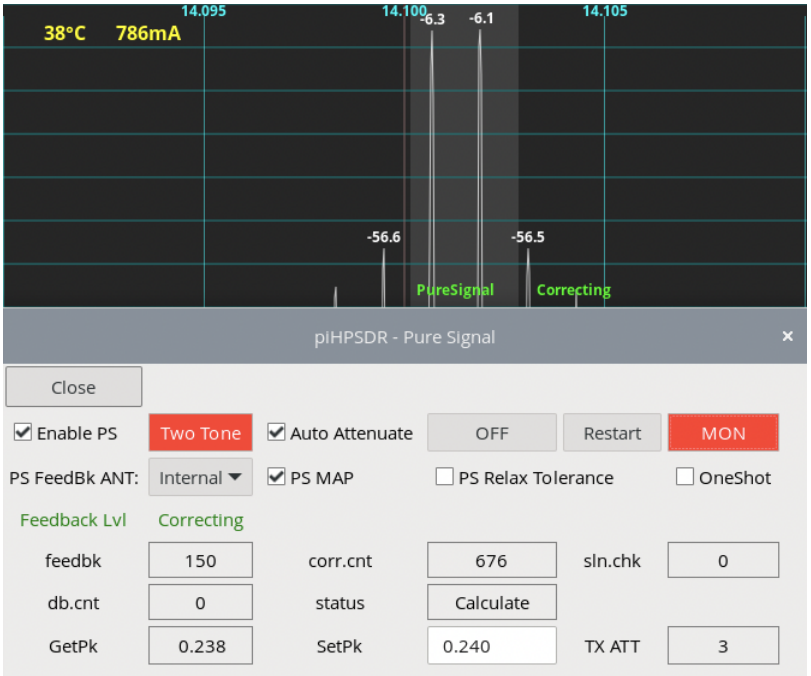

Fig. 9.11: PS: TwoTone with MON

dBc). The reason for this seemingly bad performance is of course, that the TX spectrum scope normally shows the signal that is *sent* to the PA, so we see a signal that is distorted on purpose, and magically exactly such that the PA makes a nice signal out of this (*adaptive pre-distortion*). If one wants to see what the antenna is actually transmitting, one must activate the MON button such that it is highlighted. This is shown in Fig. 9.11, where one sees the feedback signal, that is, what the PA sends to the antenna. This is a much cleaner two-tone signal (the IM3 value is about −57 dBc). To demonstrate how effective the PS algorithm is, I have pushed the OFF button which stops the PureSignal correction, the result is shown in Fig. 9.12. After hitting that button, IM3 satellites immediately grow and IM3 decreases to about −35 dBc which reflects the intrinsic non-linearity of the PA. Note that this is still a quite acceptable value: unless you have a class-A amplifier, your PA probably won't be much better. This experiment demonstrates that adaptive pre-distortion is a mechanism that allows you to produce a clean signal with a quality that would be impossible (or *very* difficult) to achieve with traditional hardware.

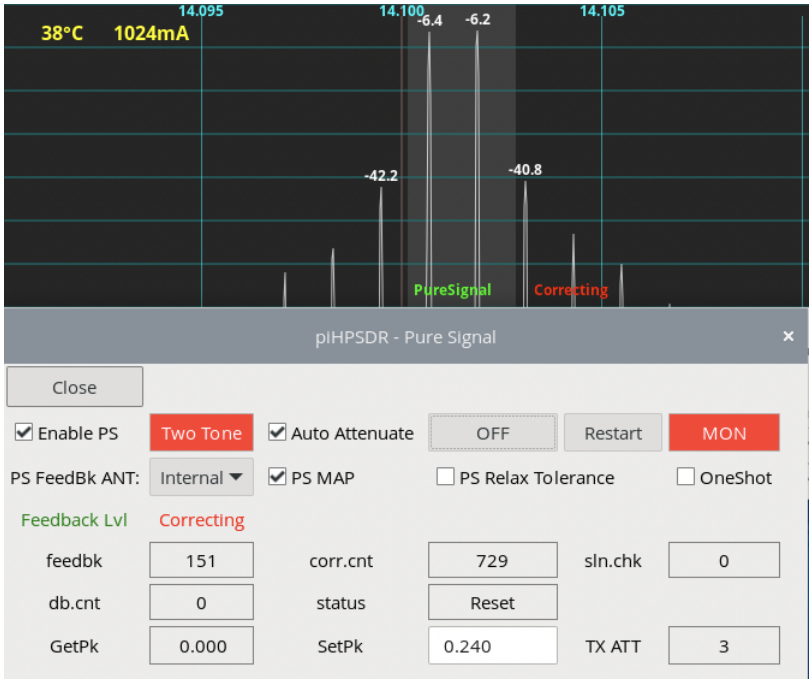

Fig. 9.12: PS: after hitting OFF

9.5 The CW Menu

The CW menu controls parameters related to CW operation, and can be used to alter or set pre-defined CW texts. The menu as it opens is shown in Fig. 9.13. In the first row, besides the `Close` button, there are two buttons `CW Options` and `CW Texts` which change the menu from a layout where one change change the CW options to a layout where one can alter the pre-defined CW texts. Those texts can be sent as CW through the piHPSDR commands `CW Txt1` through `CW Txt5`. These commands can be associated with a GPIO/MIDI/Panel push button or with a button in a toolbar.

9.5.1 Changing CW options

The layout of the CW options tab is shown in Fig. 9.13. Many radios have a connection for a paddle or at least for a straight key, and contain firmware to do CW. CW handling by the radio firmware is enabled by the

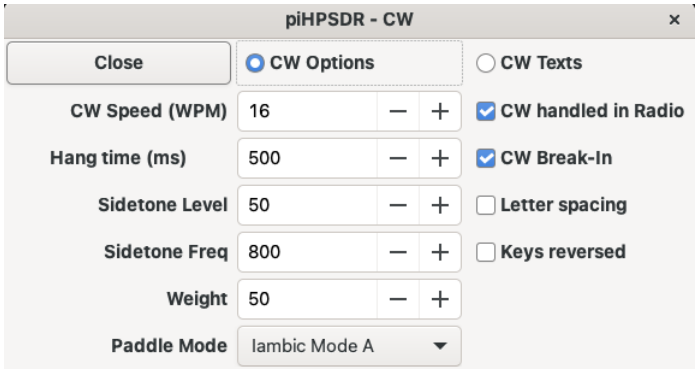

Fig. 9.13: The CW menu – Options tab

`CW handled in Radio` check box. If unchecked, CW (that is, generating and forming the RF pulses) is done by piHPSDR. For most radios, you can still use the Morse paddle connected to the radio since the radio sends the dash/dot paddle press events to the host computer, but it is more versatile to connect a Morse paddle, straight key or an external keyer is connected to the host computer (see Appendix E). Further controls are:

`CW Speed`. Here the speed (in wpm) can be chosen. If CW is handled in the Radio, the value is simply sent to the radio firmware, which implements the (iambic) keyer. If CW is done within piHPSDR, this value is used by piHPSDR's built-in iambic keyer. If using a straight key, or an external keyer whose output is then treated like a straight key, the speed has no meaning.

In either case, this speed is operative when sending CW text via CAT commands (KY command), and it can also be changed by CAT (KS command).

`CW Break-In`. This implements some sort of ,,CW-VOX". In break-in mode, the radio is automatically switched to TX when a key or paddle is pressed. Without break-in, the radio has to be manually put into TX mod.

`Hang time` While TXing in break-in mode, this spin button specifies the time the radio goes RX after the last Morse key closure. Unfortunately, this is an absolute time that does not depend on the CW speed. This cannot be changes as it is part of the HPSDR protocol.

`Sidetone Level`. This is the level of the side tone, either generated by the radio (if CW is handled there and the radio has an audio codec) or by piHPSDR (if CW is not handled in the radio). The allowed range is 0–127,

typical values are between 10 and 20. The side tone level is usually set to zero if, for example, a low latency side tone is produced outside piHPSDR. In this case, one usually wants to hear the tone from CAT CW messages being sent, so if the side tone is zero, a default side tone level (12) is used while transmitting via CAT.

`Letter spacing`. This option forces you to give ,,cleaner" CW when in iambic mode. If at the end of an inter-element pause no key is pressed, then there is a forced additional pause of two times a dot length. While this prevents you from sending too short spaces between two letters, it might well corrupt a letter you want to send. For example, when sending the letter "X" and the dot paddle is pressed a little too late, you instead send "TU". This option is probably more useful for practising than for doing real QSOs.

`Sidetone Freq`. This is the frequency of the side tone and the ,,BFO offset". That is, if a CW signal is received exactly at the dial frequency, the CW audio signal has this pitch. Note that unless one uses XIT, the transmitted CW signal is exactly on the VFO dial frequency as well.

`Keys reversed`. When checking this box, the dot and dash contacts are reversed, so you need not re-wire your paddle.

`Weight`. If using a iambic keyer (either in the radio or the builtin keyer), this value (0-100) determines the dash/dot ratio. The normal value is 50, which means that a dash is three times longer than a dot. The dash length is proportional to this value, so it can be from zero to six times the dot length.

`Paddle Mode`. Here the choice is Iambic Mode A, Iambic Mode B, and Straight Key. In Straight Key mode, the key has to be connected to the dash paddle, since the built-in keyer implements a bug mode there (automatic dots from the dot paddle, straight key behaviour for the dash paddle). When using an external keyer, use StraightKey mode and connect the keyer output to the dash paddle input.

9.5.2 Changing CW texts

If the `CW texts` tab is activated, the menu changes to the layout in Fig. 9.14.

It contains five text entry fields (marked `CWtxt1` through `CWtxt5`) in which texts up to 255 characters can be entered. There is an additional text entry

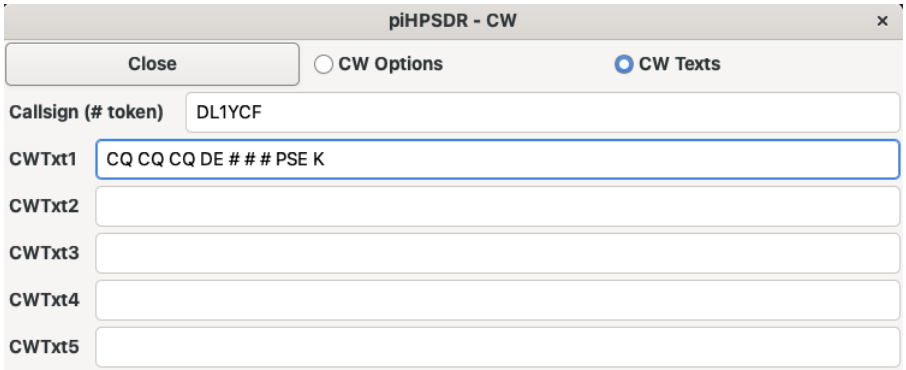

Fig. 9.14: The CW menu – Texts tab

field marked `Callsign` where you can also enter a text with up to 255 characters. This can be any text but the intended use is to enter a call sign there. When transmitting any of the five CW texts, a ,,#" character contained in these texts will be expanded by the text in the `Callsign` field. For the example shown, pressing a `CWtxt1` button will transmit a CQ call for DL1YCF. Without too much typing, the call sign can be transmitted repeatedly, and if another operator takes over, only the `Callsign` field need be changed.

As for CAT generated CW, aborting a running transmission of a CW text can be performed by hitting the morse key or paddle. If no such key is attached to the radio, switching to a non-CW mode also drains the buffer containing the text to be transmitted.

Chapter 10

The Main Menu: menus for RX and TX

10.1 The DSP (Signal Processing) Menu

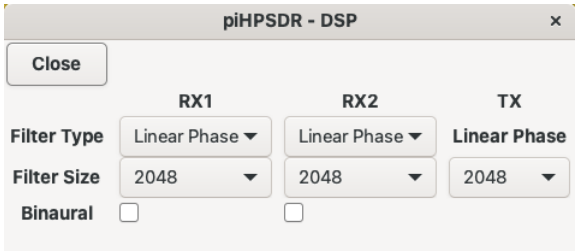

Fig. 10.1: The DSP menu.

The DSP menu sets parameters related to DSP (digital signal processing) within the WDSP library. Filter characteristics can be specified separately for the RX1 (and RX2, if running two receivers) and TX (if the radio does have a transmitter) filters. In addition, enabling/disabling binaural receiver audio is done here. Most users will very rarely need to invoke this menu, which is shown in Fig. 10.1.

Filter Type (RX only). Digital filters can be designed such that a signal within the pass band leaves the filter in a shape as similar as possible to what went into the filter. This requires that the phase difference between input

and output signal is a linear function of the frequency. Another desirable property of a linear filter is that the time delay between a signal going into a filter and what comes out is as small as possible. Unfortunately there some sort of uncertainty relation between these two properties, so you only can trade one for the other. The options for the filter type are thus `Linear Phase` or `Low Latency`. But note that there is a lot of latency in the HPSDR data processing which you cannot avoid, so ,,low" latency is not really low. Therefore, the default option is `Linear Phase`, and there should be little reason to change this. Note that piHPSDR does not allow low latency filters for TX, since in this case the TX compressor could not be used with the CESSB overshoot correction.

`Filter Size`. This is the number of ,,taps" of the digital filter. Increasing this size will inevitably increase the latency, but makes the filter edges steeper. The allowed values are powers of 2, and the minimum value equals the buffer size, which is hard-coded in piHPSDR to be 1024 (except for the transmitter in Protocol-2, where it is reduced to 512). The default value of 2048 should be fine in almost all cases, if you increase it, you can notice that the filter edges become little more ,,brick wall" like.

`Binaural`. The RX audio signal by default is a mono signal. Although you have the RX audio signal on both ears if using a headphone, both the left and right channel are the same. Checking `Binaural` for a receiver implies that its RX audio signal is stereo (left and right channel differ). This is accomplished by copying the primary I and Q signal of the RX output to the left and right channels instead of using the I signal for both ears (which is the default). Some users report that in modes such as CW and SSB, binaural audio is more pleasant than the default. It is up to the user to try and set this parameter according to personal preferences. This checkbox is available for all receivers but not for the transmitter.

10.2 The `Equaliser` Menu

In the `Equaliser` menu, you can modify the frequency response of the RX and TX audio. You can adjust the RX equaliser to your personal preferences for listening to the RX audio. The TX equaliser affects your transmitted signal. You can, for example, provide some extra amplification to the low-

frequency part of your voice. The menu is shown in Fig. 10.2.

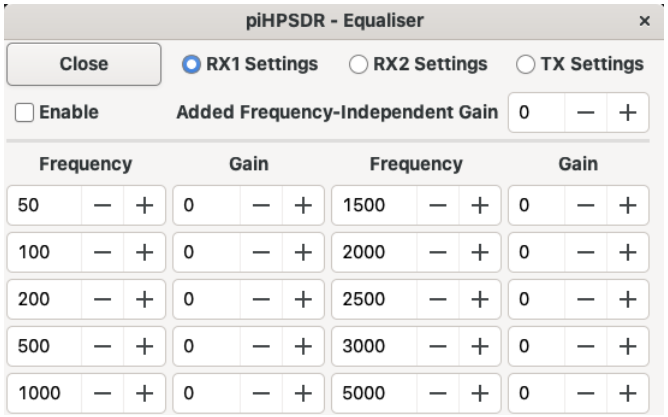

Fig. 10.2: The Equaliser menu

With the radio buttons in the top row you can select whether you want to control or modify the equaliser settings for the receivers RX1 or RX2, or for the transmitter. If only one receiver is running, the `RX2 Settings` button does not appear. Likewise, if the radio has no transmitter, the `TX Settings` button does not appear.

Below the top row, there are the controls that affect the equaliser that has been selected there. The `Enable` checkbox enables/disables the equaliser. If the equaliser of the active receiver is enabled, the EQ indicator in the VFO bar, upon receiving, turns yellow and reads `RxEQ`. If the TX equaliser is enabled, this indicator turns yellow while transmitting and reads `TxEQ`. So while receiving, you cannot tell, from the VFO bar, whether the TX equaliser is actually enabled or not!

The spin button `Added Frequency-Independent Gain` specifies an *additional* gain (or attenuation) applied to all frequency channels. Gains in this menu are in dB and can vary from -20 to +20, -20 is of course not a ,,gain" but an attenuation by 20 dB.

In the bottom part of the menu, there are ten pairs of spin buttons that specify 10 corner frequencies and the gains associated therewith. This defines the 10 channels of the equaliser. Note frequencies may appear in any order since they are sorted before the equaliser settings are updated. Note however, if you specify a frequency twice with two different gains, the outcome is undetermined.

The gain at one of the corner frequencies is exactly the gain chosen by the spin button (plus any frequency-independent gain selected by the spin button in the second row). Between two corner frequencies, the gain is frequency dependent and obtained by liner interpolation between the two corners. For a frequency between zero and the lowest corner frequency, the gain chosen for that corner frequency applies, and likewise the gain selected for the largest corner frequency applies to all frequencies above that frequency.

Equaliser settings are saved with the mode, so if you adjust the equalisers when doing USB or LSB, and then switch to DIGU or DIGL, the equaliser settings for DIGU/DIGL become effective (normally, equalisers are disabled for digital modes). They resume their USB/LSB settings upon switching back to USB or LSB. This also applies to other modes such as CWU/CWL, where the TX equaliser has no meaning anyway, and where the RX equaliser is normally not needed.

10.3 The `Ant` (Antenna) Menu

The `Ant` menu, as shown in Fig. 10.3, applies to HPSDR radios. For SoapySDR radios, the layout is much simpler because there are much fewer choices possible.

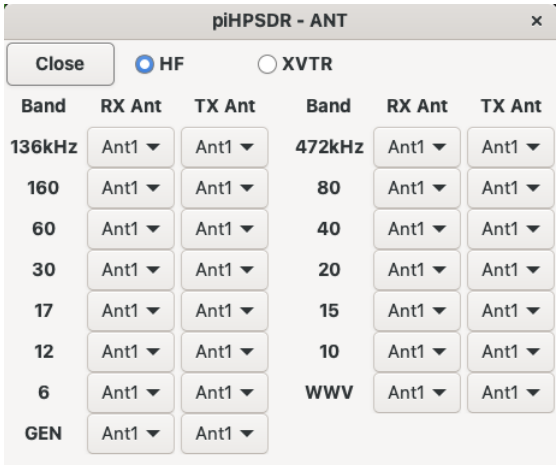

Fig. 10.3: The ANT (antenna) menu.

Standard HPSDR radios have, in most cases, three main antenna jacks de-

noted Ant1, Ant2, Ant3, which can be used both for receiving to the first ADC and transmitting. Then there are up to additional antenna jacks (Ext1, Ext2, and Xvtr) which can only be used for receiving and are also connected to the first ADC. If the radio has more than one ADC, the (RX only) antenna jack, usually denoted RX2, is hard-wired to the second ADC.

If the menu is opened, the HF button is checked, and the HF bands are displayed. The WWV and general ,,bands" are on display as well so you can e.g. choose an antenna for listening to signals outside of the amateur bands. If one checks the XVTR button, the transverter bands are shown (this leads to an empty window if no transverter bands have yet been defined), and one can go back to the HF bands by re-checking HF. For each band, one can now choose one (out the three) antennas for transmitting and one (out of six) antennas for receiving. The main purpose of this is the possibility to connect an additional receive-only antenna such as a beverage antenna which often has a better signal-to-noise ration than standard antennas used for transmitting.

Transverter operation. Newer radios (Anan-7000, 8000, and Saturn/G2) have a switchable low-power TX output, e.g. the `Xvtr Out` jack at the Anan-7000 back plane. This output is automatically enabled if Xvtr is selected as the RX antenna of RX1.

—— Attention, potential damage! ——

A problem that may potentially damage your external hardware occurs if you use one of the antennas Ant1/2/3 for receive and another for transmit. This is especially true if you have sensitive hardware (such as an active RX antenna) connected to the Ant jack used for RX and operate CW with the Key attached to the radio with the `CW handled in Radio` box checked in the `CW` menu.
In this case, starting CW transmission lets the FPGA (processing unit inside the radio) put the radio into TX mode, start forming the first RF pulse, and informs the host computer running piHPSDR that a RX/TX transition has been made.
Only then, piHPSDR can start telling the radio to switch the relays that connect the Ant jacks with the TX circuitry.
As as result, a small part (few ms) of the first RF pulse (dot or dash) may appear at the Ant jack used for RX only. If, say, an active antenna is connected there, this may well destroy the active antenna.
Even if not using CW this way, it cannot excluded that Ant relay switching is so slow that such ,,RF spikes" appear at an Ant jack intended for RX only.
A very recent (Jan 2024) update of the Protocol-2 definition offers the possibility to tell the radio which antenna settings is must use when it goes TX on its own behalf. piHPSDR fully supports this, and if you have an updated Protocol-2 firmware in your radio then the problem disappears. For Protocol-1, there is no such update so the problem remains.

If possible, use the Ext1/Ext2/Xvtr jacks for connecting sensitive equipment.

10.4 The OC (Open Collector) Menu

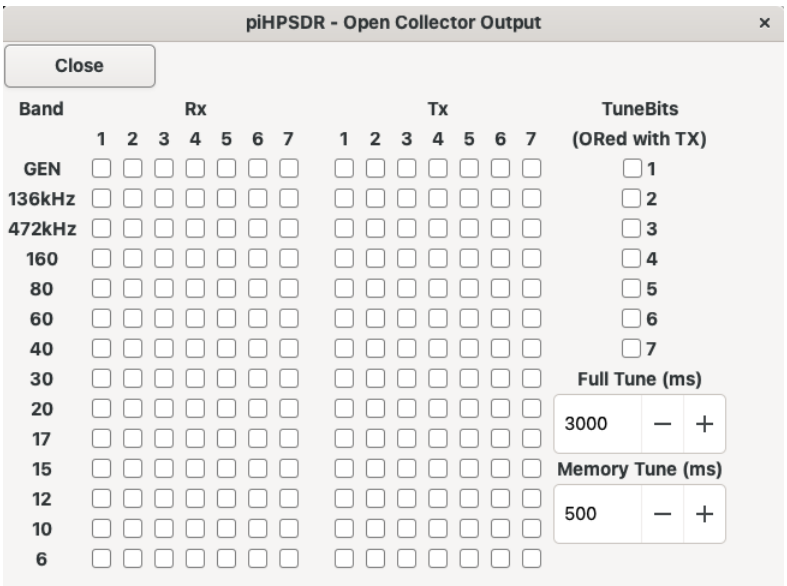

Fig. 10.4: The OC (open collector) menu.

Standard HPSDR radios have seven individually programmable outputs wired as open collector output. In the OC menu, you can specify, separately for each band, and separately for receive and transmit, which output should be ,,set". This can be used to switch the band filters of an external PA or of an external RX pre-selector, to control an automatic antenna tuner, and many more things, since it is your external hardware which in the end has to make sense of the output bit pattern.

For non-HPSDR radios, the OC menu does not appear in the main menu. Since transverter bands are also shown at the bottom of the list, the OC menu may become rather tall, so a scroll bar may appear on at the right margin for small screens.

To facilitate control of an automatic tuner, there are seven TuneBits which are OR-ed with the bit pattern chosen for TX on the actual band, as long as you are TUNE-ing with piHPSDR. Besides the Tune command, there are the Tune Full and Tune Mem commands which are functionally equivalent, except that the open collector tuning pattern is removed for Tune Full after the full tune delay, and for Tune Mem after the memory tune delay, which

can also be specified in this menu (spin-buttons below `Full Tune(ms)` and `Memory Tune(ms)`, the values are in milli-seconds). This can be used to send tuning pulsed of varying length to the external automatic tuner at the beginning of the tuning. If the open collector outputs are not used, or if the `TuneBits` are all unchecked, there is no functional difference between the `Tune`, `Tune Full` and `Tune Mem` commands.

The `OC` bit pattern you see in Fig. 10.4 is what you automatically get when selecting the `N2ADR` filter board in the `Radio` menu (this is usually the case if you are working with a HermesLite-II radio). This means that you *can* change the `OC` patterns when using the `N2ADR` filter board (e.g. for experimental purposes), but these changes are lost once you select another filter board and then select `N2ADR` again, and also upon the next program start. At the right margin of the menu, you see a scroll bar and you usually need to scroll down to see/modify the OC settings for the upper bands: at the bottom of Fig. 10.4 you see the settings for the 12m band but when you scroll down, those for the 10m and 6m band (and possibly transverter bands) also appear.

Chapter 11

The Main Menu: controlling piHPSDR

In this chapter, the customisation of the toolbar (at the bottom of the piHPSDR window), as well as how to configure GPIO and MIDI controllers, is described. Furthermore, in this chapter we discuss the `CAT` menu which allows controlling piHPSDR by some external program such as a logbook or contest program, via standardised CAT commands that can be sent to piHPSDR either over a serial line or via TCP. If TCI is enabled at compile time, this menu is called `CAT/TCI` and also allows to control the built-in TCI server.

Note for Controller1 owners: The eight switches (push-buttons) of the controller, that a positioned below the screen, are bound to the eight toolbar buttons on the screen. Therefore, there is no "Switches" menu for this controller, and the switches are implicitly configured via the Toolbar menu.

11.1 The `Toolbar` Menu

We start with the "Toolbar" menu, that can be found at the top of the rightmost column in the main menu. The toolbar consists of eight buttons that can be assigned to a set of eight functions. There are six such sets, and pressing the rightmost button of the toolbar cycles through these six sets. The text on the rightmost toolbar button, `FNC(x)`, indicates which layer (x

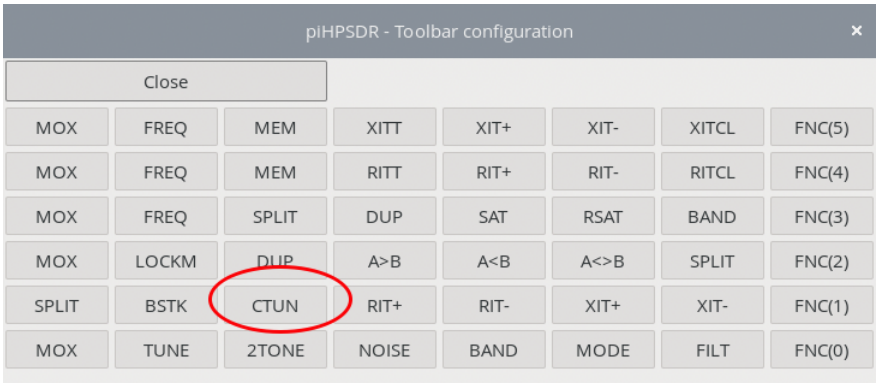

Fig. 11.1: The Toolbar menu, just opened.

runs from 0 to 5)is currently active.

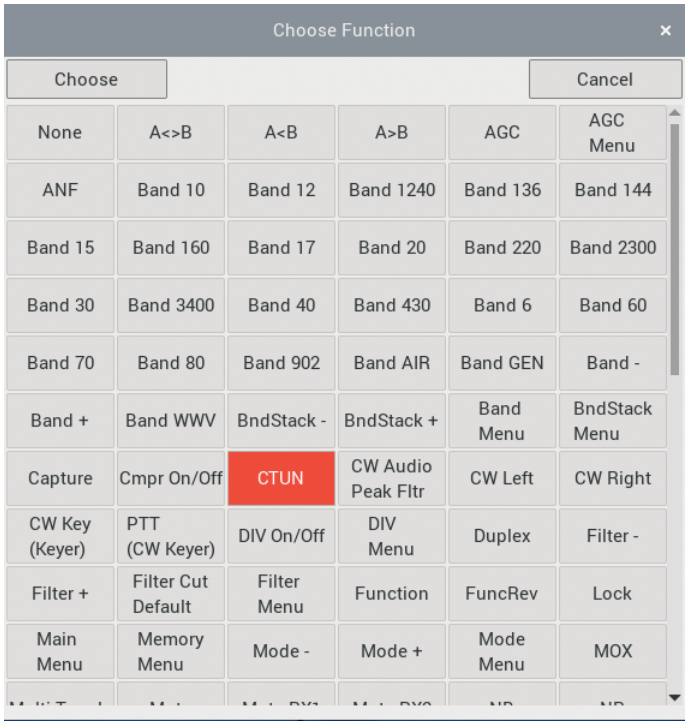

Fig. 11.2: Toolbar menu. Changing third button in `F1` layer.

If the `Toolbar` menu is opened, it looks like Fig. 11.1. The rows correspond to the six different layers, and the rightmost button in each row indicates

to which layer this row belongs. Now imagine we want to replace the CTUN command by Duplex in the row FNC(1). To this end we click the CTUN button (marked with a red circle) and a ,,chooser dialog" pops up that looks as in Fig. 11.2.

Because the list of possible functions to choose from is so long, the height of the menu is restricted even if screen space permits, so it always contains a scroll bar at the right margin that you may need to use to scroll through the list of functions. The current command selected (CTUN) is high-lighted (you may need to scroll down to see it). Then click the button labelled with Duplex (in this example you can already see it, but again it may be necessary to scroll down). After clicking the desired button, that one gets highlighted.

Fig. 11.3: Toolbar assignment accomplished.

In the top row of the chooser menu, you find two buttons. Pressing Cancel closes it without any effect (your choice is not reported back), while pressing Choose closes the menu and reports back your choice, that is, the function currently highlighted. So pressing Choose after having chosen the Duplex function, the chooser menu closes and one sees that in the toolbar menu now reappearing (Fig. 11.3), the third button in the line FNC(1) (marked by a red circle) has changed, it now gives the short text (DUP) of the Duplex command. You can also see that the FNC(1) toolbar at the bottom of the

screen has adopted the change (also marked with a red circle), so after closing the toolbar menu you can enable/disable `Duplex` with just one mouse click, without first opening the `Radio` menu.

11.2 The `Sliders` Menu

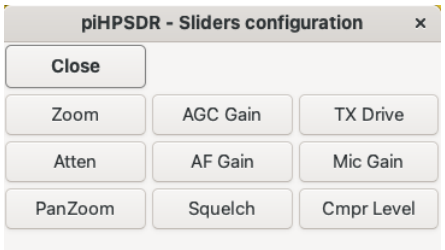

Fig. 11.4: The `Sliders` menu.

The sliders lets you assign function to the nine sliders in the 3x3 Sliders area. It does not matter how many slider rows (0 to 3) are actually shown, you can assign a function to all nine sliders using the same type of dialog used for assigning functions to toolbar buttons (see previous section).

For example, a CW operator usually does not need to adjust the Mic gain, the TX compressor, the Squelch level or the VOX threshold. But this operator may find it convenient to have a slider at hand that adjusts the CW speed. With an individually configured Slider area, it should be possible to have only two rows of sliders on display in most situations. It is possible to assign the same function to more than one slider, but a given function will only be assigned to one slider (the first one if going through the sliders row-wise). So in case of a duplicate assignment, the second slider position will just be empty. Note that RF gain and Attenuation are mutually exclusive, since a given radio either has a programmable RF gain (HermesLite-II, RadioBerry, SoapySDR radios) or a programmable attenuator (most HPSDR radios).

The functions, as all piHPSDR functions, are described in Appendix A but for the convenience of the reader, those functions which can be assigned to sliders in the Slider area are listed here. Since the space on screen is scarce for labelling the sliders, the acronym used there is also given in parentheses.

`None` No function is assigned to this slider (and it will thus not appear)

AF Gain (AF) AF volume of the active receiver

AGC Gain AGC gain of the active receiver

Atten (ATT) RF front-end attenuation for the ADC feeding the active receiver. Note if the ,,other" receiver gets the signal from the same ADC, it is also affected. If the radio does not have a step attenuator in the RF front end, the slider is not shown.

Cmpr Lvl (Cmpr) Compression level of the transmitter. Together with this slider, there appears a check-box that enables or disables the TX compressor.

CW Speed (WPM) CW speed (in wpm) of the internal or the radio keyer.

Linein Gain (Line) The gain of the Line-In jack of the radio (if available).

Mic Gain (Mic) The gain applied to the AF signal at the beginning of the TX chain.

PanZoom (Pan) The PAN value of the ZOOM function. This slider has no effect when ZOOM=1.

Panadapter Low (PLow) The lower-cut of the RX panadapter.

RF Gain (RF) RF front-end gain for the ADC feeding the active receiver. Note if the ,,other" receiver gets the signal from the same ADC, it is also affected. If the radio does not have a programmable gain in the RF front end, the slider is not shown.

Squelch (Sqlch) The squelch level applied to the audio from the active receiver. Together with this slider, there appears a check-box that enables or disables squelch.

TX Drive (TX Drv) The TX drive.

VOX Level (VOX) The VOX threshold. Together with this slider, there appears a check-box that enables or disables VOX.

Zoom (Zoom) The ZOOM value for the panadapter of the active receiver.

11.3 The CAT/TCI Menu

piHPSDR has a built-in rig control or CAT (computer aided transceiver) facility. This can be used to control piHPSDR from other programs or even other computers. You can have up to three simultaneous CAT connections via TCP, and three additional CAT connections via serial lines (provided the host computer running piHPSDR has those serial interfaces available). One further serial port (or USB-to-serial adapter) can be used for a PTT input and output line. Note that on the new Anan G2-Ultra radios, one of the serial lines is used for controlling the front panel. It is also possible to use FIFOs (also known as named pipes) instead of real serial devices, which offers a hardware-free connection of, say, a logbook program running on the same computer to piHPSDR, even if the logbook program cannot use TCP. On my Macintosh computer for example, using a named pipe and the Kenwood TS-2000 radio model, I can connect the MacLogger DX logbook program with piHPSDR. piHPSDR fully supports (thanks Rick!) the ANDROMEDA controller (see `github.com/laurencebarker/ Andromeda_front_panel`). This controller (or rather the Arduino inside) is connected to the host computer via USB and appears as a USB-to-serial device on the host computer. The CAT command set is explained in Appendix D. In most cases, using the Kenwood TS-2000 as the radio model would do it, if the digi mode or laptop program uses the `hamlib` library to interface with radios, either choose TS-2000 or (preferably) the ,,OpenHPSDR PiHPSDR" radio model because this uses time-out values adapted to piHPSDR.

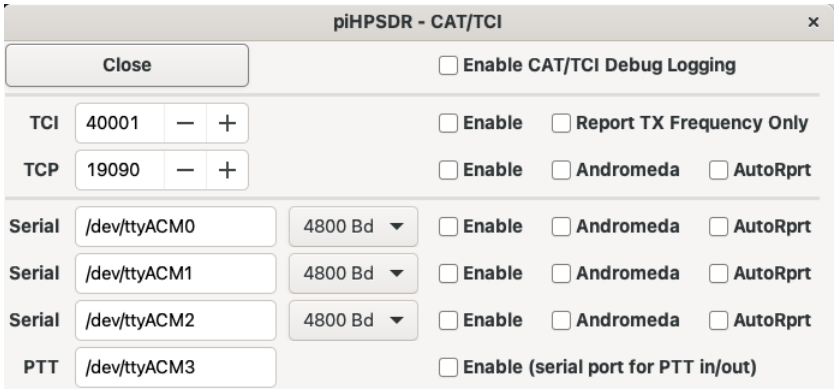

Fig. 11.5: The CAT/TCI menu.

piHPSDR also offers a (substantially stripped-down) TCI server piHPSDR cannot be *controlled* via TCI (all incoming TCI commands are ignored) but it periodically (every 500 msec) reports operating frequencies and modes, if they have changed. This is meant to support logbook programs and external PAs that use TCI to get frequency and/or mode information. The CAT/TCI menu is shown in Fig. 11.5.

Enable CAT/TCI Debug Logging. If enabled, the piHPSDR CAT subsystem sends lots of debug messages to the standard output. If piHPSDR is run from within a terminal window, these messages appear in the terminal window. If it is run from double- clicking a desktop icon, these messages can be found in a log file within the piHPSDR working directory (this is the directory where the preferences are stored. This checkbox is only of interest for software developers to analyse programming or connection errors, and should not be checked for normal use.

TCI. To the right of this string, there is a spin button which selects the TCP port used by the TCI server (the default value is 40001). The TCI server identifies itself as an ExpertsSDR3 program running TCI version 1.8 with a SunSDR2PRO radio, but in fact it ignores all incoming commands and only reports VFO-A/B frequencies and modes, if they have changed. It also reports the TX frequency if it has changed. This is necessary since the TX frequency can jump from the VFO-A to the VFO-B frequency and back e.g. when changing the split status or the active receiver. It is thus clear that TCI can be used to inform logbook programs and/or PAs about frequencies and modes, but not much more.

Report TX Frequency Only. If power amplifiers connect to piHPSDR, they usually only want to know the TX frequency to adjust the output low-pass filters. Checking this box restricts the TCI outgoing commands to the TX frequency messages. It is important to note that the TX frequency is only checked every 500 msec, so after a band change, the PA may not be aware of the changed TX frequency for half a second.

TCP. This sets the TCP port number for CAT connection to TCP. The default value (19090) is rather standard, using another one is only necessary if you are running more than one SDR program on the host computer at the same time. This port number must match the port number used in the (digi-mode or logbook) program that wants to connect.

Serial. piHPSDR supports up two three CAT connections over serial lines or named pipes. In the text field, enter the device name of the serial port or the named pipe to use. Which names to use is highly operating system dependent. On a RaspPi, USB-to-serial adapters (which are nowadays the standard way to add serial ports to a computer) have names such as `/dev/ttyACM0` or `/dev/USB0` (on RaspPi) or `/dev/tty.usbserial-....` (on MacOS). On Anan G2-Ultra radios, a serial line is used for the communication with the front panel. This port is detected automatically and the third "Serial" line done not contain any controls but simply states the auto-detected port name and whether the connection to the panel has been successfully established.

To the right of the serial port text field, there is a pop-down menu for choosing the baud rate. Only 4800, 9600, 19200, and 38400 baud are offered, but this should cover most cases.

PTT. One serial port can be used for PTT-in and PTT-out. Only the DTR, RTS, and CTS control lines and GND of that port will be used. According to the RS232C standard, these control lines have two states (OFF and ON) defined through voltage levels (w.r.t. GND) that are below -3 V for OFF and above +3 V for ON. For a serial port used for PTT, piHPSDR will set the DTR output line to ON constantly and sense the CTS input line periodically (every 100 msec). If its level changes, ON is interpreted as PTT pressed, and OFF as PTT released. Since a serial port will usually read the OFF state for an open (unconnected) line, a foot-switch used for PTT then must simply connect the DTR and CTS lines of the serial port. The RTS line follows whether piHPSDR is transmitting (RTS=ON) or not (RTS=OFF). For most operating systems, RTS is OFF when the serial port is not in use and set to ON when it is opened. piHPSDR will then set it OFF as soon as possible, but most likely there will be a short ON pulse on the RTS line when piHPSDR starts or when the PTT port is enabled in this menu, and this might lead to a short click-clack of the T/R relay in the external gear if this is controlled by the RTS line.

Using serial control lines as general purpose input/output lines (which is outside of the RS232C standard anyway) is different if a USB-serial interface with TTL-level lines is used. In this case, in addition to the RTS, CTS, and GND lines, also a +5V line is provided. There is no official standard for serial TTL levels, but according to my experiments negative TTL logic is used. This means that the RTS line has HIGH level (> 2.5 V) during RX and

LOW level (< 0.4 V) during TX, and the CTS input line must be connected to GND with a PTT foot switch, and to +5V via a pull-up resistor.

Enable. This checkbox enables or disables the function in that particular line. Enabling and disabling can be done separately for the TCI server, the TCP CAT subsystem, for each of the serial CAT connections, and for the PTT in/out serial port.

Andromeda. (For TCP and serial CAT only) This checkbox enables the TCP or serial CAT connection for use with ANDROMEDA consoles or the G2-Ultra front panel. Serial lines are forced to 9600 baud if enabled for ANDROMEDA. Furthermore, status changes of piHPSDR (e.g. RIT on/off) are automatically reported via ZZZI CAT commands (see Appendix D). These messages are used by the ANDROMEDA console or the G2-MkII front panel to switch LED indicators such that they reflect the current state of piHPSDR. Note that the Andromeda flag is only inspected when the connection is opened. Serial CAT connections are established when successfully opened upon program start or via this menu. TCP CAT connections are established when the *client* successfully connects.

AutoRprt. (For TCP and serial CAT only) If a CAT connection is opened and this box is checked, the connection is put in ,,auto reporting" mode. This mode can generally be enabled and disabled with the ZZAI or AI CAT commands (see Appendix D), but with this checkbox, the connection can be put into "level 1" auto reporting mode without the CAT client sending AI commands. "Level 1" means that piHPSDR behaves as if an `AI1;` or `ZZAI1;` command had been received. In this level, frequency changes are transmitted via FA and FB CAT messages, but mode changes are not transmitted. The main use for auto reporting mode is, if a buggy power amplifier or a buggy logbook program is connected via a serial line that needs frequency or mode info via CAT but fails to send a proper `AI` command. If you do not have hardware that depends on such unsolicited (without requesting them via ZZAI or AI commands) messages, this box should never be checked.

11.4 The MIDI Menu

MIDI (musical instrument digital interface) is a protocol designed for the communication of musical instruments, such as keyboards and tone genera-

tors. Because of its widespread use, support in all major operating systems, and it inherent ability to deliver real-time ,,events", it is also an ideal protocol to control an SDR program. The only MIDI messages piHPSDR processes are NoteOn, NoteOff, and ControllerChange messages. Typically, a NoteOn message is sent if a key on a keyboard is hit. The first parameter of a NoteOn/Off message is the key it refers to. Although keyboards rarely have more than 88 keys, the allowed range for the key is 0-127. There is an additional parameter (,,velocity", range 0-127) that tells how fast the key has been hit (this makes the difference between a soft and loud tone on the piano). NoteOn/Off messages are ideally suited for indicating button press and release events. In principle, piHPSDR does not need the velocity. However, several MIDI consoles, in a sloppy interpretation of the MIDI standard, send a NoteOn value with zero velocity if a button is released. Therefore, piHPSDR interprets a NoteOn message with a velocity different from zero as a ,,button press", and interprets both a NoteOn messages with zero velocity and a NoteOff message as a ,,button release".

The original MIDI standard was built upon a daisy-chained serial connection. Each device echos all messages it receives at its input side to the output. Therefore, a key-down message that originated on one keyboard is sent to all tone generators. Likewise, a tone generator receiving a key-down message cannot tell from which device the message was originally sent. To resolve possible conflicts, each MIDI message contains a channel number. There is some confusion about channel numbers: the MIDI says channel numbers go from 1 to 16. Because this is encoded in a 4-bit data field whose numerical value goes from 0 to 15, computer users normally refer to channel numbers from 0 to 15, and this convention is also followed by piHPSDR. Different channel numbers can be used to discriminate MIDI events from different sources (devices). An example of such a setup is if you connect a DJ console as well as a micro controller to which a CW key is attached.

The second type of messages are ControllerChange messages. Typically, they report the value of an expression pedal, if it has changed. A ControllerChange message also has two parameters, namely the number of the controller (0-127) and the value (0-127). A ControllerChange message could be sent if the MIDI controller has a potentiometer, to report its position, encoded from 0 (full counter clock wise) to 127 (full clock wise). Such a message could then be used to control in piHPSDR, say, the AF volume or the TX drive. Such a potentiometer is not suited to become a ,,VFO knob". Here one

uses rotary encoders, a piece of hardware which you can turn (as long as you like) in either direction, and which reports (by hardware pulses) how fast and it which direction it is turned. Unfortunately, there is no standard how to encode these increments into MIDI ControllerChange messages. My Behringer CMD-PL1 console, for examples, uses ControllerChange numbers 65, 66, 67, ... for clockwise rotations and 63, 62, 61, ... for counter clock wise rotations, and further encodes the speed of rotation in how far the value differs from 64. Other brands interpret the 7-bit number as a signed quantity, such that values 0, 1, 2, ... correspond to clockwise, and numbers 127, 126, 125, ... to counter-clockwise rotations. In order to support a variety of low-cost ,,DJ consoles" from music stores, the piHPSDR MIDI configuration menu must be flexible enough to handle all these situations.

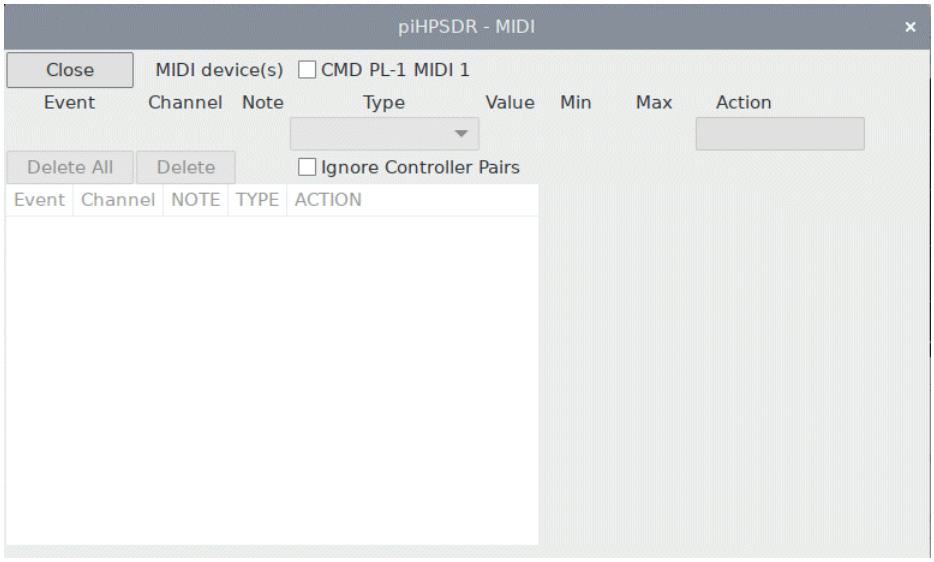

Fig. 11.6: The (virgin) MIDI menu.

From this it is clear that within piHPSDR, we have to distinguish three types of MIDI commands:

`Button`. This type hard-wired to NoteOn/Off MIDI events. piHPSDR commands (,,Actions") that can be assigned to this type are typically those which can also be assigned to toolbar buttons.

`Slider`. This type is hard-wires to MIDI ,,pitchbend" events, but can also be assigned to MIDI ControllerChange events. This make only sense if these

events are generated by a slider or a potentiometer, that is, the MIDI ControllerChange messages report values between 0–127 that depend on the position of the slider or knob. This can be used for piHPSDR functions that are usually controlled by a slider, such as adjusting the AF volume, setting the TX drive, setting the AGC gain, etc. The characteristic of such a control element that it reports values in a fixed range.

`Encoder`. This type can only be assigned to ControllerChange MIDI events, which makes sense if the MIDI messages are generated from a rotary encoder and the ControllerChange value reports the number of increments and the direction when the encoder is turned. The prototypical piHPSDR function controlled by a WHEEL is a VFO knob, which you can spin forever. However, you can also assign to to the AF volume control. piHPSDR takes care that the AF volume stops at the extreme cases (-40 and 0 dB for AF volume) even if you continue spinning.

The first kind of MIDI device which is often used for SDRs are the so-called MIDI DJ consoles. If you search the internet for ,,Hercules DJ controller" or ,,Behringer DJ controller" you will find lots of examples. For a very decent price, you can obtain a device which features lots of controls which you can conveniently use as VFO knobs, smaller knobs for controlling the AF volume etc., and push buttons to be used, for example, instead of toolbar buttons. The second kind of MIDI devices are small MIDI-capable micro controllers, starting with Teensy and Arduino devices which have a 32U4 micro controller which has built-in MIDI capability. With such a micro controller, you can build your own "DJ controller". Let the 32U4 control lots of push buttons and rotary encoders, and send the MIDI messages via USB to the computer. Using such a micro controller is also the most convenient and general way to connect a Morse key or paddle to the host computer running piHPSDR (see Appendix E, you can but need not use the same micro controller for taking care of the buttons/encoders and the CW key).

If you open the `MIDI` menu for the first time, it presents itself as shown in Fig. 11.6.

At the top of the menu, besides the close button, you find a list of MIDI devices in the system, each of which with a check box. In Fig. 11.6, there is only one such device with name ,,CMD PL-1 MIDI 1". You will find all MIDI devices attached to the host computer here. With the check box(es), enable those you want to use. This way it is possible to run two instances

of piHPSDR on the same computer, both connected to different radios, and control them independently with two different MIDI consoles. The first thing you have to do is to check all MIDI devices you want to use.

`Ignore Controller Pairs`. This check box which is unchecked by default The MIDI standard offers the possibility to combine two controllers, one (primary one) in the range 0–31 and the other (auxiliary one) with a a controller number larger by 32, thus in the range 32–63. The value of the auxiliary controller is then interpreted as a fine resolution correction of the value of the primary controller. Technically speaking, a pair of ControllerChange events sets a 14-bit value for the lower controller. Some MIDI consoles from the music market (for example, the Hercules DJ200 controller) use this mechanism. piHPSDR, especially the MIDI menu, gets confused by these "controller pairs". Checking the `Ignore Controller Pairs` box just tells piHPSDR to ignore MIDI ControllerChange event if the controller number is between 32 and 63. As a consequence, only the most significant 7 bits of the 14-bit value are used, which is fine for most applications.

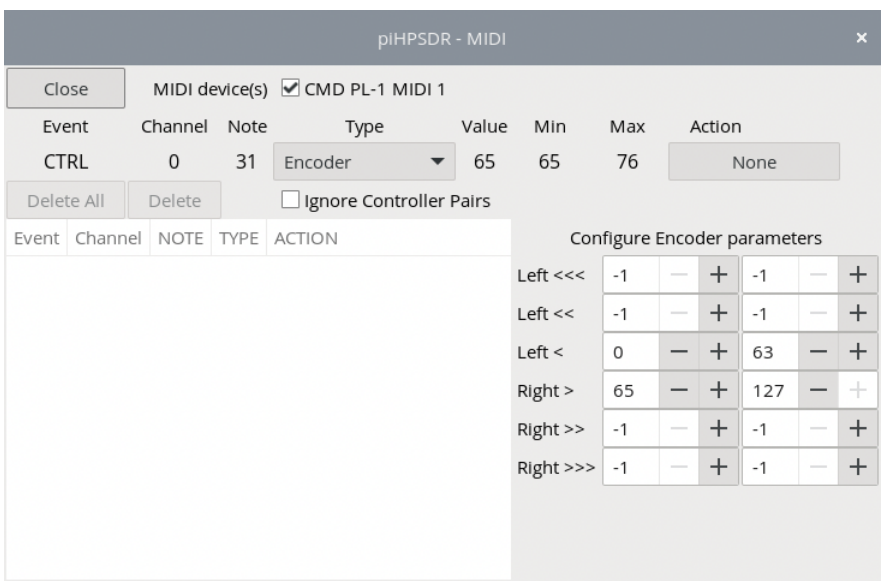

Fig. 11.7: The MIDI menu, VFO dial turned.

It is important to note that as long as the MIDI menu is open, the radio cannot be operated through MIDI since all MIDI messages are captured by the menu and displayed. This is used to implement a „self-learning"

configuration. To explain this in detail, we demonstrate how to configure MIDI such that the big wheel on the controller is used as a VFO dial. After I have activated the checkbox of our MIDI device, I just turned the big wheel of the MIDI console a bit. This resulted in a menu window that is shown in Fig. 11.7. In the third line, below `Event`, you see `CTRL` which indicates that the last MIDI messages received was a ControllerChange message. In the case of a NoteOn/Off messages, this field would read `NOTE`. You should rotate the knob in both directions to see what happens: below `Value` the ControllerChange value of the latest message is recorded, while the `Min` and `Max` fields report the smallest and largest value seen (all in the range 0-127). By playing around, it became quickly clear that this is a rotary encoder, sending messages in the range 65, 66, 67, ... for clockwise rotation and values 63, 62, 61, ... for counter clockwise rotation. If it were a potentiometer, you would see values between 0 and 127 depending on the position of the potentiometer.

Below `Channel`, you see the value zero which indicates that the channel number of that MIDI message was 1 (see above on the different channel numbering). Below `Type`, you see a pop-down menu, here you can choose between `Encoder` and `Slider`. In the example shown, it must be an `Encoder` since this is a rotary encoder. Because there is no standard how the values map to increments, a separate panel `Configure Encoder parameters` pops up when an encoder is to be configured. Here one has to define ranges of values that apply for very fast left turns, fast left turns, normal left turns, normal right turns, fast right turns, and very fast right turns. Specifying an interval from -1 to -1 means that this case will never be realised. In the example shown (Fig. 11.7), we have chosen to map all values from 0-63 to a left turn, and all values from 65-127 to a right turn.

Now we have to specify which piHPSDR function should be triggered when moving the wheel. The current command is shown in a button below the string `Action` and defaults to `None`. By clicking this button, a dialog to choose the function opens as described for the `Toolbar` menu (chapter 11.1), with the current choice (`None`) highlighted. The only difference is that now only functions are listed that can be assigned to encoders. Because we want to assign the wheel to the `VFO` function, we click the `VFO` button, which then becomes highlighted (Fig. 11.8).

Then one has to click the OK button to make the choice, and one returns to

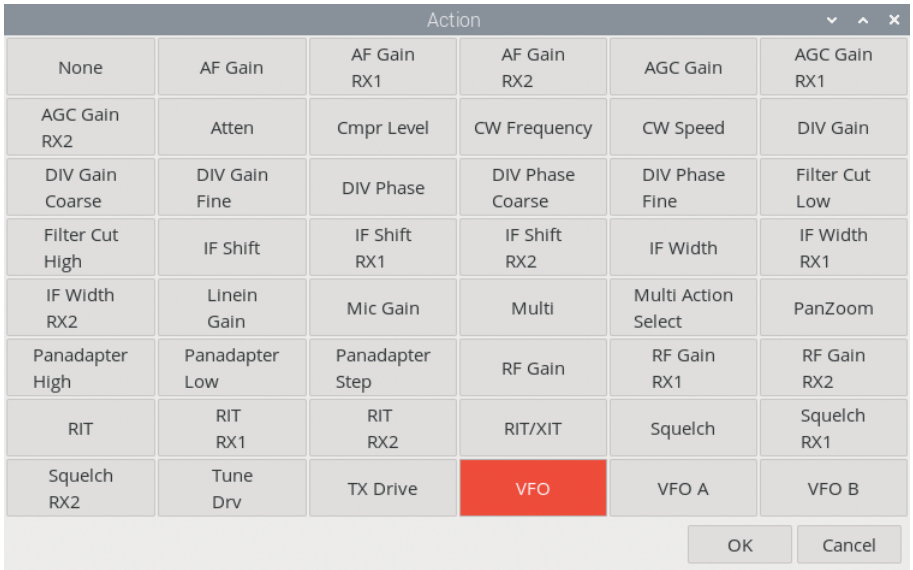

Fig. 11.8: The MIDI menu, selecting VFO command

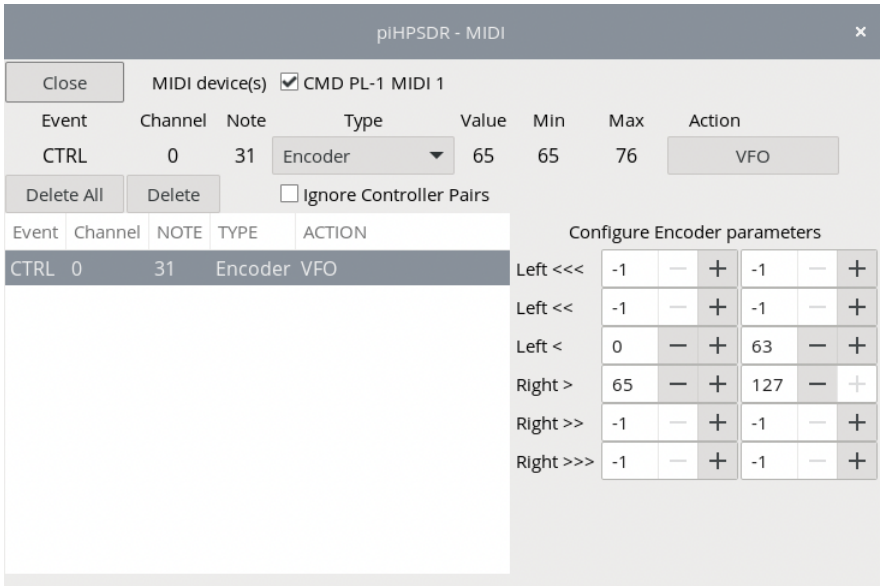

Fig. 11.9: The MIDI menu, VFO command selected

the MIDI menu (see Fig. 11.9).

One sees that the choice just made has entered the MIDI configuration, as

documented by the list in the bottom right part of the menu. At this stage, we can continue assigning more encoders, potentiometers, or buttons. If we close the menu at this point, then the big wheel on the MIDI console can immediately be used to change the VFO frequency.

11.5 The Encoders Menu

Fig. 11.10: A picture of the (first generation) G2 front panel (image courtesy of Apache Labs).

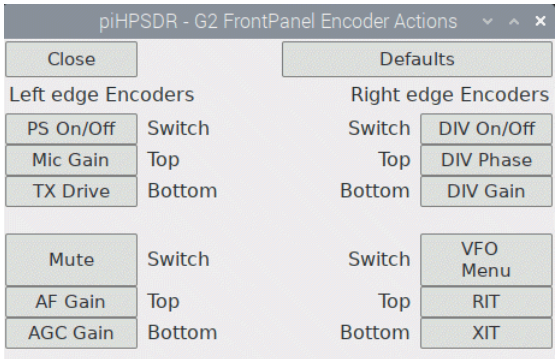

Fig. 11.11: The Encoder menu for the G2 front panel controller

The `Encoders` menu can be used to assign functions to the encoders of a GPIO-based controller, that is, either a Controller1, Controller2 or the G2 front panel of the first-generation Anan-G2 radios (those without LEDs and a

7-inch screen). For the second-generation G2 radios ("G2 Ultra") with LEDs and an 8-inch screen in the panel, see the G2panel menu, chapter 11.7.

This implies that this menu is only available if piHPSDR has been compiled with GPIO support, and if furthermore on the initial (discovery) screen one of the controllers mentioned above have been selected. Note further that the ,,large knob" of these controllers cannot be assigned a non-default function, this knob is hard-wired to the VFO function.

While the function of this menu is the same in all three cases (Controller1, Controller2, G2 front panel), the layout is different, because the position of the menu buttons are meant to indicate which encoder is referred to.

The G2 front panel (Fig. 11.10) has, in addition to the large VFO knob at the bottom right, four small knobs, two (one above the other) at the left edge and two at the right edge. All four knobs are double encoders with a switch. This means that there is an inner/upper knob (,,top encoder") and an outer/lower knob (,,bottom encoder"), which are two separate encoders. Furthermore, you can push the knob and have and additional push button function (,,Switch"). If you open the Encoders menu for a G2 front panel controller, the menu opens as shown in Fig. 11.11. You see four groups with three buttons (Switch, Top, Bottom) each, and it should be clear which group belongs to which encoder. With the buttons, you can choose which function to assign, in the same way as described for the Toolbar (chapter 11.1) and MIDI (chapter 11.4) menus. With the Default button, you can re-assign the default values (those shown in Fig. 11.11) to the encoder functions, which match the silk printing on the enclosure (see Fig. 11.10).

The Controller2 (see Fig. 11.12 has (besides the VFO knob at the bottom right) three knobs (arranged horizontally) at the bottom left, and a fourth knob at the top right, all of which are double encoders with a switch. So if you run piHPSDR with a Controller2, then the menu looks different (Fig. 11.13). The menu shows for groups with three buttons each, and it should be clear which group belongs to which button. The Default button again re-installs the default functions (those shown in Fig. 11.13).

Finally, the Controller1 (see Fig. 11.14) has (besides the big VFO knob at the bottom right) three knobs (denoted E1, E2, E3), arranged vertically at the right edge. These knobs are single encoders with a switch (you can turn the knob, but you can also push it). Therefore, the Encoders menu in this

Fig. 11.12: A picture of the Controller2 (image by courtesy of Apache Labs).

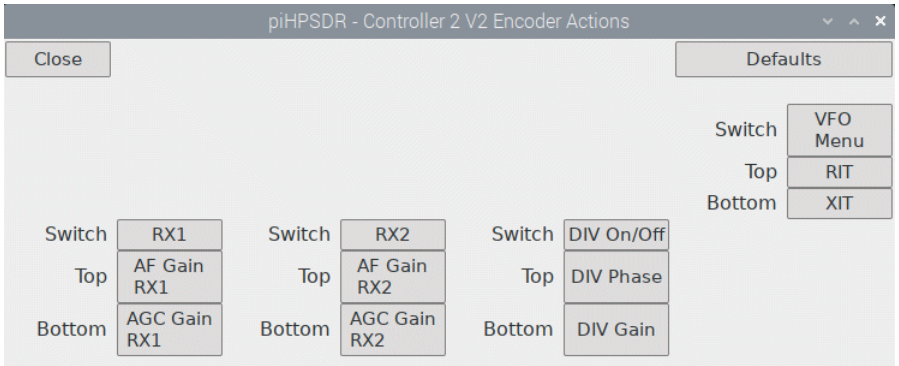

Fig. 11.13: The Encoder menu for Controller2

case (Fig. 11.15) shows three groups with two buttons each. The `Default` button again re-installs the default values shown in Fig. 11.15, these are chosen just for convenience since there are no default function printed on the enclosure.

Fig. 11.14: A picture of the Controller1 (image by courtesy of Apache Labs).

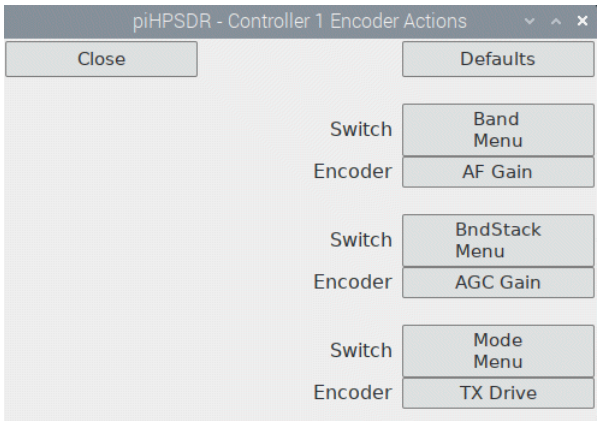

Fig. 11.15: The Encoder menu for Controller1

11.6 The Switches Menu

he Switches menu can be used to assign functions to the encoders of a GPIO-based controllers Controller2 or the G2 front panel of the first-generation Anan-G2 radios (those without LEDs and a 7-inch screen). For the second-generation G2 radios ("G2 Ultra") with LEDs and an 8-inch screen in the panel, see the G2panel menu, chapter 11.7.

This implies that this menu is only available if piHPSDR has been compiled with GPIO support, and if furthermore on the initial (discovery) screen one of the controllers mentioned above have been selected. Note that this menu is also not available for the Controller1 because the eight push buttons of this controller are hard-wired to the toolbar buttons and their functions as thus assigned via the Toolbar menu (see Chapter 11.1).

On the G2 front panel (see Fig. 11.10), there are a lot of push buttons: at the left edge, to the right of the left edge encoders, are two buttons, at the bottom right, below the VFO knob, there are two more buttons, and at the top right, to the right of the left edge encoders, is an array of 12 (4x3) buttons. The layout of the Switches menu for the G2 front panel (Fig. 11.16) features (besides the Close button) sixteen buttons, and their arrangement is such that you can easily guess which menu button refers to which button on your G2 front panel. Assigning functions to these buttons is done exactly as described for the Toolbar menu (chapter 11.1). With Default one re-installs the default values shown in Fig. 11.16 which match the functions printed on the enclosure.

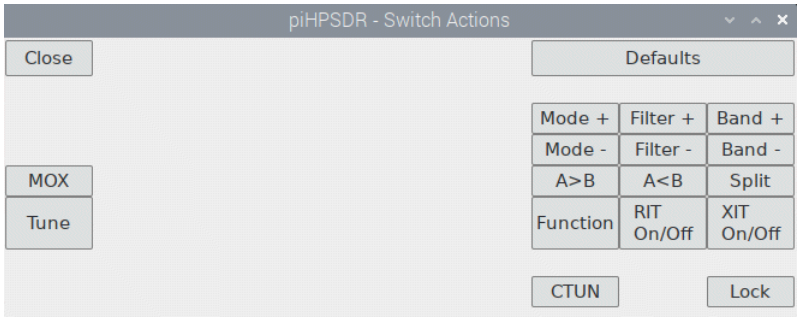

Fig. 11.16: The Switches menu for the G2 front panel controller.

The Controller2 (see Fig. 11.12 in the last section) also has 16 push buttons,

but they are arranged differently: at the bottom edge there are 7 buttons arranged horizontally. At the right edge, there is an array of 8 buttons (4x2) with one additional button above the right column that is to the right of the fourth encoder, just below the power button. Looking at the Switches menu for the Controller2 (Fig. 11.17), you see representations of these 16 buttons in an arrangement for which it is self evident which menu button refers to which Controller2 push button. Assigning functions to these buttons is done exactly as described for the Toolbar menu (chapter 11.1). With Default one re-installs the default values shown in Fig. 11.17 which match the functions printed on the enclosure.

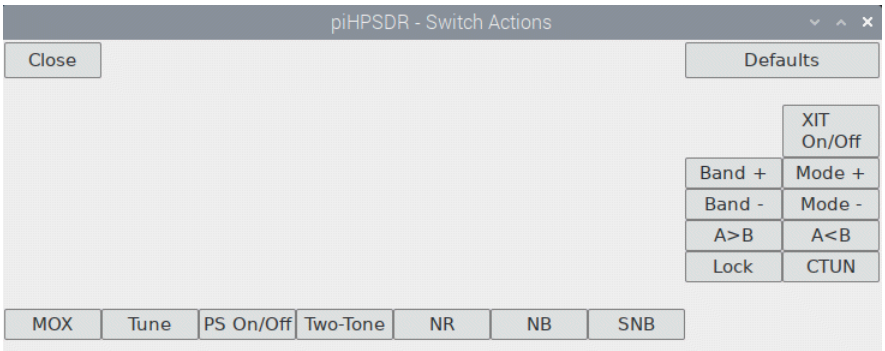

Fig. 11.17: The Switches menu for Controller2.

11.7 The G2panel Menu

The G2panel menu becomes available as soon as the presence of a second-generation G2 front panel (G2 Ultra or upgraded first-generation G2s) has been detected. These panels communicate with piHPSDR through a serial line connection using an ANDROMEDA-like protocol. Fig. 11.18 shows the front panel of a G2-Ultra. In contrast to a first-generation G2, it has an 8-inch screen and panel contains several LEDs. The G2panel menu also applies to first-generation G2 radios with an upgraded compute module running piHPSDR. Part of such an upgrade is an additional micro controller that handles the panel and communicates with piHPSDR via a serial line connection.

Note that this menu cannot be used to assign a non-default command to the

Fig. 11.18: The front panel of a second-generation G2 radio (image courtesy of Apache Labs).

VFO knob. After first opening the `G2panel` menu, it looks quite empty (Fig. 11.19):

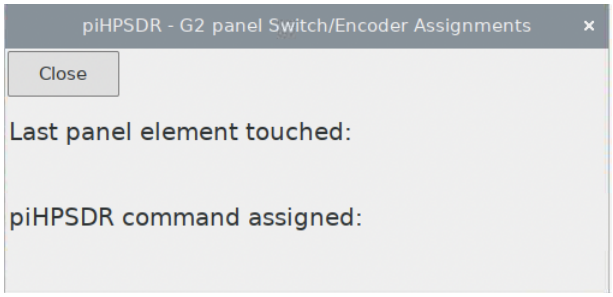

Fig. 11.19: The `G2panel` menu, just opened.

As long as this menu is open, the front panel seems inoperative. The reason is, that all button and encoder event are "digested" by the menu itself. As soon as a button is pressed or an encoder is turned in either direction, the menu records which controller was last used, and reports the last recorded action (either a button press or turning an encoder knob). For example, after pressing the TUNE button, the menu changes to the following (Fig. 11.20):

So the menu states that the last "panel action" recorded was a button press (the internal code number of the button is of little importance here), and that the piHPSDR command assigned to this button was `Tune`. Furthermore, once there has been communication between the panel and the `G2panel` menu, a large button appears at the bottom of the menu which can be used to

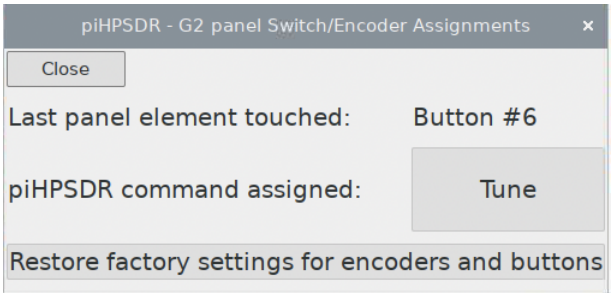

Fig. 11.20: The `G2panel` menu, reporting the last "panel action".

overwrite all button and encoder assignments with the factory default.

I you wish to assign a new command to this button (it is the last button you have touched), simply click into the button in the second line which states which command is currently assigned. In our example, it is the button with the label `Tune`. Clicking this button opens the same "action dialog" already discussed in the previous sections. In this dialog, you can choose a new command to be assigned with that button. In our example, I have chosen the `Cmpr On/Off` command which switches TX compression on/off (Fig. 11.21).

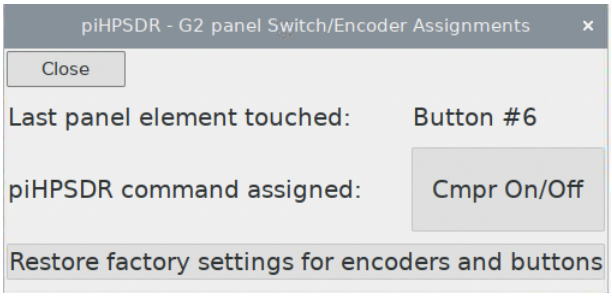

Fig. 11.21: The `G2panel` menu, reporting the newly chosen command.

If one wants to assign new commands to other buttons or encoder, just press the button or turn the encoder, and the menu will report a new "last panel action" and lets you assign a new command to that button or encoder. If all assignments have been made, close the menu and the new panel functions become operative.

Note that your assignments are permanent. They will be stored in the props file and be restored the next time you start piHPSDR. If you think your

choices were not that clever, you can always go back to the factory settings, which will then overwrite your personal assignments in the props file as well.

Chapter 12

Client/Server remote operation

In client/server operation, two instances of piHPSDR are running closely coupled. The ,,local" piHPSDR program (called the server) runs as usual on a computer with direct network connection to the radio. This computer is usually located in the vicinity of the radio. On the other hand, the ,,remote" piHPSDR program (called the client) may run on a computer quite far away from the radio. Both instances of piHPSDR are connected via a network connection that may bridge a long distance. Wireless connection is perfectly possible between the client and the server. The connection of the two piHPSDR instances need much lower bandwidth as e.g. the server/radio connection. To keep bandwidth low, the RX audio samples are transferred in mono, since the data stream from the server to the client is dominated by the audio samples (48000 16-bit mono samples per second, or 768 kBit/sec). The same bandwidth is used from the client to the server while TXing in modes other than CW (,,microphone" samples). For those, no heavy protection/encryption is necessary: the data flowing is essentially that what can be heard on the air anyway. The method of generating CW inside piHPSDR has been converted to a ,,time-stamped event model", so sending CW is perfectly works smoothly with remote operation. Of course, server to client bandwidth is twice as large (1.5 MBit/sec) if two receivers are running, but still, this is perfectly possible even using WiFi.

Client and server communicate using both the TCP and UDP protocols. UDP is used for data periodically sent, such as audio data (in both directions) and panadapter data (from the server to the client). Thus sending this data

cannot stall the sender if the connection ,,hangs" for a short time. TCP is used for all types of commands, user interactions, or CW events since here is should be secured that nothing is lost. The port number specified in the `Server` menu and the discovery screen (see below) is used both for TCP and UDP.

Since this is a legal requirement in many countries, access to the server is password protected to prevent illegal use of the transmitter. While the password is stored unencrypted on the server side, a challenge/response model ensures that a non-encrypted password is never contained in the data stream between the client and the server.

12.1 Server side: the Server menu

In the piHPSDR "server" instance (connected directly to the radio), client/server operation must be enabled via the `Server` menu, to be opened by a button in the bottom part of the leftmost column of the main menu. The server menu is shown in Fig. 12.1 and only has a few controls.

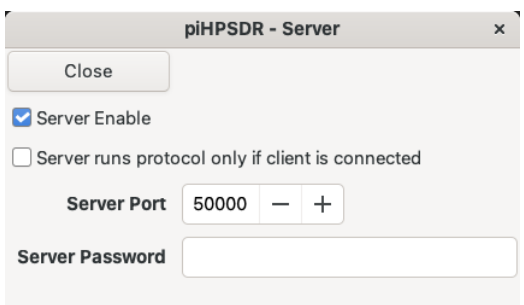

Fig. 12.1: The `Server` menu

The `Server Enable` check-box can be used to enable or disable remote operation. If the box is not checked, a client cannot connect. The spin-box to the right of the `Server Port` label lets the user choose the TCP port to be used for remote operation. This must be a port that is not yet used on the computer running the piHPSDR server instance. The default value 50000 works in most cases, but you can choose another value should the port 50000 be in use on your computer. Other ports used by piHPSDR are 19090 (the

default port for CAT-over-TCP) and 40000 (the default port for the TCI server).

In the text entry field to the left of the `Server Password` label, you must enter a password for client-server operation. If it is a really long password, only the first 64 characters are used, and if it is shorter than 5 characters, no client is allowed to connect. No particular security measure is taken to protect the password on the server side: it is shown while you type it into the text field, and it is stored in the local preferences file without encryption. The idea here is that you set up the server in a private environment, and against those who manage to hack your computer running the piHPSDR server instance there is nothing one can do from within piHPSDR.

`Server runs protocol only if client is connected`. If this option is checked, the server halts the communication with the radio when it starts. It restarts the communication once a client is connected, and stops it when the client disconnects. This option has been implemented for a group of amateurs which run a remote radio hooked up with several computers running several SDR programs offering remote operation: Without this option, piHPSDR would connect to and take over the radio when it starts, and then ,,occupies" the radio even if no client is connected. With this option checked, the radio can be operated by several SDR programs (not at the same time, of course). If the radio is available, piHPSDR resumes data exchange with the radio once a client connects, and stops the protocol once the client disconnects, and then other amateurs using other SDR programs can use the radio.

If the server is already running, and you want this option to become effective immediately, un-check and re-check the `Server Enable` box. Choosing `Restart` in the main menu will also resume radio operation.

12.2 Client side: connecting the server

Having explained the `Server` menu, we now discuss how operate remote. Let us assume you have a server instance of piHPSDR running, so how to connect to it from a remote location? Here it is useful to show again the discovery menu on the piHPSDR start screen (Fig. 12.2):

Here, no local radios have been found as indicated in the first line. The second line contains the controls relevant for client/server operation. There

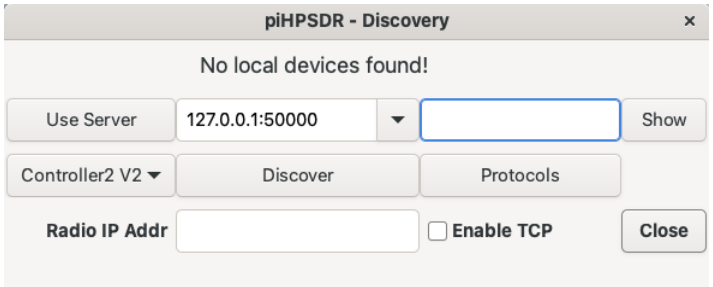

Fig. 12.2: The discovery menu on the client side

is a pop-down menu to the right of the `Use Server` button. Here one has to specify the host address and port number of the server, address and port separated by a colon. For the host address, on can use numerical IP addresses of the form xxx.xxx.xxx.xxx but equally well a host name. The port number must, of course, match the port number specified in the `Server` menu in the piHPSDR server instance (see above).

In the `host:addr` pop-down menu, you can choose the entry matching your server. On a virgin system, this menu will only show a default entry which is 127.0.0.1:50000, as well as an additional empty entry. By clicking into the text field, you can change an entry such that it matches for piHPSDR server instance. You can also select the empty entry from the pop-down menu and enter a new `host:addr` pair, which will then be added to the list. The last element of the pop-down menu is always an empty one, so you can add as many different entries as you wish. If you select an entry, click into the text field and clear it, then this entry is removed from the pop-down menu. In this way, you can configure your client computer to use several different piHPSDR server instances, but also to use a piHPSDR server instance that has different IP addresses or host names depending from where you connect.

The second text field in the server line must be used to enter a password. Hitting the `Return` key while typing in a password here is equivalent to pushing the `Use Server` button (this saves you one mouse click after having entered the password). Normally, this password will not be stored in the preferences file so you have to enter it each time you connect. This seems to be the most secure setup. However, to support portable client operations where no keyboard is available, there is a hook to circumvent this restriction. To this end, a line of the form

`host_pwd=⟨pwd⟩`

has to be *manually* added to the file `remote.props` in the piHPSDR working directory. The password ⟨`pwd`⟩ will only be accepted if it is at least 5 characters long and will then be filled into the text entry field for the host password. It can be changed there, then the changed password will be written to `remote.props` but only if such as password has initially been successfully read therefrom, and if the (possibly updated) password is at least 5 characters long. This means, changing a password read from the props file to a short one effectively deletes it from the props file. I know that this procedure looks terribly inconvenient, and it is so by design. For general operation, storing passwords on client computers is not recommended and the hook described here is for those who need it and are aware of the possible security implications.

By default, the last button in the server line reads `Show` and the password is hidden (each character is represented by a bullet). If clicking the `Show` button, its label changes to `Hide` and the password becomes visible. You might have guessed that clicking the `Hide` button then conceals the password again and the button label reads `Show` again.

If all data has been entered, pushing the `Use Server` button starts establishing a connection with the server. The same happens if hitting `Return` in the password field.

12.3 Encryption and network security

The password you enter is **never ever** transmitted "as is" over the line. Instead, the server sends a random 512 bit (64 byte) *challenge* which is different for each connection. On the client side, the password is added to the challenge and a SHA512 key is built from the result. Using the last result as the new challenge, this process is repeated 99999 times to make brute-force attacks compute intensive, and the last hash generated is then sent back to the server. The server does the same with the password entered in the `Server` menu and then asserts that the SHA512 key thus generated matches the one received from the client. On a RaspberryPi5, generation of the hash takes about 120 msec, and about 3 times less on (more powerful) desktop computers. Once a hacker has "sniffed" a valid challenge/hash pair,

trying to guess the password from generating all possible hashes involves a lot of computation, depending on the quality of the password. It must be emphasised that this is **not** top-notch cyber security, but it should be good enough to fulfil your (legal) obligations as a radio amateur to prevent your gear from unlawful use. If you feel you must go beyond, one option is to set up a VPN *virtual private network* in which you can have expert-level (professional) cyber security provided by your network infrastructure, which is far beyond the amateur-level security features that I can build into piHPSDR.

12.4 Auto disconnect feature

Equally important is to ensure that your radio stops transmitting if the connection to the client breaks down. To this end, the server closes the connection and goes RX if it has not received any command from the client for about 30 seconds, or if the attempt to read the next command from the client failed (the client will send a "heartbeat" message every 15 seconds).

12.5 Network bandwidth

The user commands and even the spectrum data to be displayed in the panadapters and waterfalls are the smallest part of the data flowing between the client and the server. The (by far) largest part is the audio data. Audio data is transferred as signed 16-bit integers at a sample rate of 48 kHz, in mono both for RX audio (Server → Client) and TX microphone data (Client → server). During RX, and also during TX when doing CW, no TX microphone data is transferred. If not using duplex, no RX audio data is transferred when TXing because the receivers are shut down.

This means that during RX, about 96 kByte/s (768 kbit/s) are transferred from the Server to the Client (twice as much if running two receivers), while 96 kByte/s (768 kbit/s) are transferred from the Client to the Server during TX if doing SSB. When TXing in CW, very little data (the CW key-down/up time stamps) is actually sent from the client to the server. My first tests indicate that client/server operation works very well even if the connection

between Client and Server involves WiFi. I mention this because my attempts using WiFi for connection to the *radio* (that is, for the base band IQ data) were not entirely satisfactory.

12.6 Settings on the Client side

When the client connects to the server, most radio settings are transferred from the server to the client side. In the following, applying any changes on the client side are transferred to the server and utlimately stored in the settings (,,props file") on the server side. Some settings, however, are independent on the client and server side. These settings are read from the props file on the client when the client starts, and if they are changed, they are written to that file. These local settings include

- Screen height and VFO bar layout
- Layout of the slider and toolbar area
- Panadapter/Waterfall settings
- Local audio settings (sound cards, stereo/mono settings, etc.)
- All CAT/TCI/Midi settings
- GPIO settings (if GPIO is enabled)
- VOX settings and VOX threshold

Note the screen width must be the same on client and server, and is stored in the settings on the server side. This is so because the spectrum in the panadapter is calculated for a specific width (in pixels). The server will never be in full-screen mode, even if the client is. Note the complete RX/TX profiles are stored in the props file on the client side, but only the local audio settings contained therein are actually used.

Chapter 13

Latencies

In traditional (analog) radios, latencies are usually so short that they are not problematic. With SDRs, latencies are usually larger, mostly for a good reason (see below). So let us first describe which types of latencies we are talking about:

RX The RX latency is the time difference between a signal arriving at the antenna input, and the (demodulated) audio playing in the headphone. It could be measured by connecting the (strongly attenuated) output of a second transmitter to the antenna input, sending a stream of CW dashes, and monitoring both the output of the second transmitter and the headphone of the first radio on a two-channel oscilloscope. Alternatively, one can use a second conventional (analog) receiver with known low latency, feed the antenna signal to both devices, tune to a CW signal and compare the audio output of both radios with a two-channel oscilloscope.

TX This is the time difference between a signal arriving at the microphone input, and the (modulated) RF output at the antenna output of the transmitter. It could be measured by monitoring the microphone input and the RF output on a two-channel oscilloscope, and feeding the side tone of a string of dashes to the microphone input.

Audio The delays mentioned above are only in part caused by the signal processing within piHPSDR. There is also a delay between piHPSDR actually sending audio to the radio or to a sound card, and the signal

appearing in the headphone. The reverse holds for the microphone input signals. Latencies are introduced here on purpose and are a feature rather than a bug: Because of CPU stalling and many other reasons, it may happen that RX audio samples are produced in bursts with pauses in between, so one needs a buffer with variable filling to eliminate sample over- und underruns.

13.1 RF (base band) latencies

The notion of what is an acceptable RF delay very much depends on the kind of operation. In a rag-chew QSO, large delays (few hundred milliseconds) are not really harmful, but in a contest or DX pileup situation, it may be detrimental if your RF signal comes late. But is makes not much sense to try reducing the delay to, say, below 10 ms simply because the travel time of the RF signal between you and your QSO partner is already about 33 ms if the distance is 10 thousand kilometres.

To understand the source of the RF delay, we have to look at the fate of an RF signal pulse after it arrives at the antenna. The RF signal goes through the RF front end and eventually arrives at a ADC (analog to digital converter). The digitised signal is then processed by the FPGA in the radio and the baseband data is sent in packets to the host computer running piHPSDR over the ethernet. I have no solid numbers on the delay from the antenna input to when the signal arrives in piHPSDR, but I guess this is shorter than what follows.

piHPSDR typically collects 1024 such samples before dispatching them for processing, this amounts to a minimum delay of 11 ms at a RX sample rate of 96 kHz. Of course, one could collect samples in smaller numbers, but then the overhead associated with ,,dispatching" data grows.

When baseband data is fed to the input side of the digital signal processing (DSP) engine, it takes some time before the audio data appears at the output side. How long this is, is not primarily determined by by how many ,,taps" the signal processing chain contains. The number of taps required, for example, in filtering depends on the shape factor of the filter. The minimum value for the filters used in piHPSDR is 2048 taps (this can be increases in the `Filter` menu) which corresponds to 43 ms at 48 kHz which is the (fixed) DSP rate

in the RX engine. Any additional processing (equalisers, noise reduction, notch filter, etc.) will increase the DSP processing time. Faster CPUs do not decrease the latency because the samples come out of the DSP chain in the same pace they enter.

After having retrieved the audio signal from the DSP processing, it can be sent to the audio hardware. For radios having a built-on codec (that is, radios with a headphone jack), the audio data is sent from piHPSDR to the radio over the ethernet cable. How long it takes before you hear it in the headphone depends on the processing within the radio. If you connect a sound card to the computer running piHPSDR, the audio data is sent to the sound card and will eventually be heard in a headphone connected to that sound card. Again, this delay very much depends on the audio system of the computer, and audio delays will be discussed below.

Having said this, it might come as a surprise that there is an addition delay of about 100 ms built in by design. Basically, piHPSDR will not start producing audio on the sound card until it has collected (and buffered) about 100 msec of audio. The reason is, that the world is not ideal and the audio data is not produced at a constant pace. For example, the operating system of the computer may, for a moment, give too little CPU time to the DSP engine such that its output lags behind. Subsequently, all the pending data is processed, such that the audio data stream may be interrupted for a small period, and then a burst of audio samples arrives. Therefore, piHPSDR manages and audio buffer that can hold about 200 ms of audio data and try to keep it at half filling. This way, a period of up to 100 ms where no audio data arrives, and also a burst of 100 ms of audio data, can be tolerated without producing audio cracks. This part of the latency is actually a feature and not a bug: of course one can make the latency smaller, but at the expense of making audio cracks more likely.

The same holds for the reverse (TX) path. Here, the audio input (microphone) signal is fed to the input side of the DSP processor, and filtering/processing takes some time (the DSP sample rate in the TX chain is usually 96 kHz in Protocol 2). But again, the data that comes out of the DSP processor may not arrive at a constant pace. Therefore, a certain amount of ,,RF silence" is sent to the radio upon a RX/TX transition to pre-fill the outgoing TX data buffer. While not necessary for many radios (they do not start transmitting before the TX data buffer is sufficiently filled), this should

make under-runs of the TX data buffer in the FPGA less likely.

The bottom line of this section is, that a certain amount of latency cannot be avoided because it is inherent in the digital signal processing, but that this latency is further increased on purpose to establish a resilience against jitters (bursts and pauses) in the data flow, that may e.g. be caused by the signal processing thread getting no CPU cycles for a small amount of time etc.

13.2 CW side tone latency

The audio latency is of utmost importance when doing CW, no matter whether you use the iambic keyer built into piHPSDR is used, or whether you use an external keyer which sends key-up and key-down events via MIDI to the computer running piHPSDR. Imagine you want to sent a letter 'V' in CW, using a paddle. If the side tone arrives a little bit late at your ear, then the keyer may have advanced to the point where it sends the fourth dot before you have heard the third dot. Such a delay then leads to typical keying errors, e.g. sending a number '4' where one wants to send a letter 'V'. Since the length of a dot is 50 ms at a CW speed of 24 wpm, an audio delay of 100 ms (or even more) is certainly not acceptable when doing CW. In the audio modules (ALSA, PortAudio and PulseAudio/PipeWire), piHPSDR has a built-in mechanism to remedy this situation.

This is possible since when generating the CW side tone, audio samples come at a much more regular and predictable pace than when producing RX audio, such that we do not need a large latency to cope with small pauses and bursts of samples in the audio stream. Upon a RX/TX transition, most pending audio samples in the audio output buffer are discarded and the audio stream is re-started with a small latency. Furthermore, piHPSDR monitors the latancy and very slightly increases or decreases the pauses between the dots or dashes to keep the audio output buffer at the desired filling level. This way, a piHPSDR generated side tone can be used for CW speeds up to 24 wpm.

To demonstrate this, the side tone latency has been measured with a two-channel oscilloscope with piHPSDR running on a RaspPi5 with a USB sound card attached, see Fig. 13.1. piHPSDR was compiled with the ALSA sound

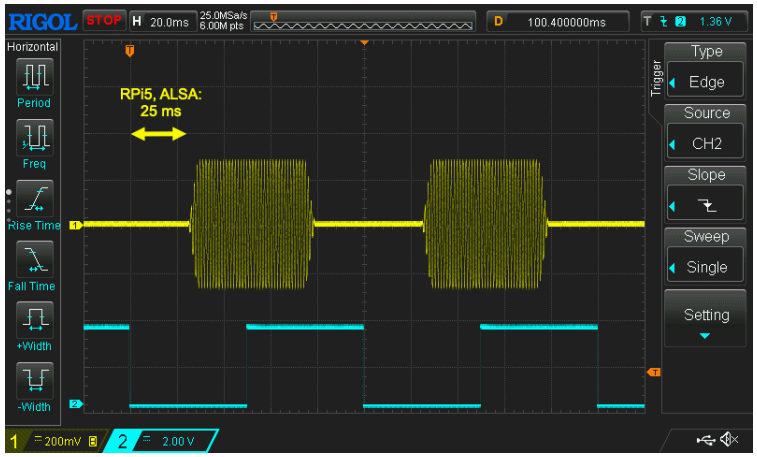

Fig. 13.1: Side Tone Latency with ALSA. Blue trace (bottom): CW key (active low); Yellow trace (top): signal at the headphone.

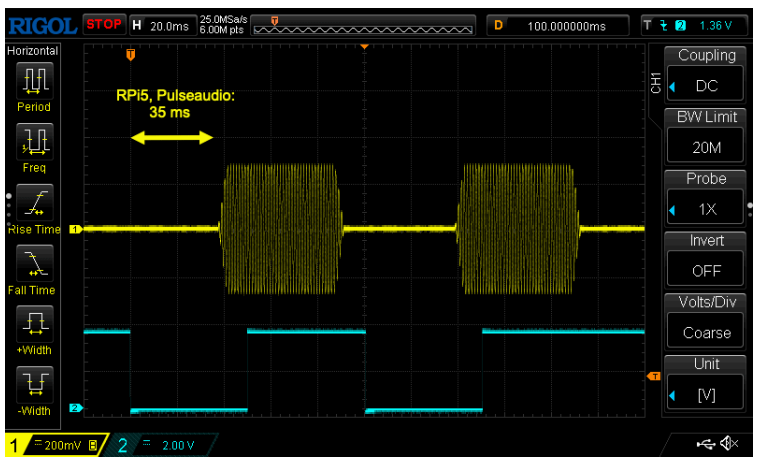

Fig. 13.2: Side Tone Latency with PulseAudio. Blue trace (bottom): CW key (active low); Yellow trace (top): signal at the headphone.

module, and the internal (iambic) keyer was used at a speed of 24 wpm (dot length: 50 ms). A GPIO active-low output line (Channel 2, blue trace) was used to report the state of the keyer and to trigger the scope. Channel 1 (yellow trace) then displays the signal at the headphone. The measured delay is 25 ms, about half the dot length. With the PulseAudio module (Fig. 13.2) a somewhat larger delay (35 ms) was obtained, while on a Macintosh (using the PortAudio module), a slightly smaller delay (20 ms) could be achieved.

For CW speeds up to 24 wpm, this is usable but not optimal, and high-speed CW operators certainly want delays smaller than 10 ms (see next section).

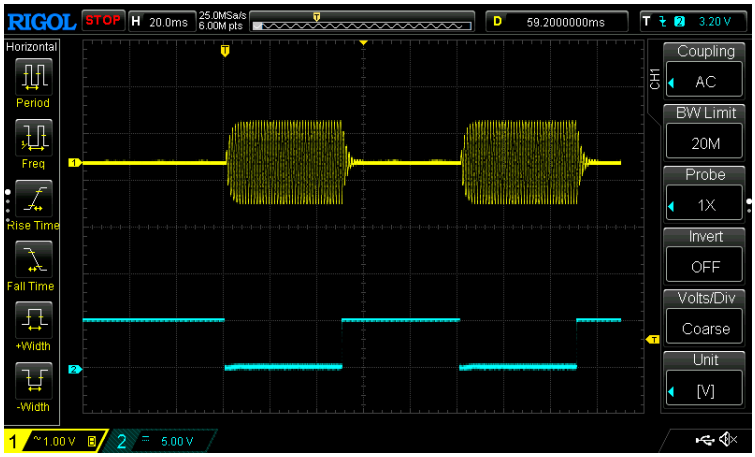

Fig. 13.3: Zero Latency from an ,,external" side tone.Blue trace (bottom): CW key (active low); Yellow trace (top): signal at the headphone.

13.3 External solution to CW side tone latency

Reducing the delay down to below 20 ms is probably possible with dedicated low-latency audio hardware and specialised audio drivers, but this is beyond the scope of piHPSDR. The most stringent solution to the low side tone latency problem is to generate the side tone externally, thus *not* using the side tone generated by piHPSDR. On a RaspBerry Pi with GPIO outputs, there is an active-low `CWOUT` GPIO line (see Appendix H) that could be used to trigger an external side tone generator whose output is then mixed into the headphone cable. On computers that do not have GPIO input and output lines, or if using controllers that reserve all those lines for themself, I use an external keyer built around a micro-controller, which sends the key-up and key-down events via MIDI to the computer running piHPSDR. You will find lots of examples for micro-controller (uC) codes implementing a keyer on the internet, and for these codes, you can usually enable the generation of a square wave zero-latency side tone at one of the uC output pins. This square

wave side tone is not pleasant to listen to: the harmonics contained in the square wave do not sound nice to operators expecting a clean sine-shaped signal, but the real problem is the sharp rise and fall of the side tone pulse envelope which leads to harsh audio clicks. Both problems are eliminated if one routes the side tone through a narrow audio band pass filter, that is easily realised with a single OP amp.

Using two simple AF transformers, the side tone can then be mixed with the (stereo) RX audio from the headphone cable. Fig. 13.3 shows the measured result of such a setup. The blue trace, used for triggering, shows the keying output of the external keyer. One immediately sees that the side tone latency is virtually zero. The pulse envelope of the signal measured at the headphone is characteristic for audio filters realised with OP amps, the envelope depends on the chosen bandwidth that can be adjusted: the band width has to bee increased (decreased) if the side tone sounds too sloppy (harsh). The only drawback is that the side tone then must have a fixed frequency that matches the centre frequency of the audio band pass filter. This is not a problem since most CW operators have their preferred side tone frequency which does not change from one day to the other.

Of course, this setup implies that you set the side tone volume to zero in the `CW` menu, such that the side tone generated by piHPSDR does not interfere. You may still want to have a piHPSDR generated side tone for CAT-generated CW messages. To address this, piHPSDR will use a low default volume for those messages if the side tone volume has been set to zero in the `CW` menu. Note that side tone latency is not important for CAT generated CW messages.

There is another, more sophisticated, ,,external" solution to the side tone latency problem which is built around the powerful Teensy4 uC. It is described in

`https://github.com/softerhardware/CWKeyer`

Briefly, the uC is so powerful that it can run both a CW keyer software, and act as a USB sound card to the computer running piHPSDR. If one chooses this device for RX audio output, the CW side tone from the keyer will be mixed by the keyer software into the RX audio stream in software. The side tone latency obtained this way is not zero but very small (about 5 ms).

Chapter 14

RX/TX profiles

Many settings such as VFO step sizes, filter choices, noise reduction settings, equaliser settings, and TX compressor settings are only reasonable for a specific mode and need be re-adjusted if one changes the mode. For example, one might wish to use different VFO step sizes for different modes, and get the step size chosen for that mode automatically if one switches to that mode. The same applies to noise reduction settings which are very different for CW and SSB. For digital modes (DIGU/DIGL), one normally does not want neither noise reduction or equaliser being active on the RX side, nor a speech processor for transmitting. In addition, one usually uses microphone and headphone for USB audio input and output, and wants to automatically switch to virtual audio cables for DIGU.

piHPSDR assumes that the user ,,wants" the same settings for USB and LSB, or for DIGU and DIGL. Therefore three mode groups have been defined that encompass USB/LSB/DSB, CWU/CWL, and DIGU/DIGL. There are two more mode ,,groups" with AM and FM as the only one member.

Each mode has an associated RX/TX profile containing lots of settings (see below). When switching a mode (either by an explicit request, or implicitly by a VFO swap/copy operation, a band or band stack change, or by recalling a memory slot), the profile of that mode overwrites the current settings.

The RX/TX profile for a specific mode (group) is updated whenever one of the settings contained in the profile is updated for RX1 or for the transmitter. Note that the mode-specific RX/TX profile is also applied to RX2 if the mode

of RX2 is changed, but changing the settings for RX2 does not update the profile. The exception is the RX audio output device, here changes are only stored for RX1 and only the RX2 audio output device is possibly changed when the RX1 mode changes.

For example, if you have adjusted the noise reduction settings while operating USB, these settings are stored in the RX/TX profiles of the USB, LSB, and DSB modes. Whenever switching to USB, LSB or DSB in the future (until the profile is changed again), these settings become active. So you can quickly cycle, say, between USB, CWU and DIGU and upon each mode change you get the settings that you have chosen for that mode.

Here is the list of settings that form the RX/TX profile:

- VFO step size
- VFO RIT step size
- RX Filter (which one to use)
- RX CW peak filter (on/off)
- RX Noise reduction (all settings)
- RX Noise blanker (all settings)
- RX Automatic notch filter (on/off)
- RX Spectral noise blanker (on/off)
- RX AGC characteristic (slow/medium/fast etc.)
- RX squelch (on/off, squelch level)
- RX equaliser (all settings)
- RX1 audio output device and stereo/mono setting (only RX1, not RX2!)
- TX equaliser (all settings)
- TX speech processor (on/off, compression level)
- TX continuous frequency compressor (CFC, all settings)
- TX downward expander (DEXP, all settings)
- TX mic gain slider position
- TX filter settings (low/high or use RX filter)
- TX audio input device

Chapter 15

The TX audio chain

It seems that how the TX audio chain works is a field where myths and facts are equally dominant, therefore this section tries to shed some light on this. We will discuss the most relevant parts of the TX chain which process the input signal in the following order

- The downward expander (DEXP)
- The Mic gain slider (MicGain)
- The equaliser (TXEQ)
- The auto-leveller (AUTOLVL)
- The continuous frequency compressor (CFC)
- The speech processor (CMPR)
- The controlled envelope SSB overshoot control (CESSB)
- The automatic level control (ALC)

In piHPSDR, there are no user-accessible controls for AUTOLVL and CESSB as they are activated automatically when appropriate (see below).

For digital modes, the input audio level is fairly constant and can be set to be just below full scale clipping. In this case, the Mic gain slider can stay

at 0 dB and CMPR, CFC, DEXP, and TXEQ should all be switched off. If the audio level is constant but below full scale, the input signal can be amplified via MicGain until the TX ALC value is just below or at 0 dB. All what follows is concerned with the potential problems of human speech that comes from a microphone. Note that for FM mode, things are slightly more complicated because of the FM pre-emphasis, this is explained in chapter 9.1.1, see the explanation of the `FM PreEmp/ALC` button.

15.1 Start of the TX chain: The MicLvl indicator

At the beginning of the signal chain, there invariably is an analog-to-digital converter (ADC) in the sound card (or the codec) the microphone is connected to via an analog wire (the ADC may be built into the microphone in the case of USB microphones). The ADC has a maximum input amplitude beyond which clipping sets on, and the digitised microphone signal at the beginning of the chain is usually represented by a finite range of integers (usually a signed 16-bit or 24-bit number). This input signal may or may not be processed by the computer's operating system and eventually arrives in piHPSDR, where it is converted to a floating point number in the range $-1.0 \ldots +1.0$. Here we shall denote a signal whose peak amplitudes are ± 1.0 a *full scale* (FS) signal. Due to the clipping of the ADC, the signal cannot exceed FS unless there is a sort of volume control in the host operating system. No matter whether the adjustment is done in hardware (microphone preamp) or in software (volume control in the operating system), it should be such that you reach FS if you violently whistle into the microphone. Since clipping leads to extreme distortion, most audio gear is levelled such that you stay significantly below FS during normal speech. Note that once you represent the signal as a floating point number, you can apply virtually any amplification without introducing distortion, so from here on until close to the end of the TX chain, clipping is not an issue.

It is clearly of interest how the peak input levels are, and this is displayed in the `Mic Lvl` meter that shows up at the top of the meter (to the right of the VFO bar) during TX. This is the input level that piHPSDR "sees" before any processing takes place. The scale of this meter is $-40 \ldots 0$ dBFS, so at

normal speech it should show more than half-deflection. If it stays below, this can be compensated by advancing MicGain to positive values. But in this case you may want to first check whether the operating system applies extra attenuation (PulseAudio/PipeWire: `pavucontrol` or `wpctl` program, MacOS: Audio/MIDI setup utility). Dynamic microphones have rather low output levels, so adding a battery powered preamplifier (I have built a small one that could be fitted inside the microphone stand) could also be useful. To warn about possible clippings (integer overflows) of the microphone signal, the `Mic Lvl` bar is shown in green colour if peak levels are below -3 dB FS (71% of the peak amplitude), a yellow colour is used between -3 and -0.5 dB, and a red colour if peak microphone levels are above -0.5 dB FS (94% of the peak amplitude). For example, when using a microphone, connected to the radio, with an adjustable preamplifier, its gain should be reduced if the `Mic Lvl` bar regularly flashes yellow or red. This can also occur if a microphone is connected to the radio has the extra (20 dB) mic level boost (see the `TX menu` engaged but which does not need it. On the other hand, for digital modes when using a virtual audio cable, the peak level is about 100% almost constantly and here it is no reason to worry if the `Mic Lvl` bar is constantly red.

15.2 Monitoring the TX signal

The TX audio signal, as it comes from the microphone or from a digi program, can be mirrored to the audio output of the active receiver if not running `DUPLEX`. To this end, check the `TX Audio Monitor` box in the `TX` menu (see Chapter 9.1.1). With this you can check whether the microphone preamplifier is too weak or too strong, or whether transmitted RF finds back its way to your microphone cable and leads to RFI distortions. This does *not* check the audio you are actually transmitting.

It is (in principle) possible to monitor the audio signal downstream in the TX chain, that is, after applying compression, equaliser, etc. But this then comes with a large delay so this is not implemented. But it is possible to monitor the TX signal even without using a second receiver or a web SDR: just have the piHPSDR receiver tuned to the TX frequency and mode while using `DUPLEX`: in most cases, the cross talk at the T/R relay is by far strong enough such that you can hear your transmitted signal in the receiver. This

signal then went from the piHPSDR transmitter through the FPGA, the DAC converter and the PA to the T/R relay, and then back through the RF front end, the RX ADC and the FPGA to the piHPSDR receiver.

15.3 The downward expander (DEXP), a flexible noise gate

The first processor in the TX chain is the downward expander (DEXP). This is best simply viewed as a noise gate. Well, it is a fancy noise gate. It only opens if there is input signal of sufficient strength in a certain frequency window (see chapter 9.1.3, the default is 1000 ... 2000 Hz), so even strong low-frequency noise (such as the noise from a heavy PA fan transported to the microphone stand through the desktop) won't open it. Furthermore, when the gate is closed, it does not just block the input signal but applies a non-linear amplification. Usually you will have the DEXP activated with default settings for phone operation and it is not discussed further here.

15.4 MicGain and peak TX ALC value

The problem is now that the TX chain needs data that is about 0 dBFS. If it is below, you have not only low output power (in SSB) or thin modulation (AM, FM) but also parts of the machine, e.g. the speech processor (CMPR) or the adaptive pre-distortion (PURESIGNAL) won't work. So on one hand we should stay well below FS do avoid clipping in the ADC, but on the other hand we need amplification to reach FS. At the very end of the TX chain, the base-band signal is converted again from a floating point to an integer representation (in the range $\pm 2^{23}$) so it is critical to ensure that the base-band signal (in floating point representation) does not exceed ± 1.0. This is where the ALC (automatic level control) at the end of the TX chain sets in: it attenuates the base-band signal such that it "fits". The optimum level is reached when the ALC (peak) value is about zero, since this means the signal is about FS (negative ALC peak reading indicate a signal below FS, while positive readings indicate it is above FS). This is why the TX meter shows ALC info. The TX ALC peak value is most useful here. The operator

should then adjust the TX signal chain (e.g. slowly increasing or decreasing MicGain) until the TX ALC peak value is about 0.

Note that the TX meter can both show the TX ALC peak and average values, at this place the peak value is important. So in the simplest case, the recipe for adjusting the TX input chain is: Slowly adjust MicGain until the TX ALC peak is about zero when applying normal speech to the microphone.

15.5 The TX equaliser

Then comes the TX equaliser, which is described in chapter 10.2 and I guess any reader knows what an equaliser is. What is relevant here is, that the filtering involved in the equaliser changes the peak signal amplitude. This could be compensated by changing the MicGain slider, but the preferred way is to adjust the frequency independent gain of the equaliser such that the peak amplitude remains unchanged when switching on/off the equaliser.

15.6 The auto leveller

The next important part is the auto leveller (AUTOLVL). It solves the problem that the input signal may vary in amplitude, for example because the distance between your mouth and the microphone is not really constant. So if the signal is below FS, the auto leveller applies an extra amplification (up to 8 dB, this is hard coded in piHPSDR) so it "levels out" the signal if the input has varying strength. In piHPSDR, there is no user-accessible control for the auto leveller, it is automatically engaged if you use the speech processor (CMPR) or the continuous frequency compressor (CFC). So if you want the auto leveller but no compression just activate CMPR with a compression value of 0 dB.

15.7 The speech processor (CMPR, CFC)

The continuous frequency compressor (CFC) need not be discussed in detail here. It is a generalised version of the speech processor that applies a

frequency-dependent compression value. The CFC settings are part of the TX menu and are described in chapter 9.1.2. Here we discuss the ,,normal" speech processor, also termed compressor (CMPR).

Normal speech is very different from a constant sine signal. The amplitudes are greatly varying, and the average power emitted by your transmitter will be considerably below its PEP rating even if the peak amplitude of your TX signal reaches FS. The bad news is, there is nothing you can do to increase average TX power without distorting your audio signal. The good news is, you can increase average TX power if you accept that your audio signal is distorted. While it is a little more involved, imagine that the input signal is amplified beyond FS and then clipped at FS. This increases the average power but introduces a lot of distortions. If the input signal is already filtered such that it is confined, say, in a frequency window from 150 to 2850 Hz, then the compressed signal will contain components outside of these limits. This is taken care of by introducing additional filtering after compression.

Having said this, it should be clear that one probably does *not* want compression in a rag-chew local QSO and in all situations where the transmitted signal should be as natural as possible, while compression is often activated in contests of when working a DX station with a pile-up. The speech processor can be switched on/off in the TX menu, and a compression level between 0 dB (no compression) and 20 dB (very strong compression) can be chosen. In my experience, a compression level of 8 dB is usually enough and still tolerable, but this is a very personal notion.

Since the speech processor implicitly assumes that the input signal peaks at FS, the auto leveller is always used if the speech processor is activated. It therefore seems a good idea to always use the compressor when operating phone, but to select 0 dB compression level when you do not want compression.

15.8 The controlled envelope SSB overshoot control (CESSB)

Ideally, the compressor produces an output signal that peaks at FS. This effect can nicely be seen when the TX ALC meter is set to display the peak

ALC value. If one cranks up the Mic gain without using compression, then the peak ALC value will assume large positive values, indicating the the ALC at the end of the chain has reduced the signal amplitude. When using a compressor and increasing the mic gain, the peak ALC value will reach zero but ideally not go beyond. However, the additional filtering that has to be applied after clipping in the compressor may introduce some artefacts, namely that the signal after filtering exceeds FS. In an experiment where I temporarily deactivated CESSB the peak TX ALC value went up to 5 dB when using a large mic gain, a compressor level of 20 dB, and speaking violently into the microphone. This means that at the end of the TX chain, the compressed signal was attenuated by 5 dB and this is not what one wants.

This is where CESSB sets in: it removes any overshooting after the post-compressor filtering, such that the peak ALC value is again close to zero. piHPSDR automatically activates CESSB if the speech processor is enabled and the compression level larger than 5 dB. This border line has been chosen to keep speech as natural as possible for small compression levels, while preventing the ALC to reduce the output RF power at large compression levels.

Not that piHPSDR does not auto-engage CESSB when using the CFC. The idea here is that those who want to use the fancy CFC do not want their settings to be affected by auto-engaging CESSB.

15.9 The automatic level control (ALC)

At the very end of the TX audio signal chain, it has to be *guaranteed* that the base-band signal does not exceed FS. The ALC does nothing if the signal at the end of the chain stays below FS, and it attenuates the signal to not exceed FS in the other case. The TX ALC peak meter shows the peak amplitude *before* ALC, so it is positive (negative) if the input signal is too strong (weak).

piHPSDR also offers the possibility to show the average TX ALC value. This is the average output power w.r.t. to a FS single-tone signal, so for a correctly chosen PA calibration value this is the average output power w.r.t. to the PEP rating of your PA. It reads zero if you feed a FS single-tone sine signal to the TX chain, it reads -3 dB if you feed a two-tone signal consisting of two

sines with the same (half of FS) amplitude and different frequencies. Choose this reading to assess the efficiency of the speech processor: narrowing the gap between the peak and average value is what the speech processor is supposed to do. Note that the average TX ALC value can never be larger than zero.

Chapter 16

Audio recording and replay

Audio recording and replay is typically used in QSOs to let your QSO partner listen to its own transmission, and how it arrived at your site. It is not meant to do automatic SSB CQ calls etc., here a dedicated ,,voice keyer" software should be used. The main restriction of the record&replay facility in piHPSDR is that only a single piece of audio can be stored, and that the maximum length of this piece is about 20 seconds.

Because the main purpose of this feature is to play back a recorded transmission of your QSO partner, the RX equaliser is not used while recording (but noise reduction etc. is still active!). Likewise, when replaying, the TX equaliser, the TX speech processor, the continuous frequency compressor (CFC), and the downward expander (DEXP) are temporarily deactivated. Furthermore, the recorded audio is normalised such that it peaks at full amplitude, and therefore, the TX mic gain is set to 0 dB while replaying.

The piHPSDR recording and replay feature requires a single button triggering the `Capture` command, so this command must either be mapped to a MIDI/GPIO push-button or to one of the toolbar buttons. There is another command, `Replay` (see below), that is mostly equivalent to `Capture` except when RXing: here it does not start a new recording but plays the recorded audio in the audio output of the active receiver. If you do not need to check the recorded audio before you transmit it, you do not need the `Replay` command. To avoid blind flying, the status of the recording/playback is indicated in the top right corner of the pan adapter in the main window (waterfall-only setups won't show the record/playback status).

In the normal ,,idle" situation, nothing is shown on the pan adapter. This is the case if the Capture or Replay buttons have not yet been used since the start of the piHPSDR program, or if the last recording, transmit or playback ended more then 30 seconds ago. If noting is on the screen, one has to push the Capture or Replay button once and a string Available string in yellow colour will appear in the top right corner of the pan adapter, and below a horizontal progress bar that indicates how long the actual recording is: a full bar corresponds to maximum filling which corresponds to 20 seconds, but this can easily be changed at compile time. Fig. 16.1 shows the status bar, as it has been brought up by hitting the Capture or Replay button. It is important to note that when no status bar is on display, this first button press only brings the status bar back but does not start any action.

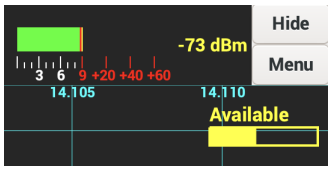

Fig. 16.1: The Capture status bar.

As can be seen, the length of the currently stored recording is about half of the maximum length. If hitting the Capture or Replay button for the very first time, the length of the stored recording is of course zero. If the Available string is shown in the pan adapter, hitting the Capture button again will either start a new recording (while RXing) or start transmitting the recorded audio (if hitting the button while TXing). If a new recording is done, the status string will change to Record and the yellow progress bar will fill from the left to the right. Recording will stop at the latest if the bar hits the right end (that is, if the audio buffer is full), but can also by stopped before that if hitting the Capture or Replay button. During transmit, the status string changes to Transmit (written in red colour), the progress bar turns read and starts from the left until (at the latest) all of the actual recording (marked by a small vertical yellow line) is completely replayed (Fig. 16.2). Again, the transmit can be stopped any time by hitting the Capture or Replay button.

After the transmit is done, the status line again changes to Available (see Fig. 16.1) and the recorded audio can, if one wants, be transmitted again. If the status line has been shown for about 30 seconds without any recording

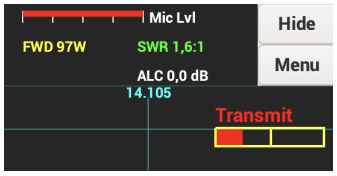

Fig. 16.2: Transmitting recorded audio data

or transmitting taking place, it is hidden and can be brought up again by hitting the `Capture` or `Replay` button.

The big advantage of this user interface is that only a single button (`Capture`) is needed. However, sometimes one wants to check what has actually been recorded, and then one needs the `Replay` button. While TXing, there is no difference between `Capture` and `Replay`, but while RXing, hitting the `Replay` button will replay the recorded audio in the audio stream of the active receiver. For Replay, the status bar looks similar to the transmit case except that it is labelled `Replay`.

So the things to remember are:

- If no recording/transmit/replay status is shown in the top left of the pan adapter, pressing both the `Capture` or the `Replay` shows it. Note no playback, transmit, or replay process is started.
- If the status bar has a yellow label reading `Available`, then the bar indicates how much audio data has been recorded. If RXing, hitting `Capture` deletes the recorded audio and starts a new recording, and hitting `Replay` replays the recorded audio in the audio output of the active receiver. If TXing, hitting either `Capture` or `Replay` starts transmitting the recorded audio.
- If recording (status text = `Record`), transmitting (status text = `Transmit`) or replaying (status text = `Replay`), a moving progress bar shows how much audio data has already been recorded, transmitted, or replayed.
- Recording automatically stops if the buffer is full, and transmitting and replaying automatically stops if all recorded audio has been transmitted or replayed. In all cases, hitting either `Captur` or `Replay` immediately stops a running recording, transmit, or replay.g

Chapter 17

HermesLite-II and RadioBerry support

A. https://github.com/softerhardware/Hermes-Lite2/wiki/Releases
B. https://james.ahlstrom.name/hl2filter/
C. https://www.hermeslite2plus.com/
D. https://github.com/jimahlstrom/HL2IOBoard

Note. piHPSDR mostly does not distinguish between the HermesLite-II and the RadioBerry. The only difference is the automatic disabling of most of the GPIO lines for the RadioBerry, see the last section in this chapter.

The HermesLite-II (HL2 for short, see internet link A above) is an open software / open hardware project, a small SDR radio with a QRP (5 watts) PA and covering the HF bands up to about 38 MHz. Unlike standard HPSDR boards, it does not have a fixed pre-amplifier and a programmable step attenuator in the RF front end, but instead a single combined preamp/attenuator that goes from -12 to +48 dB (a negative value indicates attenuation). The RX calibration value (see the `Radio` menu) which is around zero for standard HPSDR boards must be in the range 11–14 for the HL2, depending on the filtering in the RF front end (a value of 14 is the default one if a HL2 is detected). The sliders area of piHPSDR does not show an attenuation (`ATT`) slider, but instead a `RF gain` slider that encompasses values from -12 to +48. A value of about 14 lets the HL2 behave similar to a standard HPSDR radio with the `ATT` slider at zero. Note that when using adaptive pre-distortion (PURESIGNAL), the whole range of these values is used when auto-levelling

the RX feedback signal.

Normally, the HL2 is used together with the N2ADR filter board (internet link B) featuring the low-pass filters that are required before connecting the HL2 to an antenna. Some have also included the so-called AK4951 companion board (internet link C) that features, among others, an audio codec so you can connect a headphone and a microphone. There is also a more recent HL2 I/O-board (internet link D) that offers a variety of I/O resources. All these boards are supported by piHPSDR. Several HL2-specific settings can be made in the `Radio` menu (chapter 6.1) and are discussed there.

17.1 Default PA calibration values

The default value for the PA calibration value (see the `PA` menu) is 53.0 dB. The (by far) most prominent problem for beginners with the HL2 and piHPSDR is, that it seemingly produces no RF output. The reason for this is, that well-adjusted PA calibration values for a HL2 are rather small (about 40.5 dB) which is so much below the default value that this leads to essentially no RF output. While this would not be a problem if users read the manual carefully and went to the PA menu first, this is obviously not the case. Therefore, as a bonus to HL2 users, the default PA calibration value (for all bands) is reduced to 40.5 dB for HermesLite-II (and RadioBerry) radios. This only applies at the very first start of piHPSDR, since in later runs the default value will be overwritten anyway when reading the settings from the props file.

17.2 Support for the HL2 I/O board

The HL2 I/O board (internet link D) is a very versatile add-on to the HL2. It is so versatile because it contains a micro-controller (a Raspberry Pico) with a lot of I/O lines. Depending on which program it runs, the Pico can switch antennas, transverters, or external filter boards (based on the RX and TX frequencies), deliver a band voltage to the PA (based on the TX frequency), control an automatic antenna tuner, and so forth. All this is *not*

done within piHPSDR, but of course the Pico needs some information from piHPSDR to work. If the presence of an HL2 I/O board has been detected, piHPSDR transfers data in regular intervals and in round-robin fashion to the I/O board. Note that such a round trip can take about half a second, so it may happen that the I/O board shows some delay when switching PA band filters or engaging an auto tuner. This data is simply written into some internal registers of the IO-Board, and it is up to the Pico to make good use of it.

The IO-Board registers updated by piHPSDR are:

- TX Frequency (registers 0 to 4)
- Antenna Tuner (register 7)
- RF input path (register 11)
- RX1 frequency code (register 13)
- RX2 frequency code (register 14)

TX frequency. The TX frequency is a 40-bit word that contains the (dial) TX frequency in Hz. When using a transverter, this it *not* the frequency at which the HL2 operates.

Antenna Tuner. When starting TUNE-ing and the `HL2 AH4 ATU` box is unchecked in the `Radio` menu, a '1' is written into register 7. It is up to the HL2 IOB firmware in the Pico to take care that an external automatic antenna tuner starts its tuning sequence. For piHPSDR, it is simply ,,fire and forget".

RF input path. If for the current band, anything different from `Ant1` has been chosen as the current RX antenna, then the `Alt RX` (alternate RX antenna) input jack is used to feed the HL2 receiver. If using PURESIGNAL, the PS feedback signal can be connected to the `PURE` input jack in either case.

RX frequencies. The RX1/RX2 frequency codes encode the RX1/RX2 dial (!) frequency f in a one-byte value ("band code") C that is defined as

$$C = \text{Floor}\,(0.5 + 15.47 * \ln(f/18748.1))$$

so frequencies in the 80m band generated code 81 or 82, frequencies in the 2m band code 138 or 139, and the 3cm (10 GHz) band generates a band code that is 204.

17.3 "TX latency" and "PTT hang time"

There has been much discussion about which are the optimal values for these parameters. piHPSDR has no user interface to change these values but rather sets the TX-latency to 40 msec and the PTT hang time to 20 msec, and a modification of the source code and recompilation is required if you are not happy with these values.

After a RX/TX transition, the HL2 does not start transmitting until the internal buffer (within the FPGA) for the TX samples reaches a minimum filling, this is the TX latency. A too low value may lead to frequent TX buffer under-runs especially for a sloppy network connection, a too high value does not only delay the TX signal but also risks TX buffer overflows. Experience in the HL2 discussion group seems to be that the default value of 20 msec of the HL2 firmware is somewhat small, so piHPSDR sets the TX latency to 40 msec. My measurements indicate that the buffer can hold about 75 msec of TX data so a value of 40 msec should leave enough headroom.

The so-called PTT hang time stops transmitting if the TX sample buffer became empty for the specified amount of time. Together with a too small TX latency value this leads to "relay chatter" during transmit if the network connection is sloppy because the T/R relay is frequently switched. With a decently chosen value of the TX latency, the PTT hang time is not very critical so I set it to 20 msec.

17.4 HL2 parameters in the panadapter

The HL2 transfers some non-standard information about its state to the host program (piHPSDR) that can be displayed. If the box `Display PA current` is checked in the `Screen` menu (see Fig. 6.3), the PA temperature and the PA current is shown (while transmitting) in the panadapter in the upper left corner in yellow colour. If `Display Warnings` is checked in the same menu,

then in addition to the usual warnings, also an under-run or an overflow of the FPGA TX sample buffer is reported.

17.5 RadioBerry support

Information on RadioBerry project created by Johan PA3GSB can be found on his home page `https://www.pa3gsb.nl/`. It is built around the same Analog Devices modem chip AD9866 as the HermesLite-II (see previous section) and even identifies itself as a HermesLite-II. So from the viewpoint of an SDR program such as piHPSDR, there is little difference between the RadioBerry and the HermesLite-II.

Normally the RadioBerry hardware is a "hat" that is mounted on a RaspberryPi computer which runs the drivers for the RadioBerry but which also can run the SDR software. The communication between piHPSDR and the RadioBerry driver is via the local loop-back internet interface. Normally piHPSDR does not query loop-back devices when discovering radios, but to support the RadioBerry, local loop-back devices are queried for protocol-2. It is even easier (and faster) just to insert the IP address 127.0.0.1 in the `Radio IP addr` field in the discovery menu.

The communication between the RaspberryPi and the RadioBerry uses nearly all GPIO lines of the RaspPi, so the general recommendation is to compile piHPSDR without GPIO support. However, two GPIO lines are actually available, namely

- GPIO14/15 for RadioBerries with firmware versions smaller than 73.2,
- GPIO17/18 for those with firmware versions 73.2 to 74.9,
- and GPIO 17/13 from firmware version 75.0 on.

Therefore, if piHPSDR is compiled with GPIO support and detects a RadioBerry "hat" from the presence of the special device file `/dev/radioberry`, then it enforces the `No Controller` GPIO choice and only uses the two available lines for `CWL/CWR`, so one can attach a paddle to the two lines and use the iambic keyer built into piHPSDR. For non-portable use my suggestion is to use an external MIDI keyer and compile piHPSDR without GPIO.

Appendix A

List of piHPSDR Commands

In this chapter, we give a list of commands implemented in the piHPSDR program. These commands can be assigned to toolbar buttons on the screen, or push-buttons/encoders of a GPIO-connected or MIDI controller. Not all commands can be assigned to all control elements. Changing the AF volume, for example, can only be assigned to a knob which you can turn, while switching RIT on/off can only be assigned to a button that you can push. For each command in the following table, there is a long and a short string assigned. The long string will be used when there is enough space, while the short string is used for small buttons and to store commands in preference files (therefore the short strings never contain a blank character or a line break). Then, for each command we give the type of control element allowed for this command as a combination of the letters B, P, S, E, which stand for

`B` "Button": A button in the toolbar, or a push-button or switch on a GPIO or MIDI connected console

`P` "Potentiometer": A potentiometer or a slider on a MIDI connected console

`S` "Slider": A slider in the Slider area

`E` "Encoder": A rotary encoder on a GPIO or MIDI connected console

The main difference between a "potentiometer" and an "encoder" is, that the former reports a value in a predefined interval between a minimum and a

maximum, while an encoder can be turned in either direction without stopping. This means that a potentiometer reports a value (between minimum and maximum), while an encoder reports an increment, that is, whether it has been turned clock wise or counter clock wise. The existing GPIO consoles do not have potentiometers (most likely because of the lack of analog inputs), but many MIDI consoles do have, and Arduino-based MIDI controllers might have it because Arduinos have analog inputs to which a potentiometer can be connected.

To give an example, controlling the TX drive can be down both with a slider and with an encoder. While for a slider/potentiometer, the values from min to max are simple mapped to the TX drive values from 0 to 100, the signals from an encoder will just increase or decrease the value until one of a limits has been reached.

All ,,button" commands can be assigned to toolbar buttons via the TOOLBAR menu (see Sec. 11.1). Some (but not all) ,,potentiometer" commands can be assigned to sliders in the Slider area. Note that some ,,button" and some ,,potentiometer" commands can also be triggered through check-boxes, sliders, or spin-buttons in the menus. This is described in the preceding sections.

In the following, the commands are alphabetically sorted by their long name, with the "empty" command listed first.

NONE	NONE	BPSE
This is a command which does nothing. It can be assigned to buttons or encoders that are often accidentally operated. Some MIDI consoles, for example, report a button press event if the VFO knob is touched, and this we want to be able to ignore.		

A<>B	A<>B	B
Swap VFOs A and B. This will not only swap the frequencies, but also all other settings associated with that VFO, such as mode, filter, CTUN, and RIT settings.		

A<B	A<B	B
Copy VFO B to VFO A.		

A>B	A>B	B
Copy VFO A to VFO B.		

AF Gain	AFGAIN	PE
Change the AF gain (headphone volume, $-40 \ldots 0$ dB) of the active receiver.		

AF Gain RX1	AFGAIN1	PE
Change the AF gain (headphone volume, $-40 \ldots 0$ dB) of the RX1 receiver.		

AF Gain RX2	AFGAIN2	PE
Change the AF gain (headphone volume, $-40 \ldots 0$ dB) of the RX2 receiver. This operation is a no-op if only one receiver is running.		

AGC	AGCT	B
Cycle through the AGC modes (slow, medium, fast, ...) of the current receiver.		

AGC Gain	AGCGain	PSE
Change AGC gain ($-20 \ldots 120$ dB) of the active receiver. Changing the AGC gain moves the horizontal green line (doted by a green 'G') on the pan adapter. Its optimal position is at or slightly below the noise floor level.		

AGC Gain RX1	AGCGain1	PE
Change AGC gain ($-20 \ldots 120$ dB) of RX1.		

AGC Gain RX2	AGCGain1	PE
Change AGC gain ($-20 \ldots 120$ dB) of RX2. This is a no-op if only one receiver is running.		

AGC Menu	AGC	B
Opens the AGC menu.		

ANF	ANF	B
Toggles the state (on/off) of the automatic notch filter for the active receiver.		

Atten	ATTEN	PSE
Changes the value (0-31 dB) of the step attenuator of the active receiver. This function is only available for radios that have such an attenuator. Increasing the attenuation is required if ADC overload occurs, otherwise there is little reason for having high attenuation. If the attenuation value is changed for RX1, this is stored "with the band" in a database. Changing the band for RX1 will set the attenuation from that database.		

Band 10	10	B
Change band of the active receiver to the 10m band. If already on that band, move to the next band stack entry. This command is a no-op if the frequency of the band falls outside the frequency range of the radio.		

Band 12	12	B
Change band of the active receiver to the 12m band. If already on that band, move to the next band stack entry. This command is a no-op if the frequency of the band falls outside the frequency range of the radio.		

Band 1240	1240	B
Change band of the active receiver to the 1240 MHz (23 cm) band. If already on that band, move to the next band stack entry. This command is a no-op if the frequency of the band falls outside the frequency range of the radio.		

Band 136	136	B
Change band of the active receiver to the 136 kHz band. If already on that band, move to the next band stack entry. This command is a no-op if the frequency of the band falls outside the frequency range of the radio.		

Band 144	144	B
Change band of the active receiver to the 144 MHz (2m) band. If already on that band, move to the next band stack entry. This command is a no-op if the frequency of the band falls outside the frequency range of the radio.		

Band 15	15	B
Change band of the active receiver to the 15m band. If already on that band, move to the next band stack entry. This command is a no-op if the frequency of the band falls outside the frequency range of the radio.		

Band 160	160	B
Change band of the active receiver to the 160m band. If already on that band, move to the next band stack entry. This command is a no-op if the frequency of the band falls outside the frequency range of the radio.		

Band 17	17	B
Change band of the active receiver to the 15m band. If already on that band, move to the next band stack entry. This command is a no-op if the frequency of the band falls outside the frequency range of the radio.		

Band 20	20	B
Change band of the active receiver to the 15m band. If already on that band, move to the next band stack entry. This command is a no-op if the frequency of the band falls outside the frequency range of the radio.		

Band 220	220	B
Change band of the active receiver to the 220 MHz (1.25 m) band. If already on that band, move to the next band stack entry. This command is a no-op if the frequency of the band falls outside the frequency range of the radio.		

Band 2300	2300	B
Change band of the active receiver to the 2300 MHz (13 cm) band. If already on that band, move to the next band stack entry. This command is a no-op if the frequency of the band falls outside the frequency range of the radio.		

Band 30	30	B
Change band of the active receiver to the 30m band. If already on that band, move to the next band stack entry. This command is a no-op if the frequency of the band falls outside the frequency range of the radio.		

Band 3400	3400	B
Change band of the active receiver to the 3400 MHz (9 cm) band. If already on that band, move to the next band stack entry. This command is a no-op if the frequency of the band falls outside the frequency range of the radio.		

Band 40	40	B
Change band of the active receiver to the 40m band. If already on that band, move to the next band stack entry. This command is a no-op if the frequency of the band falls outside the frequency range of the radio.		

Band 430	430	B
Change band of the active receiver to the 430 MHz (70 cm) band. If already on that band, move to the next band stack entry. This command is a no-op if the frequency of the band falls outside the frequency range of the radio.		

Band 6	6	B
Change band of the active receiver to the 6m band. If already on that band, move to the next band stack entry. This command is a no-op if the frequency of the band falls outside the frequency range of the radio.		

Band 60	60	B
Change band of the active receiver to the 60m band. If already on that band, move to the next band stack entry. This command is a no-op if the frequency of the band falls outside the frequency range of the radio.		

Band 70	70	B
Change band of the active receiver to the 70 MHz (4m) band. If already on that band, move to the next band stack entry. This command is a no-op if the frequency of the band falls outside the frequency range of the radio.		

Band 80	80	B
Change band of the active receiver to the 80m band. If already on that band, move to the next band stack entry. This command is a no-op if the frequency of the band falls outside the frequency range of the radio.		

Band 902	902	B
Change band of the active receiver to the 902 MHz (33 cm) band. If already on that band, move to the next band stack entry. This command is a no-op if the frequency of the band falls outside the frequency range of the radio.		

Band AIR	AIR	B
Change band of the active receiver to the 108 MHz band, used for aircraft communication. If already on that band, move to the next band stack entry. This command is a no-op if the frequency of the band falls outside the frequency range of the radio.		

Band GEN	GEN	B
Change band of the active receiver to the current band stack entry of the "general" band. If already on that band, move to the next band stack entry. This command is a no-op if the frequency of the band falls outside the frequency range of the radio.		

Band -	BND-	B
Change band of the active receiver to the next lower band in the list of bands. If already at the lowest band, switch to the highest band (including transverter bands which have been defined) whose frequency is with the radio's frequency range.		

Band +	BND+	B
Change band of the active receiver to the next higher band in the list of bands (including transverter bands that have been defined). If already at the highest band, switch to the lowest band whose frequency is with the radio's frequency range.		

Band WWV	WWV	B
Change band of the active receiver to the current band stack entry of the WWV band. If already on that band, move to the next band stack entry. This command is a no-op if the frequency of the band falls outside the frequency range of the radio.		

BndStack -	BSTK-	B
Cycle backward through the band stack entries of the active receiver.		

BndStack +	BSTK+	B
Cycle forward through the band stack entries of the active receiver.		

Band Menu	BAND	B
Open the `BAND` menu.		

BndStack MENU	BSTK	B
Open the `BANDSTACK` menu.		

Capture	CAPTUR	B
Capture/Replay button: The procedure for audio recording and replay is described in detail in chapter 16.		

Cmpr On/Off	COMP	B
Toggle the state (on/off) of the compressor used in the TX audio input. Note that the CESSB overshoot correction is automatically engaged when using compression.		

Cmpr Level	COMPVAL	PSE
Change the value of the compressor (0-20 dB) used in the TX audio input. The compressor is automatically switched on (off) if the "new" value of the compressor is larger then (equal to) zero.		

CTUN	CTUN	B
Toggle the state (on/off) of the CTUN state of the active receiver. CTUN stands for "click to tune". In CTUN mode, you can move the RX frequency over the whole spectrum scope, whose centre then remains at a fixed frequency.		

CW Audio Peak Fltr	CW-APF	B
Toggle (on/off) the CW audio peak filter for the active receiver. Note that the width of this filter (default: 75 Hz) can only be modified through the CW menu.		

CW Frequency	CWFREQ	PE
Change the CW side tone frequency in the range 300-1000 Hz. This also changes the BFO frequency upon receive.		

CW Key (keyer)	CWKy	B
Straight key key-down or key-up event. Usually assigned to a GPIO line of MIDI controller to which a straight key or an external keyer is attached. Note that this command does not automatically switch to TX, so it must be used together with either manual RX/TX switching, or with the "`PTT (CW Keyer)`" command.		

PTT (keyer)	CWKyPTT	B
This very similar to the `PTT` command (see below) with the exception that CW handling in the radio is temporarily disabled (thus, CW handling in piHPSDR is enabled). This allows to have, e.g. a paddle attached to the radio while a contest logging program ,,talks" to piHPSDR.		

Speed (Keyer)	CWKySpd	P
This is very similar to the `CW Speed` command, except that the MIDI values 1-127 are directly mapped onto the speed (only values from 1-60 are accepted). This command is not meant for physical knobs, but to be sent by CW keyers.		

CW Left	CWL	B
This command indicates the closure/opening of the left paddle of a CW key. It is usually assigned to a GPIO line or a MIDI controller to which a Morse paddle is attached, and works with the iambic keyer that is built into piHPSDR. This keyer is only active if CW is *not* handled in the radio (see CW menu).		

CW Right	CWR	B
This command indicates the closure/opening of the right paddle of a CW key. It is usually assigned to a GPIO line or a MIDI controller to which a Morse paddle is attached, and works with the iambic keyer that is built into piHPSDR. This keyer is only active if CW is *not* handled in the radio (see CW menu).		

CW Speed	CWSPD	PSE
Change the CW side tone frequency in the range 1-60 wpm. This affect the built-in iambic keyer or the keyer inside the radio, depending on whether CW is handled in the radio or not (see CW menu).		

CW Txt1	CWTxt1	B
Transmit CW text number 1. This only works if in CW mode and if the radio can transmit. See the CW menu (chapter 9.5.2) on how to change CW texts. As for CAT generated CW, aborting a running transmission of a CW text can be performed by hitting the morse key or paddle. If no such key is attached to the radio, switching to a non-CW mode also drains the buffer containing the text to be transmitted.		

CW Txt2...5	CWTxt2...5	B
This is the same as CW Txt1, except that CW text number 2 through 5 is referred to.		

DIV On/Off	DIVT	B
Toggles (enabled/disabled) DIVERSITY reception.		

DIV Gain	DIVG	E
Adjust DIVERSITY gain. One tick of the encoder increments of decrements the gain by an amount of 0.5		

DIV Gain Coarse	DIVGC	E
Adjust DIVERSITY gain (coarse adjustment). One tick of the encoder increments of decrements the gain by an amount of 2.5		

DIV Gain Fine	DIVGF	E
Adjust DIVERSITY gain (fine adjustment). One tick of the encoder increments of decrements the gain by an amount of 0.1. Since adjusting the DIVERSITY gain (or phase) is sometimes difficult, assigning one encoder to a coarse and another encoder to a fine adjustment may help in locating the ,,sweet spot".		

DIV Phase	DIVP	E
Adjust DIVERSITY phase (fine adjustment). One tick of the encoder increments of decrements the gain by an amount of 0.5		

DIV Phase Coarse	DIVPC	E
Adjust DIVERSITY gain (coarse adjustment). One tick of the encoder increments of decrements the gain by an amount of 2.5		

DIV Phase Fine	DIVPF	E
Adjust DIVERSITY gain (coarse adjustment). One tick of the encoder increments of decrements the gain by an amount of 20.1		

DIV Menu	DIV	B
Open the DIVERSITY menu.		

Duplex	DUP	B
Toggle (on/off) DUPLEX status. IN the DUPLEX mode, the receivers continue to work during TX, and the RX panels are not removed during TX. Instead, a separate TX window opens during transmitting. Generally, DUPLEX only make sense when using different and well decoupled RX and TX antennas.		

Filter -	FL-	B
Cycle forward (!) through the list of filters for the current mode of the active receiver. Normally, this means switching to a narrower filter (hence the name `FILTER -`). When reaching the last filter in the list, further cycling switches to the first (widest) filter.		

Filter +	FL+	B
Cycle backward (!) through the list of filters for the current mode of the active receiver. Normally, this means switching to a wider filter (hence the name FILTER +). When reaching the first filter in the list, further cycling switches to the last filter which is the variable Var2 filter.		

Filter Cut Default	FCUTDEF	B
This commands restores the filter edges of the active receiver to the nominal values of the currently selected filter. It thus reverts any changes made with the Filter Cut Low/High, IF Shift/Width, etc. commands.		

Filter Cut Low	FCUTL	E
Adjust the low-cut of the filter of the active receiver. Note that for the single side band modes LSB, CWL, DIGL, the filter edge affecting the low *audio* frequencies is changed. This command is meant for temporary QRM fighting, and the filter edges will be restored to their nominal values upon the next filter, mode, band, or band stack change as well when recalling a memory slot. For creating a permanent filter with user-defined filter edges, there are two filters (Var1, Var2) for each mode which can be changed permanently in the Filter menu.		

Filter Cut High	FCUTH	E
Adjust the high-cut of the filter of the active receiver. Note that for the single side band modes LSB, CWL, DIGL, the filter edge affecting the high *audio* frequencies is changed. This change is temporary, see the remark in the Filter Cut Low command.		

Filter Menu	FILT	B
This opens the Filter menu.		

Freq Menu	FREQ	B
This opens the FREQ menu.		

Function	FUNC	B
Cycle through the six toolbar sets. For the piHPSDR GPIO Controller1, where the eight switches follow the toolbar buttons, this also affects the function of the switches. Note that this command is *always* connected with the right-most toolbar button.		

FuncRev	FUNC-	B
Cycle backwards through the six toolbar sets. For the piHPSDR GPIO Controller1, where the eight switches follow the toolbar buttons, this also affects the function of the switches. When using a mouse, this command can be invoked by a secondary mouse click on the rightmost toolbar button.		

Iconify	ICON	b
Minimise (,,Iconify") the piHPSDR window.		

IF Shift	IFSHFT	E
This command shifts the filter edges of the active receiver, that is, it affects the low and high cut in the same way. This change is temporary, see the remark in the `Filter Cut Low` command.		

IF Shift RX1	IFSHFT1	E
Shift the filter edges of the receiver RX1 (low and high cut move the same way). This change is temporary, see the remark in the `Filter Cut Low` command.		

IF Shift RX2	IFSHFT2	E
Shift the filter edges of the receiver RX2 (low and high cut move the same way). This change is temporary, see the remark in the `Filter Cut Low` command.		

IF Width	IFWIDTH	E
Change the width of the filter of the active receiver. This change is temporary, see the remark in the `Filter Cut Low` command.		

IF Width RX1	IFWIDTH1	E
Change the width of the filter of the receiver RX1. This change is temporary, see the remark in the Filter Cut Low command.		

IF Width RX2	IFWIDTH2	E
Change the width of the filter of the receiver RX2. This change is temporary, see the remark in the Filter Cut Low command.		

Linein Gain	LIGAIN	PSE
Change the line-in gain of the radio. If the radio does not have a line-in input, this control has no effect.		

Lock	LOCK	B
Lock the VFOs. A locked VFO will not accept VFO frequency steps in either direction, and cannot be moved by dragging with the mouse. Band changes etc. are still possible, though. The command is intended to guard against accidentally moving the VFO dial.		

Main Menu	MAIN	B
Open the main menu.		

Memory Menu	MEM	B
Open the MEM (Memory) menu (see Chapter 7.5).		

Mic Gain	MICGAIN	PSE
Change the mic gain (from -12 to 50 dB). The amplification of the microphone audio data is done in software, and applies to the TX audio input samples wherever they come from. (See the discussion of local microphones in the TX menu.		

Mode -	MD-	B
Cycle backwards through the list of modes for the active receiver. When the first mode (LSB) has been reached, jump to the last one (DRM). Note that when changing the mode, the current filter, noise reduction, equaliser, VFO step size, and TX compressor settings are stored for the old mode, and the settings last used with the new mode are restored. This allows to quickly switch between SSB and CW, or between SSB and digi modes, without re-adjusting these settings.		

Mode +	MD+	B
Cycle forward through the list of modes for the active receiver. When the last mode (DRM) has been reached, jump to the first one (LSB). Note that when changing the mode, the current filter, noise reduction, equaliser, VFO step size, and TX compressor settings are stored for the old mode, and the settings last used with the new mode are restored. This allows to quickly switch between SSB and CW, or between SSB and digi modes, without re-adjusting these settings.		

Mode Menu	MODE	B
Open the Mode menu.		

MOX	MOX	B
Toggle between TX and RX. Unlike the PTT command, which puts the radio into TX when pressed and into RX when released, this button toggles the PTT state when pressed.		

Multi	MULTI	E
This is the multi-function encoder. It executes the encoder command it is currently assigned to.		

Multi Action Select	MULTISEL	E
With this encoder, one cycles through the list of commands assigned to the multi-function encoder. The currently active command is displayed in the VFO bar. For examples, if the AFGAIN command (change audio volume of the current receiver) is assigned to the multi-function encoder, it will change the AF gain. This command it used if one used two encoders to activate the multi-function encoder feature.		

Multi Toggle	MULTIBTN	B
This button toggles the ,,multi-encoder select" state. If this state is active, one can change the command assigned to the multi-function encoder using that encoder. This function is used if one uses one encoder and one push-button to activate the multi-function encoder feature.		

Mute	MUTE	B
Toggles the ,,mute" state of the active receiver. If a receiver is muted, its audio is replaced by silence. This applies both to the audio data stream to the radio and local audio (if enabled).		

Mute RX1	MUTE1	B
Toggles the ,,mute" state of RX1. If a receiver is muted, its audio is replaced by silence. This applies both to the audio data stream to the radio and local audio (if enabled).		

Mute RX2	MUTE2	B
Toggles the ,,mute" state of RX2. This operation is a no-op if only one receiver is running. If a receiver is muted, its audio is replaced by silence. This applies both to the audio data stream to the radio and local audio (if enabled).		

NB	NB	B
Cycles through the noise blanker states (NB off/NB1/NB2).		

Notch1 Center	Notch1C	E
Change the centre frequency of the first notch for the active receiver (see the Filter menu). The same commands exists for the other two notches (Notch2 Center/Notch2C and Notch3 Center/Notch3C). The change can only be seen on the RX pan-adapter if the notch is active.		

Notch1 Width	Notch1W	E
Change the width of the first notch for the active receiver (see the Filter menu). The same commands exists for the other two notches (Notch2 Width/Notch2W and Notch3 Width/Notch3W). The change can only be seen on the RX pan-adapter if the notch is active.		

Notch1 Enable	Notch1E	B
Toggles (on/off) the first notch for the active receiver. The same commands exists for the other two notches (Notch2 Enable/Notch2E and Notch3 Enable/Notch3E).		

NR	NR	B
Cycles through the noise reduction states (NR off/NR1/NR2).		

Noise Menu	NOISE	B
Opens the NOISE menu.		

NumPad 0	0	B
Used for direct frequency entry. This is the same as hitting the corresponding button ,,0" in the VFO (VFO) menu.		

NumPad 1...NumPad 9	1...9	B
The same as NumPad 0, except that digits (button) ,,1" through ,,9" are referred to.		

NumPad BS	BS	B
Used for direct frequency entry (BS = back-step). This is the same as hitting the corresponding button in the VFO menu. It cancels the last-entered digit.		

NumPad CL	CL	B
Used for direct frequency entry (CL = clear). This is the same as hitting the corresponding button in the VFO menu. It cancels all entered digits so far.		

NumPad Dec	DEC	B
Used for direct frequency entry (DEC = decimal point). This is the same as hitting the corresponding button in the VFO menu.		

NumPad kHz	KHZ	B
Used for direct frequency entry. This is the same as hitting the corresponding button in the VFO menu. The VFO frequency is changed to the value entered so far, multiplied with 1000. For example, to go to 7.040 MHz, one can enter the sequence ,,7", ,,0", ,,4", ,,0", ,,KHZ".		

NumPad MHz	MHZ	B
Used for direct frequency entry. This is the same as hitting the corresponding button in the VFO menu. The VFO frequency is changed to the value entered so far, multiplied with 1,000,000. For example, to go to 7.040 MHz, one can enter the sequence ,,7", ,,DEC", ,,0", ,,4", ,,MHZ".		

NumPad Enter	EN	B
Used for direct frequency entry. This is the same as hitting the corresponding button in the VFO menu. The VFO frequency is changed to the value entered so far. For example, to go to 7.040 MHz, one can enter the sequence ,,7", ,,0", ,,4", ,,0", ,,0", ,,0", ,,0". This is rarely used but offers Hz-resolution for the direct frequency entry.		

PanZoom	PAN	PSE
Change the Pan value. Note the PAN value has no effect if Zoom equals unity. The PAN value goes from 0 (show leftmost part) and 100 (show rightmost part of the spectrum).		

Pan-	PAN-	B
Decrease the PAN value.		

Pan+	PAN+	B
Increase the PAN value.		

Panadapter High	PANH	PE
Change the dBm value (from -60 to +20) at the top of the spectrum scope of the active receiver. Values outside this range can be set in the `DISPLAY` menu.		

Panadapter Low	PANL	PSE
Change the dBm value (from -160 to -60) at the bottom of the spectrum scope of the active receiver. Values outside this range can be set in the `DISPLAY` menu.		

Panadapter Step	PANS	PE
Change the step size (from 5 to 30) of the pan-adapter of the active receiver. This is the spacing of the thin horizontal lines in the spectrum scope.		

Preamp On/Off	PRE	B
Toggle the preamp of the active receiver. Although the preamp switching is part of the HPSDR protocol, this has no effect in current radio models since the preamp is hard-wired ,,on”.		

PS On/Off	PST	B
Toggle (on/off) adaptive pre-distortion (PureSignal).		

PS Menu	PS	B
Open the PS (PureSignal) menu.		

PTT	PTT	B
Put the radio into TX mode when the button is pressed, and go back to RX when the button is released. This is one of the few commands where a button release event is significant. When attaching, say, the PTT contact of a microphone to a GPIO line for this purpose, take care of proper denouncing, since piHPSDR is not good at denouncing switches where both the press and release events are significant.		

Radio Menu	RADIO	B
Open the Radio menu.		

Rcl 0	RCL0	B
Recall (restore) data from the memory slot 0 (see the Memory menu, Chapter 7.5).		

Rcl 1...Rcl 9	RCL1...RCL9	B
The same as Rcl 0, except that memory slots 1 through 9 are referred to.		

Replay	REPLAY	B
If there is captured RX audio (see the Capture command) and while RXing, the captured data is replayed in the RX audio of the active receiver. If transmitting, Replay and Capture are the same. The procedure for audio recording and replay is described in detail in chapter 16.		

RF Gain	RFGAIN	PSE
Set the gain of the RF front end of the active receiver. Only effective for radios that have such a gain control. Most HPSDR radios do not have RF gain, they have a step attenuator in the RF front end instead. Small SDR radios using the AD9866 chip (HermesLite, RadioBerry) and radios connected via the SoapySDR library usually do have an RF gain control. If ADC overload occurs, one has to decrease the RF gain. If the attenuation value is changed for RX1, this is stored "with the band" in a database. Changing the band for RX1 will set the attenuation from that database.		

RF Gain RX1	RFGAIN1	PE
Set the gain of the RF front end of RX1. Only effective for radios that have such a gain control. Most HPSDR radios do not have RF gain, they have a step attenuator in the RF front end instead. Small SDR radios using the AD9866 chip (HermesLite, RadioBerry) and radios connected via the SoapySDR library usually do have an RF gain control. If ADC overload occurs, one has to decrease the RF gain. If the attenuation value is changed for RX1, this is stored "with the band" in a database. Changing the band for RX1 will set the attenuation from that database.		

RF Gain RX2	RFGAIN2	PE
Set the gain of the RF front end of RX2. Only effective for radios that have such a gain control. Most HPSDR radios do not have RF gain, they have a step attenuator in the RF front end instead. Small SDR radios using the AD9866 chip (HermesLite, RadioBerry) and radios connected via the SoapySDR library usually do have an RF gain control. If ADC overload occurs, one has to decrease the RF gain.		

RIT	RIT	E
Change the RIT value of the active receiver in the range -9999 to 9999 Hz. Each tick of the encoder changes the value by the current RIT step size. If a zero value results, RIT is automatically disabled, if a non-zero value is set, RIT is enabled.		

RIT Clear	RITCL	B
Set the RIT value of the active receiver to zero. As a side effect, RIT is disabled for the active receiver		

RIT On/Off	RITT	B
Toggle RIT (enabled/disabled) for the active receiver. Note the RIT value is not changed, so you can temporarily disable RIT, and then enable it with the same offset (RIT value) used before.		

RIT -	RIT-	B
Decrement the RIT value of the active receiver by the RIT step size, in the range -9999 to 9999 Hz. If a value of zero results, RIT is automatically disabled, and if a nonzero value is reached, RIT is automatically enabled. Note that this command belongs to the few ones for which a button release event has an effect. If you press and hold RIT- (either on the toolbar, or on a GPIO or MIDI console), there is an auto-repeat such that the command will be repeated every 250 msec until the RIT- button is released.		

RIT +	RIT+	B
Increment the RIT value of the active receiver by the RIT step size, in the range -9999 to 9999 Hz. If a value of zero results, RIT is automatically disabled, and if a nonzero value is reached, RIT is automatically enabled. Note that this command belongs to the few ones for which a button release event has an effect. If you press and hold RIT+ (either on the toolbar, or on a GPIO or MIDI console), there is an auto-repeat such that the command will be repeated every 250 msec until the RIT+ button is released.		

RIT RX1	RIT1	E
Change the RIT value of VFO-A in the range -9999 to 9999 Hz. If a zero value is set, RIT is automatically disabled, if a non-zero value is set, RIT is enabled.		

RIT RX2	RIT2	E
Change the RIT value of VFO-B in the range -9999 to 9999 Hz. If a zero value is set, RIT is automatically disabled, if a non-zero value is set, RIT is enabled.		

RIT Step	RITST	B
Cycle through the possible values (1 Hz, 10 Hz, 100 Hz) of the RIT step.		

RIT/XIT	RITXIT	E
This dual-purpose encoder changes the XIT value if (and only if) XIT is enabled and RIT is disabled for the active receiver, otherwise it changes the RIT value of the active receiver.		

RIT/XIT Cycle	RITXTCYC	B
Cycle between RIT on, XIT on, and both off. This is meant to be used with the RIT/XIT dual-purpose encoder.		

RIT/XIT Clear	RITXTCLR	B
Clear both the XIT value, and the RIT value of the active receiver. As a side effect, both RIT and XIT will be disabled.		

RSAT	RSAT	B
If the SAT mode is either Off or SAT, change it to RSAT. If the SAT mode is RSAT, change it to Off. In RSAT mode all VFO frequency *changes* applied to one of the two VFOs will be applied to the other VFO with the sign reversed.		

RX Menu	RX	B
Opens the `RX` menu for the active receiver.		

RX1	RX1	B
Make the first receiver the active one, if piHPSDR is running two receivers.		

RX2	RX2	B
Make the second receiver the active one, if piHPSDR is running two receivers.		

SAT	SAT	B
If the SAT mode is either Off or RSAT, change it to SAT. If the SAT mode is SAT, change it to Off. In SAT mode all VFO frequency *changes* applied to one of the two VFOs will be applied to the other VFO as well.		

Shutdown OS	SDWN	B
Terminate piHPSDR and shut down operating system. Note that piHPSDR will terminate in any case, but shutting down the operating system may fail, e.g. due to missing administrator privileges. This command is primarily meant for shutting down a radio where a radio board and a small single-board computer running piHPSDR are built into a single case.		

SNB	SNB	B
Toggle (enable/disable) the spectral noise blanker for the active receiver.		

Split	SPLIT	B
Toggle (on/off) the split status of the radio. Note that a frequent beginner error is to have, say, SSB mode in VFO-A and CW mode in VFO-B. In this case, nothing is transmitted when doing split and using the microphone. Normally one starts with the `A>B` command (that is, copy VFO-A to VFO-B), and then adjusts the transmit frequency.		

Squelch	SQUELCH	PSE
Change the squelch threshold value of the active receiver. Squelch is automatically enabled (disabled) if the resulting value is non-zero (zero).		

Squelch RX1	SQUELCH1	PE
Change the squelch threshold value of RX1. Squelch is automatically enabled (disabled) if the resulting value is non-zero (zero).		

Squelch RX2	SQUELCH2	PE
Change the squelch threshold value of RX2. Squelch is automatically enabled (disabled) if the resulting value is non-zero (zero).		

Swap RX	SWAPRX	B
Make the inactive receiver the active one. This command is only effective if piHPSDR is running two receivers.		

ToolBar1	TBAR1	B
Pressing/Releasing this button is equivalent to pressing the leftmost (first) of the eight toolbar buttons on the screen.		

ToolBar2...7	TBAR2...7	B
This is the same as `ToolBar1`, except that one of the buttons following in the row is referred to. Note that there is no command referring to the eighth (rightmost) toolbar button, since this is hard-wired to the `Function` command.		

Tune	TUNE	B
Toggle (on/off) TUNE. If `TuneBits` are checked in the `OC` menu, the corresponding open collector outputs will become active (low) during TUNE-ing. This can then be used to start an external automatic tuner.		

Tune Drv	TUNDRV	E
Change the drive level (0-100) used for TUNE-ing. This is equivalent to changing the "Tune drive level" spin button in the `TX` menu **and** to check the "Tune use drive" box.		

Tune Full	TUNF	B
This command is largely equivalent to the Tune command, except that open collector bits which became active at the beginning of the TUNE cycle will become inactive again after the "full tune delay" (to be specified in the OC menu). Note the TUNE-ing itself will continue after that.		

Tune Mem	TUNM	B
This command is largely equivalent to the Tune command, except that open collector bits which became active at the beginning of the TUNE cycle will become inactive again after the "memory tune" delay (to be specified in the OC menu). Note the TUNE-ing itself will continue after that.		

TX Drive	TXDRV	PSE
Set the TX drive (RF output) level (0-100).		

TX Menu	TX	B
Opens the TX menu.		

Two-Tone	2TONE	B
Toggle (on/off) the two-tone state of the transmitter. If the two-tone state is engaged, the radio will go TX and emit a two-tone signal. If PURESIGNAL is enabled with auto calibration, then the PURESIGNAL engine will be restarted and the attenuation of the feedback signal re-adjusted while the two-tone signal is being sent. In the lower side band modes (LSB, DIGL, CWL), an RF two-tone signal with frequencies 700 and 1900 Hz *below* the dial frequency are transmitted, in all other modes the two RF frequencies are 700 and 1900 Hz *above* the dial frequency.		

VFO	VFO	E
This is the VFO frequency control of the active receiver.		

VFO A	VFOA	E
This is the VFO frequency control of VFO-A.		

VFO B	VFOB	E
This is the VFO frequency control of VFO-B.		

VFO Step -	STEP-	E
Cycle downwards through the VFO step sizes of the VFO controlling the active receiver. If the step size was already the smallest one, jump to the largest.		

VFO Step +	STEP+	E
Cycle upwards through the VFO step sizes of the VFO controlling the active receiver. If the step size was already the largest one, jump to the smallest.		

VOX On/Off	VOX	B
Toggle (on/off) VOX status. If VOX is enabled, you can automatically key the transmitter by talking into the microphone, without the need to press a PTT button. See the `VOX` menu.		

VOX Level	VOXLEV	PSE
Change the VOX level threshold. If you operate VOX, and the radio does not go TX while talking into the microphone, decrease the VOX threshold. If the radio goes TX simply because the neighbour's hound starts barking, increase the VOX threshold.		

Wfall High	WFALLH	E
Change the "high" level (-100 dBm ... 0 dBm) of the waterfalls. Signal levels between low and high are colour coded from black to yellow, while signals above "high" are yellow and signals below "low" are black. This value has no effect if the automatic waterfall colouring is chosen ("waterfall automatic"), which is usually preferable.		

Wfall Low	WFALLL	E
Change the "low" level (-150 dBm ... -50 dBm) of the waterfalls. Signal levels between low and high are colour coded from black to yellow, while signals above "high" are yellow and signals below "low" are black. This value has no effect if the automatic waterfall colouring is chosen ("waterfall automatic"), which is usually preferable.		

XIT	XIT	E
Change the XIT value of the transceiver in the range -9999 to 9999 Hz. If a zero value is set, XIT is automatically disabled, if a non-zero value is set, XIT is enabled. Each tick of the encoder changes the value by the current RIT step size.		

XIT Clear	XITCL	B
Set the XIT value of the transmitter to zero. As a side effect, XIT is disabled.		

XIT On/Off	XITT	B
Toggle XIT (enabled/disabled) for the transceiver. Note the XIT value is not changed, so you can temporarily disable XIT, and then enable it with the same offset (XIT value) used before.		

XIT -	XIT-	B
Decrement the XIT value of the transmitter by 10 times the RIT step size, in the range -9999 to 9999 Hz. If a value of zero is reached, XIT is automatically disabled, and if a nonzero value is reached, XIT is automatically enabled. Note that this command belongs to the few ones for which a button release event has an effect. If you press and hold XIT- (either on the toolbar, or on a GPIO or MIDI console), there is an auto-repeat such that the command will be repeated every 250 msec until the XIT- button is released.		

XIT +	XIT+	B
Increment the XIT value of the transmitter by 10 times the RIT step size, in the range -9999 to 9999 Hz. If a value of zero is reached, XIT is automatically disabled, and if a nonzero value is reached, XIT is automatically enabled. Note that this command belongs to the few ones for which a button release event has an effect. If you press and hold XIT+ (either on the toolbar, or on a GPIO or MIDI console), there is an auto-repeat such that the command will be repeated every 250 msec until the XIT+ button is released.		

Zoom	ZOOM	PSE
Change the ZOOM value (1...32) of the active receiver.		

Zoom -	ZOOM-	B
Decrease the ZOOM value of the active receiver by one. If the ZOOM value was already 1, this is a no-op.		

Zoom +	ZOOM+	B
Increase the ZOOM value of the active receiver by one. If the ZOOM value was already 32, this is a no-op.		

Appendix B

The MULTI encoder

When using a controller (GPIO, MIDI or G2 front panel), one (and only one) of the encoders can be chosen as *the* multi-function encoder by assigning the `Multi` command (For the G2 Mk2, the assignment is fixed, as for all other encoders an push-buttons there). This means that an command (say, changing the AF volume, the TX drive or the step attenuator of the RF front-end) can be dynamically assigned to that encoder, making it very facile to use a single ,,knob" for various purposes. One has to sacrifice either another encoder or a push-button to have an easy-to-use handle to change the command currently assigned with the multi-function encoder. Two such setups are possible:

In the first setup, one uses two encoders to implement the multi-function feature. Besides the multi-function encoder, one uses another encoder called the multi-selection encoder and assigns the `Multi Select` command to this encoder. With the latter one, one can (alphabetically) cycle through the list of commands assigned to the multi-function encoder. Currently, one can choose between 28 such commands, they are listed here by the long name (see Appendix A):

AF Gain	AGC Gain	Atten	Cmpr Level
CW Frequency	CW Speed	DIV Gain	DIV Phase
Filter Cut Low	Filter Cut High	IF Shift	IF Width
Linein Gain	Mic Gain	PanZoom	Panadapter High
Panadapter Low	Panadapter Step	RF Gain	RIT
Squelch	Tune Drv	TX Drive	VOX Level
WFall High	WFall Low	XIT	Zoom

For those controllers with double encoders on a single shaft, one can preferably assign the encoder operated with the outer ring to `Multi Select` and the encoder operated with the inner knob to `Multi`. Of course, one can equally well use two different encoders for this purpose.

In the second setup, one only uses one encoder and a push button to implement the multi-function feature. The `Multi Toggle` command is assigned to the push button which is used to toggle between the ,,select" and ,,command" states. The normal state is the ,,command" state, where turning the encoder executes the function it is assigned to. When in ,,select" state, one can assign a new command to the multi-function encoder exactly as in the first setup, except that one now turns the multi-function encoder itself. Toggling between the two states with the `Multi Toggle` button thus eliminates the need for a second encoder to implement the multi-function feature.

The current `Multi` command is shown in the VFO bar at the bottom right, that is, below the VFO B frequency, using a short and (hopefully) descriptive text, since space there is scarce for the smaller VFO bars. The text is printed in yellow in the normal (,,command") state and turns red in the ,,select" state. This text is not shown until the first multi function command (`MULTI`, `Multi Select`, or `Multi Toggle`) is executed such that it is not shown as long as no multi function encoder is defined or used. The `Multi` command is not stored in the preferences file, the default command at program startup always is the first in the list, namely `AF gain`.

Appendix C

piHPSDR keyboard bindings

There are a lot of keyboard bindings effective if you run piHPSDR. Most of them are standard key bindings of the GTK user interface. For example, when using a normal slider, you can use the up-arrow and down-arrow keys to fine adjust the value. In this section we will only list the keyboard binding that are captured and processed by piHPSDR.

C.1 Accessibility for the Blind

piHPSDR has built-in support for making operating easier for those with very limited eye-sight. The idea is that when a radio is running, pressing one of the function keys generates a message that is read aloud and that informs about frequency, mode, etc. While the radio is still in the discovery process, information about the discovery status is reported by reading this aloud.

In all cases (MacOS and Linux) a broadcast UDP packet containing the text to be spoken is ent out to port number 19080. A program running on the same computer, or even on another computer on the same network, can then receive the packet and take care of the reading. A sample program that does so is included in the piHPSDR source code tree (`udplistener.c`) for demonstration purposes, that is instrumented to use the `eSpeak` program. It is the task of the user to make good use of the data contained in the UDP packet.

Additionally, the MacOS builtin speech synthesiser is used so there should be no need for the `udplistener.c` program there. piHPSDR specifies english (en-GB) language for the text to be spoken.

The whole text-to-speech feature, if it is not needed, can be suppressed by compiling piHPSDR *without* the `TTS` compile-time option (see Sec. G). Note that even without `TTS`, the function keys F3 (start first radio) and F4 (restart discovery process) still work on the discovery screen, but without producing any speech.

Functions currently implement during startup/discovery are

- `F1` Hitting the F1 function key reports the status of the discovery process, whether it has not yet started, or whether it is running, or whether it is completed. In the latter case the number of radios discovered is reported as well.
- `F2` Hitting the F2 function key reports information (which radio model, and which protocol it is running) for the first discovered radio, or reports that this could not be done since no radio has been discovered.
- `F3` Hitting the F3 function key starts the discovered radio that is first discovered radio, or reports that this could not be done since no radio has been discovered.
- `F4` Hitting the F4 function key restarts the discovery process.

Functions currently implemented for a radio that is already running are

- `F1` Hitting the F1 function key reads the frequency of the active receiver.
- `F2` Hitting the F2 function key reads the mode of the active receiver.
- `F3` Hitting the F3 function key reads the filter width of the active receiver.
- `F4` Hitting the F4 function key reads the S-meter value of the active receiver. Note there may be strong fluctuations in the reported value.
- `F5` Hitting the F5 function key reads the value of the TX drive slider.
- `F6` Hitting Hitting the F6 function key reads the RF gain/attenuation in the front end of the active receiver.

C.2 Other keyboard shortcuts

`space` Hitting the space bar will execute the `MOX` command, that is, a transition from RX to TX or from TX to RX. Use this as the last resort for TRX switching if your microphone does not have PTT.

`u` Hitting a lower-caps letter ,,u" moves the VFO frequency up by the current VFO step size.

`d` Hitting a lower-caps letter ,,d" moves the VFO frequency up by the current VFO step size.

`U` Hitting a upper-caps letter ,,U" moves the VFO frequency of the ,,other" VFO, that is, the VFO that is *not* controlling the active receiver, up by 10 times the VFO step size.

`D` Hitting a upper-caps letter ,,D" moves the VFO frequency of the ,,other" VFO, that is, the VFO that is *not* controlling the active receiver, down by 10 times the VFO step size.

The keys `U,D` are for stations chasing DX in split mode. One can tune to the DX station, then copy the VFO of the active receiver to the other one using `A>B` or `A<B`, then then one can move the TX frequency (in split mode) using `U,D` in large steps.

`m`, `M` opens the main menu.

`I` minimises (iconifies) the piHPSDR window.

`KP 0-9` Hitting a digit on the numerical keypad executes one of the `NumPad 0` to `NumPad 9` commands. This can be used for direct frequency entry via the keyboard.

`KP .` Hitting the decimal point on the keypad executes the `NumPad Dec` command which will enter a decimal point during direct frequency entry.

`KP -` Hitting the ,,minus" key on the numerical keypad executes the `NumPad BS` (back space) command.

`KP /` Hitting the ,,divide" key on the numerical keypad executes the `NumPad CL` command which will clear any frequency entered so far.

`KP *` Hitting the ,,multiply" key on the numerical keypad executes the `NumPad Hz` command which will transfer the frequency entered (in Hz) to the VFO of the active receiver.

`KP +` Hitting the ,,plus" key on the numerical keypad executes the `NumPad kHz` command which will transfer the frequency entered (in kHz) to the VFO of the active receiver.

`KP enter` Hitting the ,,enter" key on the numerical keypad executes the `NumPad MHz` command which will enter the frequency entered (in MHz) to the VFO of the active receiver.

Appendix D

piHPSDR CAT commands

The CAT model of piHPSDR largely follows that for other SDR programs. It is based upon the Kenwood TS-2000 CAT command set, which can easily be found on the internet (see the Appendix of the Kenwood TS-2000 instruction manual) So if you want to connect a logbook or contest logging program to piHPSDR, you will normally tell this program that it has to control a Kenwood TS-2000. Modern version of these programs are not restricted to serial lines and can use TCP as well.

Many digi mode programs (and perhaps others as well) use the `hamlib` library for radio control. The digi mode program communicates with `hamlib` using an abstract interface, and `hamlib` then communicates with the radio using CAT commands. piHPSDR is supported by `hamlib`, the radio model name is ,,`OPENHPSDR PiHPSDR`".

In the SDR community, there exist a heavily extended TS-2000 CAT command set known as the ,,PowerSDR CAT command set". In contrast to the two-letter commands characteristic for the original (TS-2000) CAT command set, the extended commands have four letters and begin with two 'Z'. A small subset of the extended commands has been implemented in piHPSDR. This set has later been extended somewhat to support the ANDROMEDA open-source radio controller developed by Laurence Barker G8NJJ, see

`https://github.com/laurencebarker/Andromeda_front_panel`

and the additional commands have been implemented in the CAT module of piHPSDR by Rick Koch N1GP (thanks Rick). Furthermore, if "Andromeda"

is selected in the CAT/TCI menu, piHPSDR will constantly *send* status information to the ANDROMEDA controller using. Status information is sent if something changes (active receiver, diversity status, PTT status, TUNE mode, PS status, CTUN mode, RIT and XIT status, and LOCK status), such that the ANDROMEDA controller can update the corresponding LEDs.

The ANDROMEDA protocol is also used for the communication with the panel of the Saturn G2 Mk2 radios.

D.1 Kenwood (two-letter) CAT commands supported by piHPSDR

AG	Sets/Reads audio volume (AF slider)
Set	AG$p_1p_2p_2p_2$;
Read	AGp_1;
Response	AG$p_1p_2p_2p_2$;
Notes	p_1=0 sets RX1 audio volume, p_1=1 sets RX2 audio volume. p_2 is 0...255 and mapped logarithmically to the volume -40...0 dB

AI	Sets/Reads auto reporting status
Set	AIp_1;
Read	AI;
Response	AIp_1;
Notes	p_1=0: auto-reporting disabled, p_1>0: enabled. Auto-reporting is affected for the client that sends this command. For p_1=1, only frequency changes are sent via FA/FB commands. For p_1>1, mode changes are also sent via MD commands.

BD	VFO-A Band down
Set	BD;
Notes	Wraps from the lowest to the highest band.

BU	VFO-A Band up
Set	BU;
Notes	Wraps from the highest to the lowest band.

CN	Sets/Reads the CTCSS frequency
Set	CNp_1p_1;
Read	CN;
Response	CNp_1p_1;
Notes	p_1 = 1...38. CTCSS frequencies in Hz are: 67.0 (p_1=1), 71.9 (p_1=2), 74.4 (p_1=3), 77.0 (p_1=4), 79.7 (p_1=5), 82.5 (p_1=6), 85.4 (p_1=7), 88.5 (p_1=8), 91.5 (p_1=9), 94.8 (p_1=10), 97.4 (p_1=11), 100.0 (p_1=12) 103.5 (p_1=13), 107.2 (p_1=14), 110.9 (p_1=15), 114.8 (p_1=16) 118.8 (p_1=17), 123.0 (p_1=18), 127.3 (p_1=19), 131.8 (p_1=20) 136.5 (p_1=21), 141.3 (p_1=22), 146.2 (p_1=23), 151.4 (p_1=24) 156.7 (p_1=25), 162.2 (p_1=26), 167.9 (p_1=27), 173.8 (p_1=28) 179.9 (p_1=29), 186.2 (p_1=30), 192.8 (p_1=31), 203.5 (p_1=32) 210.7 (p_1=33), 218.1 (p_1=34), 225.7 (p_1=35), 233.6 (p_1=36) 241.8 (p_1=37), 250.3 (p_1=38).

CT	Enable/Disable CTCSS
Set	CTp_1;
Read	CT;
Response	CTp_1;
Notes	p_1 = 0: CTCSS off, p_1=1: on

DN	VFO-A down one step
Set	DN;
Notes	Parameters may be given, but are ignored.

FA	Set/Read VFO-A frequency
Set	FA$p_1p_1p_1p_1p_1p_1p_1p_1p_1p_1p_1$;
Read	FA;
Response	FA$p_1p_1p_1p_1p_1p_1p_1p_1p_1p_1p_1$;
Notes	p_1 in Hz, left-padded with zeroes

FB	Set/Read VFO-B frequency
Set	FB$p_1p_1p_1p_1p_1p_1p_1p_1p_1p_1p_1$;
Read	FB;
Response	FB$p_1p_1p_1p_1p_1p_1p_1p_1p_1p_1p_1$;
Notes	p_1 in Hz, left-padded with zeroes

FR	Set/Read active receiver
Set	FRp_1;
Read	FR;
Response	FRp_1;
Notes	$p_1 = 0$ (RX1) or 1 (RX2)

FT	Set/Read Split status
Set	FTp_1;
Read	FT;
Response	FTp_1;
Notes	p_1=0: TX VFO is the VFO controlling the active receiver, p_1=1: the other VFO.

FW	Set/Read VFO-A filter width (CW, AM, FM)
Set	FW$p_1p_1p_1p_1$;
Read	FW;
Response	FW$p_1p_1p_1p_1$;
Notes	When setting, this switches to the Var1 filter and sets its width to p_1. Only valid for CW, FM, AM. Use SH/SL for LSB, USB, DIGL, DIGU. For AM, 8kHz filter width (p_1=0) or 16 kHz ($p_1 \neq 0$) For FM, 2.5kHz deviation (p_1=0) or 5 kHz ($p_1 \neq 0$)

GT	Set/Read RX1 AGC
Set	GT$p_1p_1p_1$;
Read	GT;
Response	GT$p_1p_1p_1$;
Notes	p_1=0: AGC OFF, p_1=5: LONG, p_1=10: SLOW, p_1=15: MEDIUM, p_1=20: FAST.

ID	Get radio model ID
Read	ID;
Response	ID$p_1p_1p_1$;
Notes	/pH responds ID019; (so does the Kenwood TS-2000)

IF	Get VFO-A Frequency/Mode etc.
Read	IF;
Response	IF$p_1p_1p_1p_1p_1p_1p_1p_1p_1p_1p_1p_1p_2p_2p_2p_2p_3p_3p_3p_3p_3p_3p_4p_5p_6$ $p_7p_7p_8p_9p_{10}p_{11}p_{12}p_{13}p_{14}p_{14}p_{15}$;
Notes	p_1 : VFO-A Frequency (11 digit) p_2 : VFO-A step size p_3 : VFO-A rit step size p_4 : VFO-A rit enabled (0/1) p_5 : VFO-A xit enabled (0/1) p_6 : always 0 p_7 : always 0 p_8 : RX (p_8=0) or TX (p_8=1) p_9 : mode (TS-2000 encoding, see MD command) p_{10} : always 0 p_{11} : always 0 p_{12} : Split enabled (p_{12}=1) or disabled (p_{12}=0) p_{13} : CTCSS enabled (p_{12}=2) or disabled (p_{12}=0) p_{14} : CTCSS frequency (1 - 38), see CN command p_{15} : always 0

KS	Set CW speed
Set	KS$p_1p_1p_1$;
Read	KS;
Response	KS$p_1p_1p_1$;
Notes	p_1 (1 - 60) is in wpm

KY	Send Morse/query Morse buffer
Set	KY$p_1p_2p_2p_2$...$p_2p_2p_2$;
Read	KY;
Response	KYp_1;
Notes	When setting (sending), p_1 must be a space. When reading, p_1=1 indicates buffer space is available, p_1=0 buffer full p_2: string of up to 24 characters NOT containing ';'. Trailing blanks are ignored in p_2, but if it is completely blank it causes an inter-word space.

LK	Set/Read Lock status
Set	LKp_1p_1;
Read	LK;
Response	LKp_1p_1;
Notes	When setting, any nonzero p_1 sets lock status. When reading, p_1 = 00 (not locked) or p_1 = 11 (locked)

MD	Set/Read VFO-A modes
Set	MDp_1;
Read	MD;
Response	MDp_1;
Notes	Kenwood-type mode list: LSB (p_1=1), USB (p_1=2), CWU (p_1=3), FMN (p_1=4), AM (p_1=5), DIGL (p_1=6), CWL (p_1=7), DIGU (p_1=9)

MG	Set/Read Mic gain (Mic gain slider)
Set	MG$p_1p_1p_1$;
Read	MG;
Response	MG$p_1p_1p_1$;
Notes	p_1 0-100 mapped to -12 ... +50 dB

NB	Set/Read RX1 noise blanker
Set	NBp_1;
Read	NB;
Response	NBp_1;
Notes	p_1=0: NB off, p_1=1: NB1 on, p_1=2: NB2 on

NR	Set/Read RX1 noise reduction
Set	NRp_1;
Read	NR;
Response	NRp_1;
Notes	p_1=0: NR off, p_1=1: NR1 on, p_1=2: NR2 on

NT	Set/Read RX1 auto notch filter
Set	NTp_1;
Read	NT;
Response	NTp_1;
Notes	p_1=0: Automatic Notch Filter off, p_1=1: on

PA	Set/Read RX1 preamp status
Set	PAp_1;
Read	PA;
Response	PAp_1;
Notes	Applies to RX1 p_1=0: RX1 preamp off, p_1=1: on newer HPSDR radios do not have a switchable preamp

PC	Set/Read TX power (Drive slider)
Set	PC$p_1p_1p_1$;
Read	PC;
Response	PC$p_1p_1p_1$;
Notes	p_1 = 0...100

PL	Set/Read TX compressor level
Set	PL$p_1p_1p_1p_2p_2p_2$;
Read	PL;
Response	PL$p_1p_1p_1p_2p_2p_2$;
Notes	p_1 = 0...100, maps to compression 0...20 dB. p_2 ignored when setting, p_2=0 when reading

PS	Set/Read power status
Set	PSp_1;
Read	PS;
Response	PSp_1;
Notes	p_1 = 0: Power on, p_1=1: off When setting, p_1=0 is ignored and p_1=1 leads to shutdown Reading always reports p_1=1

RA	Set/Read RX1 attenuator or RX1 gain
Set	RAp_1p_1;
Read	RA;
Response	RA$p_1p_1p_2p_2$;
Notes	p_1 = 0 ... 99 is mapped to the attenuation range available. HPSDR radios: attenuator range 0...31 dB, HermesLite-II etc.: gain range -12...48 dB. p_2 is always zero.

RC	Clear VFO-A RIT value
Set	RC;

RD	Set or Decrement VFO-A RIT value
Set	RD$p_1p_1p_1p_1p_1$;
Notes	When p_1 is not given (RD;) decrement by 10 Hz (CW modes) or 50 Hz (other modes). When p_1 is given, set VFO-A rit value to the negative of p_1.

RT	Read/Set VFO-A RIT status
Set	RTp_1;
Read	RT;
Response	RTp_1;
Notes	p_1=0: VFO-A RIT off, p_1=1: on

RU	Set or Increment VFO-A RIT value
Set	RU$p_1p_1p_1p_1p_1$;
Notes	When p_1 is not given (RU;) increment by 10 Hz (CW modes) or 50 Hz (other modes). When p_1 is given, set VFO-A rit value to p_1

RX	Enter RX mode
Set	RX;

SA	Set/Read SAT mode
Set	SA$p_1p_2p_3p_4p_5p_6p_7p_8p_8p_8p_8p_8p_8p_8p_8$;
Read	SA;
Response	SA$p_1p_2p_3p_4p_5p_6p_7p_8p_8p_8p_8p_8p_8p_8p_8$;
Notes	p_1=0: neither SAT nor RSAT, p_1=1: SAT or RSAT p_2,p_3,p_4 always zero p_6 = 1 indicates SAT mode (TRACE) p_7 = 1 indicates RSAT mode (TRACE REV) p_8 = eight-character label, here "SAT " when setting, p_6=1 and/or p_7=1 is illegal, and p_4 is ignored.

SD	Set/Read CW break-in hang time
Set	SD$p_1p_1p_1p_1$;
Read	SD;
Response	SD$p_1p_1p_1p_1$;
Notes	p_1 = 0...1000 (in milli seconds). When setting, p_1 = 0 disables break-in

SH	Set/Read VFO-A filter high-water (LSB, USB, DIGL, DIGU only)
Set	SHp_1p_1;
Read	SH;
Response	SHp_1p_1;
Notes	When setting, the Var1 filter is activated p_1 = 0...11 encodes filter high water mark in Hz: 1400 (p_1=0), 1600 (p_1=1), 1800 (p_1=2), 2000 (p_1=3) 2200 (p_1=4), 2400 (p_1=5), 2600 (p_1=6), 2800 (p_1=7) 3000 (p_1=8), 3400 (p_1=9), 4000 (p_1=10), 5000 (p_1=11).

SL	Set/Read VFO-A filter low-water (LSB, USB, DIGL, DIGU only)
Set	SLp_1p_1;
Read	SL;
Response	SLp_1p_1;
Notes	When setting, the Var1 filter is activated p_1 = 0...11 encodes filter low water mark in Hz: 10 (p_1=0), 50 (p_1=1), 100 (p_1=2), 200 (p_1=3) 300 (p_1=4), 400 (p_1=5), 500 (p_1=6), 600 (p_1=7) 700 (p_1=8), 800 (p_1=9), 900 (p_1=10), 1000 (p_1=11).

SM	Read S-meter
Read	SMp_1;
Response	SM$p_1p_2p_2p_2p_2$;
Notes	p_1=0: read RX1, p_1=1: read RX2 p_2 : 0 ... 30 mapped to -127...-19 dBm

SQ	Set/Read squelch level (Squelch slider)
Set	SQ$p_1p_2p_2p_2$;
Read	SQp_1;
Response	SQ$p_1p_2p_2p_2$
Notes	p_1=0: read/set RX1 squelch, p_1=1: RX2. p_2 : 0-255 mapped to 0-100

TX	Enter TX mode
Set	TX;

TY	Read firmware version
Read	TY;
Response	TY$p_1p_1p_1$;
Notes	p_1 is always zero

UP	Move VFO-A one step up
Set	UP;
Notes	use current VFO-A step size

VG	Set/Read VOX threshold
Set	VG$p_1p_1p_1$;
Read	VG;
Response	VG$p_1p_1p_1$;
Notes	p_1 is in the range 0-9, mapped to an amplitude threshold 0.0-1.0

VX	Set/Read VOX status
Set	VXp_1;
Read	VX;
Response	VXp_1;
Notes	p_1=0: VOX disabled, p_1=1: enabled

XT	Set/Read XIT status
Set	XTp_1;
Read	XT;
Response	XTp_1;
Notes	p_1=0: XIT disabled, p_1=1: enabled

D.2 Extended CAT commands supported by piHPSDR

#S	Shutdown Console
Set	#S;

ZZAC	Set/read VFO-A step size
Set	ZZACp_1p_1;
Read	ZZAC;
Response	ZZACp_1p_1;
Notes	p_1 0...16 encodes the step size: 1 Hz (p_1=0), 10 Hz (p_1=1), 25 Hz (p_1=2), 50 Hz (p_1=3) 100 Hz (p_1=4), 250 Hz (p_1=5), 500 Hz (p_1=6) 1000 Hz (p_1=7), 5000 Hz (p_1=8), 6250 Hz (p_1=9) 9 kHz (p_1=10), 10 kHz (p_1=11), 12.5 kHz (p_1=12) 100 kHz (p_1=13), 250 kHz (p_1=14) 500 kHz (p_1=15), 1 MHz (p_1=16).

ZZAD	Move down VFO-A frequency by a selected step
Set	ZZACp_1p_1;
Notes	p_1 encodes the step size, see ZZAC command.

ZZAE	Move down VFO-A frequency by several steps
Set	ZZAEp_1p_1;
Notes	VFO-A frequency moved down by p_1 (0...99) times the current step size

ZZAF	Move up VFO-A frequency by several steps
Set	ZZAFp_1p_1;
Notes	VFO-A frequency moved up by p_1 (0...99) times the current step size

ZZAG	Set/Read RX1 volume (AF slider)
Set	ZZAG$p_1p_1p_1$;
Read	ZZAG;
Response	ZZAG$p_1p_1p_1$;
Notes	p_1 = 0...100, mapped logarithmically to -40 ... 0 dB.

ZZAI	Set/Read auto-reporting
Set	ZZAIp_1;
Read	ZZAI;
Response	ZZAIp_1;
Notes	p_1=0: auto-reporting disabled, p_1>0: enabled. Auto-reporting is affected for the client that sends this command. For p_1=1, only frequency changes are sent via FA/FB commands. For p_1>1, mode changes are also sent via MD commands.

ZZAR	Set/Read RX1 AGC gain
Set	ZZAR$p_1p_1p_1p_1$;
Read	ZZAR;
Response	ZZAR$p_1p_1p_1p_1$;
Notes	p_1 -20...120, must contain + or - sign.

ZZAS	Set/Read RX2 AGC gain
Set	ZZAS$p_1p_1p_1p_1$;
Read	ZZAS;
Response	ZZAS$p_1p_1p_1p_1$;
Notes	p_1 -20...120, must contain + or - sign.

ZZAU	Move up VFO-A frequency by selected step
Set	ZZAUp_1p_1;
Notes	p_1 0...16 selects the size of the step, see ZZAC command.

ZZBA	Move VFO-B one band down
Set	ZZBA;
Notes	Wraps from lowest to highest band.

ZZBB	Move VFO-B one band up
Set	ZZBB;
Notes	Wraps from highest to lowest band.

ZZBD	Move VFO-A one band down
Set	ZZBD;
Notes	Wraps from lowest to highest band.

ZZBE	Move down VFO-B frequency by multiple steps
Set	ZZBEp_1p_1;
Notes	VFO-B frequency moves down by p_1 (0..99) times the current step size

ZZBF	Move up VFO-B frequency by multiple steps
Set	ZZBFp_1p_1;
Notes	VFO-B frequency moves up by p_1 (0...99) times the current step size

ZZBM	Move down VFO-B frequency by selected step.
Set	ZZBMp_1p_1;
Notes	p_1 0...16 selects the size of the step, see ZZAC command.

ZZBP	Move up VFO-B frequency by selected step.
Set	ZZBPp_1p_1;
Notes	p_1 0...16 selects the size of the step, see ZZAC command.

ZZBS	Set/Read VFO-A band
Set	ZZBS$p_1p_1p_1$;
Notes	p_1 0...999 encodes the band: 136 kHz (p_1=136), 472 kHz (p_1=472), 160M (p_1=160) 80M (p_1=80), 60M (p_1=60), 40M (p_1=40), 30M (p_1=30) 20M (p_1=20), 17M (p_1=17), 15M (p_1=15), 12M (p_1=12) 10M (p_1=10), 6M (p_1=6), Gen (p_1=888), WWV (p_1=999).

ZZBT	Set/Read VFO-B band
Set	ZZBT$p_1p_1p_1$;
Notes	p_1 0...999 encodes the band, see ZZBS command.

ZZBU	Move VFO-A one band up
Set	ZZBU;
Notes	Wraps from highest to lowest band.

ZZCN	Set/Read VFO-A CTUN status
Set	ZZCNp_1;
Read	ZZCN;
Response	ZZCNp_1;
Notes	p_1=0: CTUN disabled, p_1=1: enabled

ZZCO	Set/Read VFO-B CTUN status
Set	ZZCOp_1;
Read	ZZCO;
Response	ZZCOp_1;
Notes	p_1=0: CTUN disabled, p_1=1: enabled

ZZFA	Set/Read VFO-A frequency
Set	ZZFA$p_1p_1p_1p_1p_1p_1p_1p_1p_1p_1p_1$;
Read	ZZFA;
Response	ZZFA$p_1p_1p_1p_1p_1p_1p_1p_1p_1p_1p_1$;
Notes	p_1 in Hz, left-padded with zeroes

ZZFB	Set/Read VFO-B frequency
Set	ZZFB$p_1p_1p_1p_1p_1p_1p_1p_1p_1p_1p_1$;
Read	ZZFB;
Response	ZZFB$p_1p_1p_1p_1p_1p_1p_1p_1p_1p_1p_1$;
Notes	p_1 in Hz, left-padded with zeroes

ZZFH	Set/Read RX1 filter high water
Set	ZZFH$p_1p_1p_1p_1p_1$;
Read	ZZFH;
Response	ZZFH$p_1p_1p_1p_1p_1p_1$;
Notes	p_1 must be in the range -9999 ... 9999 and start with a minus sign if negative. If setting, this switches to the Var1 filter first. The convention is such that LSB, the filter high cut is negative and affects the low audio frequencies.

ZZFL	Set/Read RX1 filter low water
Set	ZZFL$p_1p_1p_1p_1p_1$;
Read	ZZFL;
Response	ZZFL$p_1p_1p_1p_1p_1p_1$;
Notes	p_1 must be in the range -9999 ... 9999 and start with a minus sign if negative. If setting, this switches to the Var1 filter first. The convention is such that LSB, the filter low cut is negative and affects the high audio frequencies.

ZZGT	Set/Read RX1 AGC
Set	ZZGTp_1;
Read	ZZGT;
Response	ZZGTp_1;
Notes	p_1=0: AGC OFF, p_1=1: LONG, p_1=2: SLOW, p_1=3: MEDIUM, p_1=4: FAST

ZZGU	Set/Read RX2 AGC
Set	ZZGUp_1;
Read	ZZGU;
Response	ZZGUp_1;
Notes	p_1=0: AGC OFF, p_1=1: LONG, p_1=2: SLOW, p_1=3: MEDIUM, p_1=4: FAST

ZZLA	Set/Read RX1 volume (AF slider)
Set	ZZLA$p_1p_1p_1$;
Read	ZZLA;
Response	ZZLA$p_1p_1p_1$;
Notes	p_1 = 0...100, mapped logarithmically to -40 ... 0 dB.

ZZLC	Set/Read RX2 volume (AF slider)
Set	ZZLC$p_1p_1p_1$;
Read	ZZLC;
Response	ZZLC$p_1p_1p_1$;
Notes	p_1 = 0...100, mapped logarithmically to -40 ... 0 dB.

ZZLI	Set/Read PURESIGNAL status
Set	ZZLIp_1;
Read	ZZLI;
Response	ZZLIp_1;
Notes	p_1=0: PURESIGNAL disabled, p_1=1: enabled.

ZZMA	Mute/Unmute RX1
Set	ZZMAp_1;
Read	ZZMA;
Response	ZZMAp_1;
Notes	p_1=0: RX1 not muted, p_1=1: muted.

ZZMB	Mute/Unmute RX2
Set	ZZMBp_1;
Read	ZZMB;
Response	ZZMBp_1;
Notes	p_1=0: RX2 not muted, p_1=1: muted.

ZZMD	Set/Read VFO-A modes
Set	ZZMDp_1p_1;
Read	ZZMD;
Response	ZZMDp_1p_1;
Notes	Modes: LSB (p_1=0), USB (p_1=1), DSB (p_1=3), CWL (p_1=4) CWU (p_1=5), FMN (p_1=6), AM (p_1=7), DIGU (p_1=7) SPEC (p_1=8), DIGL (p_1=9), SAM (p_1=10), DRM (p_1=11)

ZZME	Set/Read VFO-B modes
Set	ZZMEp_1;
Read	ZZME;
Response	ZZMEp_1;
Notes	p_1 encodes the mode (see ZZMD command)

ZZMG	Set/Read Mic gain (Mic gain slider)
Set	ZZMG$p_1p_1p_1$;
Read	ZZMG;
Response	ZZMG$p_1p_1p_1$;
Notes	p_1 0-70 mapped to -12 ... +50 dB

ZZSA	Move down VFO-A frequency one step
Set	ZZSA;
Notes	VFO-A frequency moved down by the current step size

ZZSB	Move up VFO-A frequency one step
Set	ZZSB;
Notes	VFO-A frequency moved up by the current step size

ZZSG	Move down VFO-B frequency one step
Set	ZZSG;
Notes	VFO-B frequency moved down by the current step size

ZZSH	Move up VFO-B frequency one step
Set	ZZSG;
Notes	VFO-B frequency moved up by the current step size

ZZTX	Get/Set MOX status
Set	ZZTXp_1;
Read	ZZTX;
Response	ZZTXp_1;
Notes	p_1=1: MOX on, p_1=0: off.

ZZUT	Get/Set TwoTone status
Set	ZZUTp_1;
Read	ZZUT;
Response	ZZTXp_1;
Notes	p_1=1: TwoTone on, p_1=0: TwoTone off.

ZZVS	Swap VFO A and B
Set	ZZVS;
Notes	The contents (frequencies, CTUN mode, filters, etc.) of VFO A and B are exchanged.

ZZXV	Get extended status information
Read	ZZVS;
Response	ZZVS$p_1p_1p_1p_1$;
Notes	Status is reported bit-wise in the status word p_1=0-1023. Bit 0: RIT; Bit 1: Lock, Bit2: Lock, Bit3: Split, Bit 4: VFO-A CTUN, Bit 5: VFO-B CTUN, Bit 6: MOX, Bit 7: TUNE, Bit 8: XIT, Bit 9: always cleared.

ZZYR	Get/Set active receiver
Set	ZZYRp_1;
Read	ZZYR;
Response	ZZYRp_1;
Notes	The active receiver is either RX1 (p_1=0) or RX2 (p_1=1).

ZZZD	Move down frequency of active receiver
Set	ZZZDp_1p_1;
Notes	ANDROMEDA extension. p_1 = number of VFO steps. For p_1>10, the number of VFO steps is multiplied with a speed-up factor that increases up to 4 at p_1=30 (corresponds to 3 turns of the VFO dial per second). This implements an over-proportional tuning speed if turning the VFO knob faster and faster.

ZZZE	Handle ANDROMEDA encoders
Set	ZZZE$p_1p_1p_2$;
Notes	ANDROMEDA extension. p_1 encodes the encoder and the direction. p_1= 1-20 maps to encoder 1-20, clockwise p_1=51-70 maps to encoder 1-20, counter clockwise p_2=0-9 is the number of ticks

ZZZI	ANDROMEDA reports
Response	ZZZI$p_1p_1p_2$;
Notes	Automatic generated response for ANDROMEDA controller. The LED with number p_1 shall be switched on (p_2=1) or off (p_2=0).

ZZZP	Handle ANDROMEDA push-buttons
Set	ZZZP$p_1p_1p_2$;
Notes	ANDROMEDA extension. p_1 encodes the button and p_2 means released (p_2=0), pressed(p_2=1) or pressed for a longer time (p_2=2).

ZZZS	Log ANDROMEDA version
Set	ZZZS$p_1p_1p_2p_2p_3p_3p_3$;
Notes	ANDROMEDA extension. The ANDROMEDA type (p_1), hardware (p_2) and software (p_3) version is printed in the log file. The type (p_1) sent by a client does affect the processing of ZZZE and ZZZP commands from that client. Only the cases p_1=1 (original ANDROMEDA console) and p_1=5 (G2 Ultra console) are implemented.

ZZZU	Move up frequency of active receiver
Set	ZZZUp_1p_1;
Notes	ANDROMEDA extension. p_1 = number of steps. For p_1>10, the number of VFO steps is multiplied with a speed-up factor that increases up to 4 at p_1=30 (corresponds to 3 turns of the VFO dial per second). This implements an over-proportional tuning speed if turning the VFO knob faster and faster.

Appendix E

Attaching Morse Keys or Paddles

E.1 CW priorities

As detailed in the next section, there are many ways to produce CW with piHPSDR, namely

a) A key/paddle attached to the radio, and CW handled in the radio FPGA.

b) A key/paddle attached to the radio, but using piHPSDR's built-in iambic keyer

c) A key/paddle attached via GPIO/MIDI controlling piHPSDR's built-in iambic keyer

d) An external keyer attached via GPIO/MIDI

e) Sending CW using CAT messages (see Appendix D).

Case a) requires that the check box `CW handled in Radio` within the `CW` menu is active. On the other hand, cases b) and c) require that this box is inactive. Because it is very difficult to get a CW side tone with very small latency from within piHPSDR, these to cases are not recommended. Cases

d) and e) work independent on whether CW is handled in the radio or in the FPGA. This implies that with `CW handled in Radio` checked, you can still operate CW via CAT messages or via a keyer attached to GPIO or MIDI (this is a standard setup while contesting).

For example, a contest logger might key piHPSDR using CAT commands, while the CW key is attached to the radio. In this case, one wants to be able to abort the message from the contest logger just by hitting the key and smoothly continue CW transmission using the key. This makes clear that one has to prioritise the CW sources such that a CW event with a higher priority aborts a CW event running a lower priority. piHPSDR assigns CAT CW (case e) the lowest priority, a key attached to the radio (cases a, b, c) the highest priority and an external, GPIO or MIDI attached keyer (case d) intermediate priority. In other words, hitting a key attached to the radio aborts a CW message being sent either by CAT CW or by a GPIO/MIDI attached keyer, and hitting the key of a GPIO/MIDI attached keyer aborts a CAT CW message. Note that one can also make dual use of an external keyer, having a key attached there and talking to the keyer from a contest logger. In this case, the priorities are handled inside the keyer, usually in the sense that hitting a key aborts a message being sent that came from the contest logger.

E.2 How to connect

Most SDR radios have the possibility to connect a Paddle or at least a straight key to the radio itself, and then the firmware inside the radio takes care of all the CW processing. If, in addition, the radio has the option to connect a headphone, you will get a low-latency side tone, generated by the radio firmware, in this headphone. This is the easiest case (do not forget to check `CW handled in Radio` within the `CW` menu). If the radio (such as the HermesLite-II) can only connect a straight key, use an external keyer and connect the output of the keyer to the straight key input of the radio. Note, however, that the HermesLite-II does not have an audio codec and thus does not produce a side tone. You can use the keyer output to key a hardware side tone generator and mix its output with the audio that you feed to your headphone. This gives a hardware generated low latency side tone.

It is only slightly more difficult if you have to connect the Morse key to the host computer running piHPSDR. This is necessary, for example, if there is a considerable distance between the radio and the host computer running piHPSDR. Imagine, for example, that the radio is in the attic close to the antenna feed point, but that you are sitting in your living room in front of the host computer running piHPSDR, and that there is a long ethernet cable between your computer and your radio. If piHPSDR runs on a Raspberry Pi, one can use its GPIO lines to connect a keyer. Note, however, that GPIO lines may not be available if they are used for another purpose. The far mor general method of connection is to use MIDI.

E.2.1 RaspPi GPIO

If your host computer is a RaspPi, and if you are running either no controller or Controller1, then there are spare GPIO input lines which you can use for connecting a Morse key (see Appendix H. The lines `CWL` and `CWR` are handled buy the iambic keyer inside piHPSDR, while the lines `CWKEY` and `PTTIN` can be used to connect an external keyer that provides output for key and PTT. All input lines are active-low with a pull-up, so they have to be connected to ground in the active state. Appendix H lists in detail which GPIO line is used by which controller option.

For the other GPIO-based controllers, there are not enough ,,spare" GPIO lines to fully support CW over GPIO, so consider using MIDI in this case.

E.2.2 MIDI

If running piHPSDR on non-RaspPi Linux or Macintosh computers, there are no GPIO lines. The same problem arises for piHPSDR running on a RaspPi and using a controller which does not leave enough spare GPIO lines. In these cases, the best choice to get "the key into the computer" is to use MIDI. There are low-cost micro controllers which can be programmed such that they act as MIDI input devices if connected (via USB) to the host computer (in the simplest case, 32U4 based micro controllers like the Arduino Leonardo or the Teensy LC).

There are two types of external keyers (usually microcontroller based) that

send MIDI messages to piHPSDR: one does not further processing and merely report the closure and the opening of the contacts of the left and right paddle. In this case, the piHPSDR internal iambic keyer is used so the appropriate commands to invoke through MIDI are `CW Left` and `CW Right`. Other keyers do the keying inside the microcontroller and send CW key-down/up and PTT on/off MIDI messages, in this case assign the commands `PTT (keyer)` and `CW Key (keyer)` to the MIDI events. Note that in this setup, the external keyer *must* generate both a key-down and a PTT signal, and the keyer must be configured such that the PTT signal arrives earlier than the first key-down. A good value for such a PTT ,,lead-in" delay is about 100 msec. For a K1EL ,,WinKey" keyer and Arduino clones thereof, which are the most frequently used type of keyers, such a delay is easily configured. Depending on the audio subsystem, the latency of the side tone, when using the internal keyer, may be a problem but there are zero-latency solutions for this (see Chapter 13.2).

Using the `MIDI` menu to assign the `CW Key (Keyer)` and `PTT (keyer)` commands is a little bit tricky, since the menu only allows to assign an command to the latest event received. Proceed as follows:

- Press and hold one of the two paddles. The keyer will start sending a NoteOn message for PTT, but then soon infinitely send NoteOn/Off messages for key up/down. Use the `MIDI` menu to assign the `CW Key (keyer)` command to this MIDI event.
- Then, release the paddle. Some time (the hang time of the keyer) after the last key-up, a NoteOff message for PTT will be sent. After this, you can assign the `PTT (keyer)` command to this MIDI event. You should see that it has a different Note value than you saw while the keyer was sending the infinite string of dots or dashes.

Appendix F

Running piHPSDR alongside with DigiMode programs

In this section, we will cover four different scenarios, namely

- piHPSDR and digi mode program running on different computers.
- piHPSDR and digi mode program running on the same LINUX computer (including RaspPi) using ALSA.
- piHPSDR and digi mode program running on the same LINUX computer (including RaspPi) using PipeWire.
- piHPSDR and digi mode program running on the same MacOS computer.

F.1 Running piHPSDR and Digi on different computers

This typically is the situation if piHPSDR is running on a computer with a very small screen, such as it is the case if you are using a piHPSDR controller or the G2 front panel. This situation also occurs if you want to run the digi mode program on a Windows computer, since (as far as I know) piHPSDR has not yet been adapted to run with the Windows operating system.

There is not much to say here, because this setup is largely the same as for conventional (analog) rigs: you need a sound card connected to the computer running the digi program, and then you connect the sound card either to the headphone and mic jacks of the radio, or to the input/output of a sound card connected to the computer running piHPSDR. In the latter case, you have to select this sound-card for local audio output in the `RX` menu, and for local microphone input in the `TX` menu.

Unless you are using VOX, you also need a CAT command connection between the digi mode program and piHPSDR. Via CAT commands, the digi mode program can induce RX/TX transitions in piHPSDR, change frequencies or modes, etc. If both computers are connected via ethernet, TCP is the method of choice for such a connection. piHPSDR listens on port 19090 (TCP CAT connection must be enabled in the `CAT/TCI` menu, and the port number can also be changed there). For standard digi mode operation, choose the Kenwood TS-2000 radio model. If your digi mode program can use the `hamlib` CAT connection library (for example, both FLdigi and WSJTX can), choose the ,,OpenHPSDR PiHPSDR" radio model.

If there is no ethernet connection between the computer running piHPSDR and the computer running the digi mode program, you need two USB-to-serial interfaces and must connect them via a null modem. Then enable CAT on the serial port both in piHPSDR (`CAT/TCI` menu) and the digi mode program.

F.2 Running piHPSDR and Digi on the same computer

In this case it is strongly recommended that the audio never goes analog, but is exchanged (transferred to and from) as digital data between piHPSDR and the digi mode program. To this end, you need virtual audio devices known as virtual audio cables or loop-backs. A virtual audio cable is a pair of connected (virtual) audio devices, one input (,,microphone") device and one output (,,headphone") device. You can, for example, open the output device of such a virtual cable in program X at a place where you could also open a device driving a headphone. In another program Y, you can open the input device of the same virtual audio cable at a place where you could also open

a device attached to a microphone. The ,,trick" is now that all audio data which program X sends to the output device can be retrieved by program Y by reading the input device.

For digi mode operation, you need two such virtual audio cables, which we will name here RXcable and TXcable, just to give an example. The idea behind the names is, that audio data flows through the RXcable upon RX and through the TXcable while transmitting. Having said this, it is clear that

- In the piHPSDR RX menu, chose the output device of RXcable for audio output.
- In the piHPSDR TX menu, chose the input device of TXcable for audio input.
- In the audio menu of the digi mode program, choose the input device of RXcable for audio input.
- In the audio menu of the digi mode program, choose the output device of TXcable for output output.

With this, the only question remaining is, how to create virtual audio cables, and how to access them. This is not only different between Linux and MacOS, but also is different on Linux depending on whether the ALSA compile time option (see Appendix G) was activated during compilation of piHPSDR. If compiling from the sources you get the PulseAudio module if you do not change the Makefile (see Appendix G). Note that since some time, PulseAudio has been replaced by PipeWire in all relevant LINUX distributions. This does not matter to piHPSDR since normally also the `pipewire-pulse` module is installed which lets applications using the PulseAudio API (such as piHPSDR using the PulseAudio audio module) run under PipeWire. The (user) commands for listing available devices, or for creating virtual audio cables, are slightly different between PulseAudio and PipeWire. This manual covers the PipeWire case.

F.2.1 Virtual Audio Cables in ALSA

Using the ALSA sound library, the command for creating the two virtual cables (put everything in one line) is

```
sudo modprobe snd-aloop index=5,6 id=RXcable,TXcable
enable=1,1 pcm_substreams=2,2
```

For the convenience of the reader a shell script `LINUX/alsa.vac.sh` is provided which contains the command given above. The two indices chosen (5, 6) should leave enough head-room for sound devices already present. On a ,,naked" RaspberryPi, indices 0 and 1 refer to the HDMI and headphone audio output devices, but more indices may already be assigned if you have plugged in additional sound cards (use two larger numbers in this case). You can verify the existence of the virtual audio cables using the command

```
aplay -l
```

and part of the output, as obtained on my RaspberryPi, is printed here:

```
card 5: RXcable [Loopback], device 0: Loopback PCM [Loopback PCM]
  Subdevices: 2/2
  Subdevice #0: subdevice #0
  Subdevice #1: subdevice #1
card 5: RXcable [Loopback], device 1: Loopback PCM [Loopback PCM]
  Subdevices: 2/2
  Subdevice #0: subdevice #0
  Subdevice #1: subdevice #1
card 6: TXcable [Loopback], device 0: Loopback PCM [Loopback PCM]
  Subdevices: 2/2
  Subdevice #0: subdevice #0
  Subdevice #1: subdevice #1
card 6: TXcable [Loopback], device 1: Loopback PCM [Loopback PCM]
  Subdevices: 2/2
  Subdevice #0: subdevice #0
  Subdevice #1: subdevice #1
```

Remember that the device with index 5 is our RXcable and index 6 belongs to TXcable since not all programs (including piHPSDR) report the names (RXcable, TXcable) assigned. It is important to realise that each ,,cable" offers two sub-devices: the first one (#0) is the input (,,microphone") device

and the second one (#1) is the output (,,headphone") device. Note that the two devices created by ,,modprobe" only exist until the next shut-down of the computer and you have to re-create them each time after booting. There are ways to make this permanent (search the internet) or to include the ,,modprobe" in a startup script that is executed once after each boot (again, search the internet). Now we show how to set this up and show piHPSDR's RX and TX menu, as well as the audio menu of WSJTX.

We begin with the piHPSDR settings in the RX (Fig. F.1) and TX (Fig. F.2) menu:

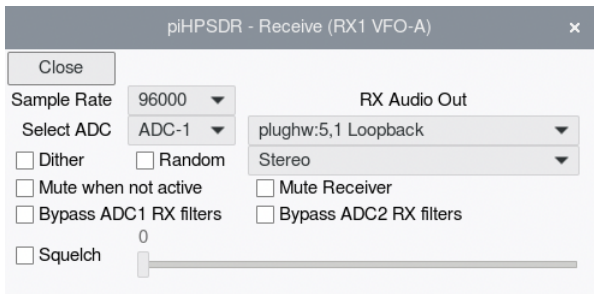

Fig. F.1: RX menu settings for using loop-back with ALSA.

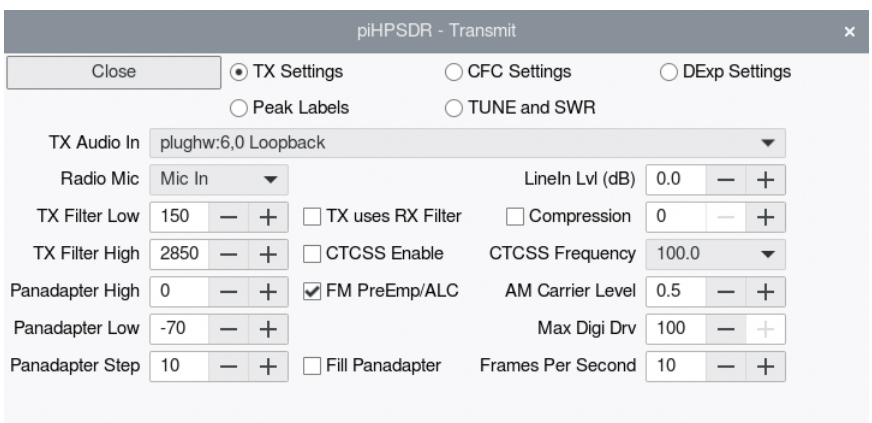

Fig. F.2: TX menu settings for using loop-back with ALSA.

Is clearly seen that the output subdevice of RXcable (5,1) is used RX audio output, and the input subdevice of TXcable (6,0) is used for TX audio input. The corresponding WSJTX settings audio tab looks similar (Fig. F.3, only the upper part of the window is shown):

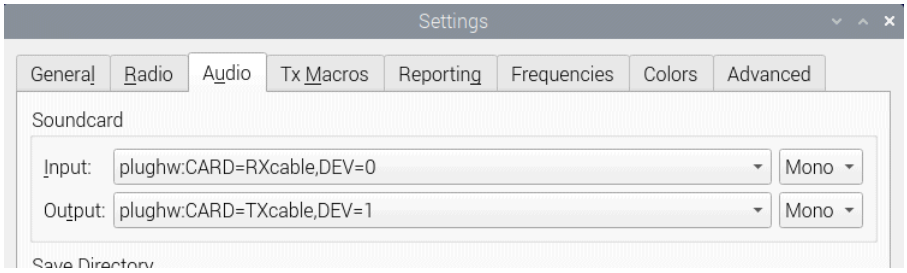

Fig. F.3: WSJTX audio settings for using loopback with ALSA.

Here, the *input* device of RXcable is used for audio input, and the *output* device of TXcable for audio output. Note this program reports the clear names of the devices rather than its ID numbers. Finally, we show the WSJTX ,,Radio" tab for establishing the CAT connection, although this is not specific to ALSA (Fig. F.4):

Important here is to choose the correct rig model `OpenHPSDR PiHPSDR` and port `:19090` (note the colon!). The other parameters in the left side of the tab have no meaning for a TCP connection. In the right tab, choose `CAT` as the PTT method (TRX transition controlled by CAT commands). Checking `Data/Pkt` takes care WSJTX switches piHPSDR to DIGU mode, and using `Fake It` for the Split Operation method is just my personal preference.

F.2.2 PipeWire: Null-Sink Modules as Virtual Audio Cables

Note: This section is relevant if the PulseAudio audio module has been selected at compile time. Meanwhile PulseAudio has been replaced by PipeWire in most recent LINUX distributions. Fortunately, there is a module called `pipewire-pulseaudio` which which lets applications using the PulseAudio API (such as piHPSDR using the PulseAudio audio module) run under PipeWire. There are some subtle differences at command line level when it comes to creating virtual devices or adjusting volumes. This section always discusses the Pipewire variants.

There are no explicit loop-back devices with PipeWire since they are not needed. For *every* PipeWire output device, there is a corresponding ,,Monitor" device which can be used for sound input, and where the data sent to

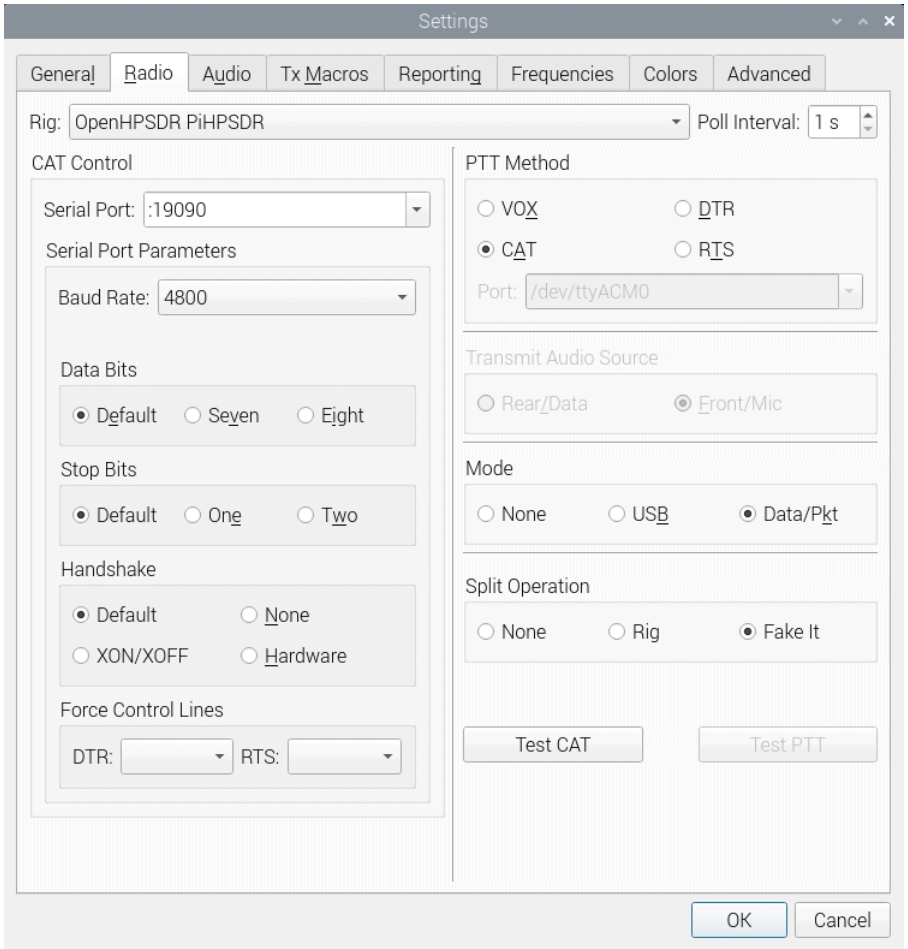

Fig. F.4: WSJTX radio settings for CAT connection to piHPSDR.

the primary (output) device can be read. This means, you can send the RX audio to the headphone device as you would do for normal SSB operation, and can then use the Monitor of the headphone device as audio input in WSJTX. We describe a different setup where we create two dummy ("null-sink") output devices. It should be clear how to change this procedure if you want to listen (with the headphone) to the digi mode RX signal while it is processed in the digi mode program. We call the two dummy output devices RXcable and TXcable just to demonstrate how it works. This is done by the two commands (do not enter the line breaks)

```
pactl load-module module-null-sink sink_name=RXcable rate=48000
sink_properties="device.description=RXcable"

pactl load-module module-null-sink sink_name=TXcable rate=48000
sink_properties="device.description=TXcable"
```

One can likewise execute the script `LINUX/pipewire.vac.sh` in which the two commands above are contained.

To control whether the output devices have been created correctly, we use the command `pactl list sinks`, which produces a very long output from which only the relevant parts are reported here:

```
...
Sink #104
    State: SUSPENDED
    Name: RXcable
    Description: RXcable Audio/Sink sink
    Driver: PipeWire
    Sample Specification: float32le 2ch 48000Hz
...

Sink #111
    State: SUSPENDED
    Name: TXcable
    Description: TXcable Audio/Sink sink
    Driver: PipeWire
    Sample Specification: float32le 2ch 48000Hz
...
```

Likewise, check the availability of the input devices using the command `pactl list sources`, from the output we quote

```
...
Source #104
    State: SUSPENDED
    Name: RXcable.monitor
    Description: Monitor of RXcable Audio/Sink sink
    Driver: PipeWire
```

```
    Sample Specification: float32le 2ch 48000Hz
...
Source #111
    State: SUSPENDED
    Name: TXcable.monitor
    Description: Monitor of TXcable Audio/Sink sink
    Driver: PipeWire
    Sample Specification: float32le 2ch 48000Hz
...
```

These devices only exist until the computer is shut down, which means that the commands given above have to be repeated after rebooting the machine. They can be made permanent (you will find instructions in the internet), but my recommendation is to create a shell script that creates these devices and have this script automatically executed each time you start the computer.

The necessary configuration for piHPSDR and WSJTX is almost evident. The piHPSDR settings are shown in Fig. F.5 (RX menu) and in Fig. F.6 (TX menu): RXcable has been selected for local RX audio output, while the Monitor of TXcable is chosen as the local microphone.

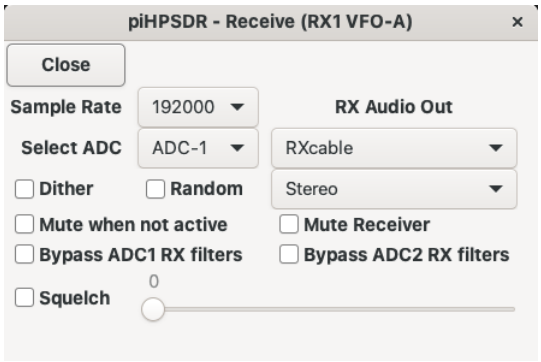

Fig. F.5: RX menu settings with PulseAudio/PipeWire

In the WSJTX audio tab, the input/output assignments are just reversed, as shown in Fig. F.7. The CAT connection between WSJTX and piHPSDR is the same as in the ALSA case.

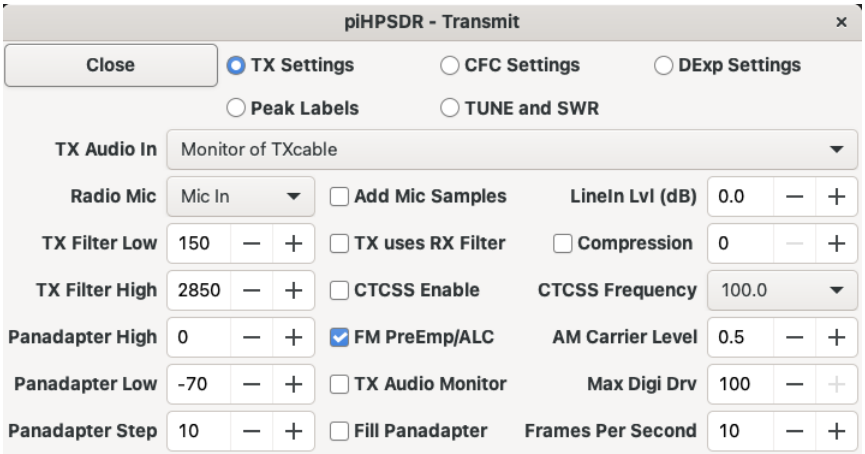

Fig. F.6: TX menu settings with PulseAudio/PipeWire

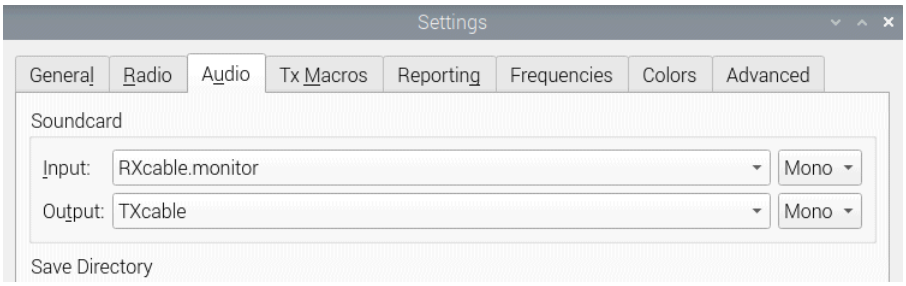

Fig. F.7: WSJTX audio settings with PulseAudio/PipeWire

Note on FLdigi and qSSTV

Some popular programs, including FLdigi and qSSTV, can use PipeWire but have no user interface to choose the PipeWire input and output devices. This means, they will only use the default PipeWire input and output device. So to use FLdigi and/or qSSTV, you have to make TXcable the default output device and the monitor of RXcable the default input device, so FLdigi/qSSTV will always read their audio input from the monitor of RXcable and put the audio they produce to TXcable. To achieve this, simply use the two commands

```
pactl set-default-sink TXcable
pactl set-default-source RXcable.monitor
```

You can check which audio devices are used by default by PipeWire with the commands

```
pactl get-default-sink
pactl get-default-source
```

F.2.3 Using the ,,loopback" program with MacOS

The standard audio option for MacOS is PORTAUDIO which has no built-in virtual audio cables. In this case the easiest, most flexible, and (from my side) most recommended option is to buy a third-party product, ,,loopback" from RogueAmoeba (`https://rogueamoeba.com/loopback/`). Note I have no connections with that company, I am just using this product and can recommend it. It can not only be used to create loop-back devices, but you get a patch panel so you can, for example, send the RX audio both to two different devices at the same time. The following picture just shows my setup (Figs. F.8 and F.9).

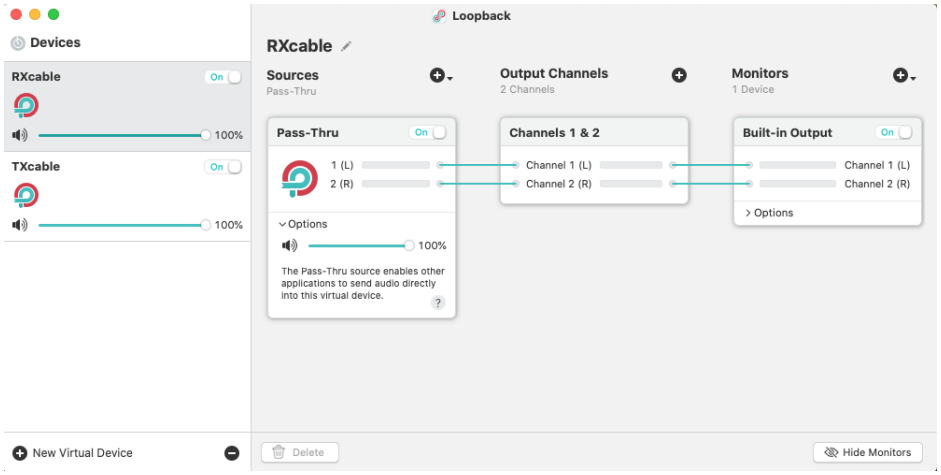

Fig. F.8: Setting up ,,loopback" on MacOS, RXcable

As one can see, two devices have been created, with name RXcable and TXcable. While the configuration of the TXcable is the minimum (default) one, all the output sent to RXcable is further routed to the Build-in Output (headphone connector) of the Macintosh. One could also modify the TXcable to accept a microphone as a second input, but then the microphone must be mutable (On/Off switch) such that it does not pickup noise while doing digi mode. In piHPSDR, simply choose RXcable and local RX output in the `RX` menu, and a local microphone with device TXcable in the `TX` menu. In WSJTX, simply choose RXcable as the input and TXcable as the output device.

As an additional bonus, loopback can also mix output devices. For example,

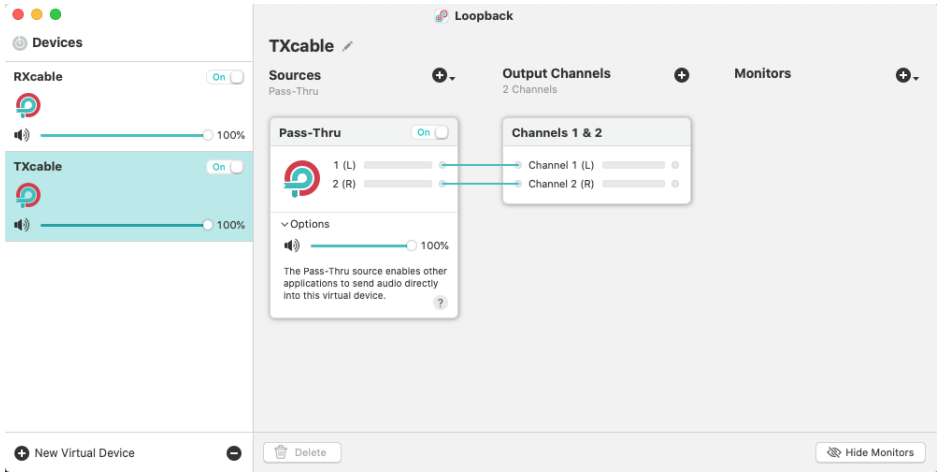

Fig. F.9: Setting up ,,loopback" on MacOS, TXcable

one can create an additional loop-back device with, say, name RX2, which can also be connected to the headphone output as shown for RXcable. If piHPSDR is running two receivers, one can use the RX menu to send the audio output of the first receiver to RXcable and the audio output of the second receiver to RX2. Additionally one can activate only the left channel for the first and only the right channel for the second receiver (see the RX menu). With this setup, one gets the audio output of the first receiver on the left ear and the audio output of the second receiver on the right ear. This can be very convenient for hunting DX in split mode, because one then hears the hounds and the fox on different ears.

Transceiver control (CAT connection) in MacOS is the same as described for Linux.

Appendix G

Compile-time options

There is a number of compile-time options that are shown on the initial screen. When compiling from the sources, you can choose which options you want (or do not want). The reason for making a feature optional is usually that it depends on some external library which may or may not be available on the computer intended to run piHPSDR.

In contrast to earlier version of piHPSDR which required editing the `Makefile`, compile-time options different from the default ones can be enforced by creating a text file file within the piHPSDR directory which has the name `make.config.pihpsdr`.

If this file does not exist, compilation proceeds with the default options defined in the `Makefile`. The file `make.config.pihpsdr` should contain lines of the form

```
OPTION=VALUE
```

which then assigns a value to the option. The value may be empty, in this case no value is assigned to the option. In most cases, the value is either `ON` or `OFF` depending on whether the compile-time option shall be activated or not. The exception is the `AUDIO` option, which may take various values (see below). For example, for compilation without GPIO support the file `make.config.pihpsdr` contains the line

```
GPIO=OFF
```

After modifying the file `make.config.pihpsdr`, you have to recompile the

whole program using the commands

```
make clean
make
```

Here is a list of options with their default values.

`GPIO` (default: `ON`) This option is needed on a RaspberryPi if you want to use GPIO input/output lines from within piHPSDR, for example since you have a controller (Controller1, Controller2, of G2 front panel) attached or want to use GPIO lines for CW or PTT. This option needs both the `libgpiod` and `libi2c` libraries on your system, therefore it **is important to deactivate this option if you compile for a Desktop PC or Laptop running LINUX**. This is so because Desktop PCs/Laptops do not have GPIO lines and therefore no libgpiod is available. For MacOS (which also does not have GPIO lines), this option is automatically disabled.

If you do not plan to let piHPSDR control any of the GPIO I/O lines you better compile without this option. This is especially true if you plan to attach some third-party hardware (such as sound card ,,hats") to the GPIO.

G2 Ultra. Second-generation ANAN G2 radios ,,G2 Ultra" have a micro-controller inside that controls the front panel and communicates with piHPSDR over a serial line that uses two GPIO lines. For these machines, it is recommended to compile with GPIO turned OFF.

`MIDI` (default: `ON`) This option activates the possibility to control piHPSDR via MIDI devices. It should normally be activated, unless you port piHPSDR to some other operating system without MIDI support.

`SATURN` (default: `ON`) This option is for piHPSDR running on the CM4 module within a SATURN/G2 radio. It allows piHPSDR to directly communicate with the FPGA via XDMA. While it does no harm to have this option activated on other systems, it also has no function there and therefore it is automatically disabled on MacOS.

`USBOZY` (default: `OFF`) This option includes support for legacy HPSDR hardware connected via USB. Note this option needs the `libusb-1.0` library be present on your system.

SOAPYSDR (default: OFF) This option enables piHPSDR to communicate with radios through the SoapySDR library. If this library is not present on your system, this option must be turned OFF. If you do not plan to connect SoapySDR radios, turning OFF this option may accelerate piHPSDR startup a little bit.

STEMLAB (default: OFF) This is an option for RedPitaya based radios that have a web interface that must be used to start the SDR application. If you have a RedPitaya exclusively being used as a radio, this app is most likely auto-started when powering up the RedPitaya so you do not need the STEMLAB option. If you have a RedPitaya and no auto-start of the SDR application, and not compiled piHPSDR with the STEMLAB option, you can open a web browser on your host computer, start the SDR application on the RedPitaya, and then start piHPSDR. With the STEMLAB option, this can all be done by piHPSDR. This option is not activated by default since it delays the startup a little bit and this should be avoided if there is no need for it, and it needs the `libcurl` library to be installed on your system.

TTS (default: ON) Activates the text-to-speech capability, meant to improve accessibility for the blind (see section C.1). With this option, hitting e.g. function key F1 reads the current frequency aloud.

AUDIO (default: empty) On Linux systems (including the Raspberry PI), if setting the value to `ALSA`, the standard Linux ALSA sound library is used for local audio output and local microphone input. In all other cases PulseAudio is used. On MacOS, the `AUDIO` choice has no effect since the PortAudio module is used no matter what the `AUDIO` setting is.

NR34LIB (default: OFF) piHPSDR offers two noise reduction models (NR3 and NR4, see chapter 8.3) that are actually implemented outside the WDSP library. NR3 is a neural network based noise reduction model called `RNNnoise`, while NR4 uses the `libspecbleach` library. Source code from the repositories indicated below has been included in the piHPSDR code to make life easier for non-expert users.

It is also possible to have `RNNnoise` and `libspecbleach` installed separately. This code is available on github:

RNNoise: `https://github.com/xiph/rnnoise.git`
SpecBleach: `https://github.com/lucianodato/libspecbleach`

So if you want your own versions of RNNnoise and libspecbleach installed (either installed from a software repository or compiled from scratch) and want to use these, set `NR34LIB=ON`. It is up to you to ensure that include files and libraries are installed in standard locations such that they can be found in the compilation process. The (only) advantage of installing these libraries from scratch is, that you (or the auto-configuration process) can take care to compile these libraries with CPU-specific optimisations (e.g. using AVX2 with x86 processors) enabled.

Appendix H

RaspPi GPIO Lines for Controllers/Panels, CW, PTT

This section is only for piHPSDR running on RaspberryPi or similar systems with a GPIO (general purpose input/output) header. piHPSDR uses `libgpiod` to access the GPIO, so this may also apply to other single-board computers with a GPIO.

Fig. H.1 lists which GPIO lines are used for which controller option. In the first column, the GPIO pin number (0–31) is listed. The second column indicates the function often associated with this pin in the RaspberryPi community, and the third column states on which connector (P1 or P5) this pin can be found. P1 is the main 40-pin GPIO header of the RaspberryPi, while P5 is an auxiliary connector that carries pins one should normally not use (like those for a second I2C devices). The four last columns indicate the use of this GPIO pin in four different setups, namely with no controller, with the Controller1, with the Controller2 equipped with single encoders, and finally with the Controller2 equipped with double encoders. The last case is electrically identical to the first-generation G2 front panel. In red, with the string `do not use!`, GPIO pins are marked which one should not use from within piHPSDR for the case at hand, since they are often use for other purposes (for example, I2C interfaces). This ensures that, for example, piHPSDR even if compiled with the GPIO option will run smoothly together with additional hardware such as ,,audio hats". Note the for some special radios (e.g. the RadioBerry), the default ,,No Controller" assignment is different due to the

GPIO #	Function	Conn.	No Controller	Controller 1	Controller 2 V1	Controller2 V2
0	ID_SD	P1	do not use!	do not use!	do not use!	do not use!
1	ID_SC	P1	do not use!	do not use!	do not use!	do not use!
2	i2c_1 SDA	P1	do not use!	do not use!	do not use!	do not use!
3	i2c_1 SCLK	P1	do not use!	do not use!	do not use!	do not use!
4	GPCLK0	P1	do not use!	E4_A	E3_A	E3_T_B
5		P1	**CWL**	S4	**available**	E2_A
6		P1	**CWR**	S3	**available**	E2_B
7	CE1	P1	do not use!	E4_F	**available**	E3_B
8	CE0	P1	do not use!	E3_F	E5_B	E5_T_A
9	MISO	P1	do not use!	**CWL**	**CWL**	E3_A
10	MOSI	P1	do not use!	**CWKEY**	**CWKEY**	E4_B
11	SCLK	P1	do not use!	**CWR**	**CWR**	E4_A
12		P1	**CWKEY**	S2	**CWOUT**	E5_B
13	PWM1	P1	do not use!	S1	**PTTOUT**	E5_A
14	TxD	P1	do not use!	**PTTIN**	**PTTIN**	**PTTIN**
15	RxD	P1	do not use!	**PTTOUT**	I2C IRQ	I2C IRQ
16		P1	**PTTIN**	E3_A	E4_A	E4_T_B
17		P1	**MICboost**	VFO_B	VFO_B	VFO_B
18	PCM_CLK	P1	do not use!	VFO_A	VFO_A	VFO_A
19	PWM_FS	P1	do not use!	E3_B	E4_B	E4_T_A
20	PCM_DIN	P1	do not use!	E2_A	E2_A	E2_T_B
21	PCM_DOUT	P1	do not use!	E4_B	E3_B	E3_T_A
22		P1	**PTTOUT**	FUNCTION	E2_F	E2_F
23		P1	**CWOUT**	S6	E4_F	E4_F
24		P1	**MICsel**	S5	E5_F	E5_F
25		P1	**MICbias**	E2_F	E5_A	E5_T_B
26	PWM0	P1	do not use!	E2_B	E2_B	E2_T_A
27		P1	**MICptt**	MOX	E3_F	E3_F
28	i2c_0 SDA	P5	do not use!	do not use!	do not use!	do not use!
29	i2c_0 SCL	P5	do not use!	do not use!	do not use!	do not use!
30		P5	do not use!	do not use!	do not use!	do not use!
31		P5	do not use!	do not use!	do not use!	do not use!

Fig. H.1: GPIO lines used with various controllers. Note that Controller2V2 and the G2V1 front panel share the used GPIO lines. The assignment is different for a RadioBerry, see chapter 17.5. None of the GPIO lines printed in blue are used on Anan-G2 radios.

lack of available GPIO lines. The function of GPIO lines actually used by the controllers are indicated in black colour. These assignements can be changed via the `gpio.props` file to support home-brewn clones of these controllers with a different wiring.

Lines which are available since they are not used by the controller are marked in blue colour. Note that no such additional ,,blue" GPIO lines are accessible and available if piHPSDR runs inside the Saturn G2 radios. piHPSDR supports up to four input lines (`CWL`, `CWR`, `CWKEY`, and `PTTIN`) and up to six output lines (`PTTOUT`, `CWOUT`, `MICboost`, `MICsel`, `MICbias` and `MICptt`). Fig.

H.1 shows the default assignment which can, however be different for specific radios and which can be overriden via the `gpio.props` file. The function of these additional lines is

`CWL` (Input with pull-up, active low). This input triggers a "left paddle pressed/ released" event for the iambic keyer built into piHPSDR. Note that to use the built-in iambic keyer, the checkbox `CW handled in Radio` has to be unchecked. This input is typically connected to a Morse paddle key.

`CWR` (Input with pull-up, active low). This input triggers a "right paddle pressed/ released" event for the iambic keyer built into piHPSDR. Note that to use the built-in iambic keyer, the checkbox `CW handled in Radio` has to be unchecked. This input is typically connected to a Morse paddle key.

`PTTIN` (Input with pull-up, active low). This input triggers a "PTT on/off" signal. Not that it does not *toggle* the PTT state, but enforces PTT as long as the input is pulled low, and releases PTT when it goes high. This input is typically connected to the PTT button of a microphone or the PTT output of an external CW keyer.

`CWKEY` (Input with pull-up, active low). This input triggers a "CW Key down" event. An RF pulse is emitted in CW mode as long as the input is pulled low and the radio in TX state. This input is typically connected to the KEY output of an external CW keyer. Note that nothing happens if the radio is not put into TX mode beforehand. Therefore, to use an external CW keyer, one needs both a KEY and PTT connection to the keyer. To avoid chopping of the first Morse element (dot or dash), PTT should become active shortly before the first key-down event occurs („PTT lead-in time"). This lead-in time can normally be adjusted for external CW keyers, I mostly use 150 msec to be on the safe side.

`PTTOUT` (Output, active low). This output indicates that piHPSDR is in the TX state. This output is typically used together with radios such as the Adalm Pluto to activate a RF amplifier, because there is no hardware output at the Adalm which indicates that the device transmits. The output can also be used to activate an external power amplifier if the radio has not PTT output.

`CWOUT` (Output, active low). This output monitors the state of the built-in iambic keyer (low = key down, high = key up). This can be used to drive external hardware with a tone generator that generates a low-latency side

tone, mixes it with the RX audio output, and feeds the combined signal to the head phone. For direct keying (using the CWKEY line or the `CW KEY (keyer)` command via MIDI) this output also follows the key-down/up state.

`MICboost` (Output, active low) This output indicates that the microphone preamp should have some additional gain, e.g. when connecting dynamic microphones. For HPSDR radio, it follows the "MIC boost" setting in the TX menu. For non-HPSDR radios, the state of MIC boost can be manually specifies via the props file. This GPIO output line can be used in controllers which have an audio codec.

`MICsel` (Output, active low) This output can be used in controllers with an audio codec and a TRS microphone jack, to reproduce the behaviour of Orion-II radios such as the Anan 7000DLE or the Anan G2 series. If the output is active, the tip of the jack is connected to PTT and the ring to Mic and Bias, if it is inactive, it is the other way round.

`MICbias` (Output, active low) This output can be used in controllers with an audio codec and a TRS microphone jack, to reproduce the behaviour of Orion-II radios such as the Anan 7000DLE or the Anan G2 series. If the output is active, a bias voltage is applied to the contact (tip or ring) connected to Mic input.

`MICptt` Output, active low) This output can be used in controllers with an audio codec and a TRS microphone jack, to reproduce the behaviour of Orion-II radios such as the Anan 7000DLE or the Anan G2 series. If the output is active, the PTT input at the Mic/PTT jack is disabled.

The reason for choosing ,,active low" for the output lines is that in the default setup, the GPIO lines are switched to ,,input with pull-up" upon booting. An input line with a built-in pull-up resistor is, however, nearly indistinguishable from an output line with high level.

Appendix I

RaspPi: Activating the I2C or serial interface

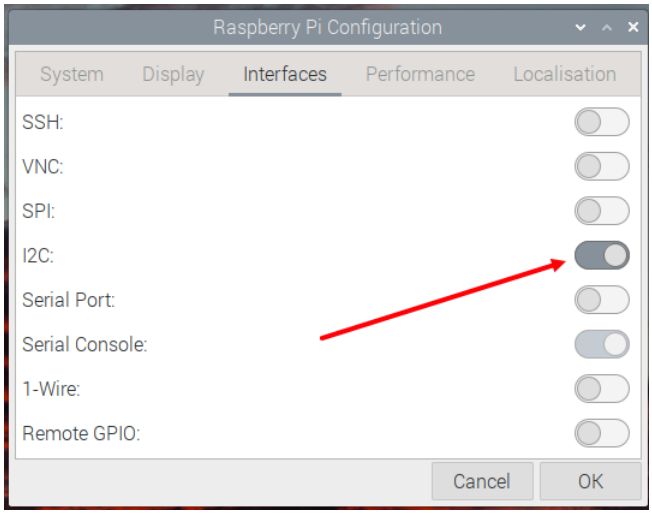

Fig. I.1: Enabling the I2C controller.

If you want to use the Controller2 or the G2 front panel controller, the Raspberry Pi I2C interface must be enabled. This is done by opening `Menu`→`Preferences`→ `Raspberry Pi Configuration` and moving to the `Interfaces` tab (shown in Fig. I.1). In the I2C line, move the button to the right (marked by the red arrows).

If you need the piHPSDR built-in serial port, activate the `Serial Port` button and deactivate the `Serial Console` button. Note that the built-in serial port uses GPIO lines that may conflict with GPIO lines used for the controller, so you should only activate it when you a) need it and b) have checked `No Controller` in the initial GPIO menu (or have not compiled the program with GPIO support).

Restart the computer after making the changes. The other buttons (SSH, VNC, etc) need not be changed and can be set according to your needs. SSH allows you to log in into your RaspPi over a network, and VNC allows operating the Pi from another computer, using the screen and keyboard and mouse from that other computer to control the Pi.

Appendix J

Installation of piHPSDR from the sources (Linux, RaspPi)

This procedure has been with current version of the RaspPi (Pi4B and Pi5) with various versions of a fresh and plain vanilla operating system (,,Bulls-Eye", ,,BookWorm" and ,,Trixie"). It has also been tested with an Ubuntu (Version 25) LINUX installed on a RaspPi. Normally I use the 64-bit versions of the operating system since this is the standard option, but it should also work with 32-bit versions. The RaspPi operating system can be obtained from

`https://www.raspberrypi.com/software/operating-systems/`

General Remarks. Installation from the sources has number of benefits compared to using binaries that have been compiled elsewhere:

- You get binaries that exactly ,,fit" to your operating system, it is not important which version of the operating system you use.
- The procedure is the same for 32-bit and 64-bit systems, so it does not matter which of the two variants you run.
- For all the compile time options (see Appendix G) you can individually choose whether you want to activate or deactivate this option.

The following *caveat* applies both to the binary installation and the compilation from the source code: If you want to use the Controller2 or the G2 front panel controller, the Raspberry Pi I2C interface must be enabled, as described in Appendix I.

—— Administrator privileges on Linux systems ——

The whole installation procedure depends on that your user account is an administrator account, such that you can execute ,,sudo" commands in a terminal window. Some Linux distributions maintain a `sudoers` file where users that are allowed to execute administrative task via the `sudo` program are listed. In normal cases (you own the computer and/or have installed the Linux operating system) you should have administrator privileges.
On a Raspberry Pi, the user account that you created during installation will have administrator privileges by default, so you can execute `sudo` commands without being asked a password. On Ubuntu, the situation is similar but you are asked for your password when executiong `sudo` commands.

J.1 Fresh install from the internet

To install from the sources, open a terminal window. If you do so, you get a prompt and are in your home directory. It is assumed that no directory named `pihpsdr` exists. If it does, then this is likely a left-over from some previous attempt to install the program and then it should be moved elsewhere, since it may contain files with saved preferences (`*.props`) which you may want to re-use.

Note for Anan-G2 radios. On these radios, the factory installation contains the `pihpsdr` directory in `$HOME/github` rather than in `$HOME`. It makes sense to keep this location. So rename the directory `$HOME/github/pihpsdr` to (say) `$HOME/github/pihpsdr.backup` to have a backup of the existing installation, this also makes sure that after the renaming there is no directory with name `pihpsdr` left in `$HOME/github`. Then you can use the following instructions provided that you replace the commands `cd $HOME` with `cd $HOME/github`.

I have seen some cases where even the `git` command is not installed in the base system. To this end, start with the command

```
sudo apt-get install git
```

In most cases, you will get a message stating that this packet is already downloaded, in this case an attempt to re-install it does no harm. The `git` command being secured, the piHPSDR repository is downloaded, and support libraries are installed, with the commands

```
cd $HOME
git clone https://github.com/dl1ycf/pihpsdr
cd pihpsdr
LINUX/libinstall.sh
```

—— Release and Development version ——

Proceeding as shown above will provide an up-to-date ,,snap shot" of the piHPSDR repository. This ensures that you have the latest (development) version, but there is a chance that there is a recently introduced bug which has not yet been found and fixed.
There is also a ,,release" version which is the previous version. The ,,release" version lags behind the current development since new features are not included there. This means that any ,,release" version is stable at least from the end user's view point (bug fixes are still occasionally made). To get the release version, you must, after typing in the the command `cd pihpsdr`, insert the command

```
git checkout Rel-2.6
```

and this will then switch your local repository to the release version 2.6. No more details on how to switch between versions with the `git` command can be given in this manual, since these are standard procedures with git.

Depending on the internet speed (and the speed of your SD card or hard disk), the first and the last command my take some time. The first command loads the complete piHPSDR repository from my GitHub account. The last command (`libinstall.sh`) contains all the magic, it not only loads

all required RaspberryPi OS packages, but also loads and installs further libraries piHPSDR depends on, including the core SoapySDR library. The libinstall procedure also does some other things for you, including creating a desktop icon for piHPSDR. If you are on a Desktop PC running LINUX, you will get error messages about libraries such as libgpiod that are missing in the repository, which you can ignore.

It is clear that double-clicking the piHPSDR desktop icon at this stage leads nowhere, since first you need to compile the program. However, this is simply done by the two commands

```
cd $HOME/pihpsdr
make
```

and that's it! The first command is even not necessary (since you are in the pihpsdr directory at that point). But if you make changes (e.g. changing the compile time options as described in Appendix G, or apply some code modifications) and want to re-compile later, this sequence is the safest way to create a new binary, namely go to the `pihpsdr` directory and recompile by executing "make".

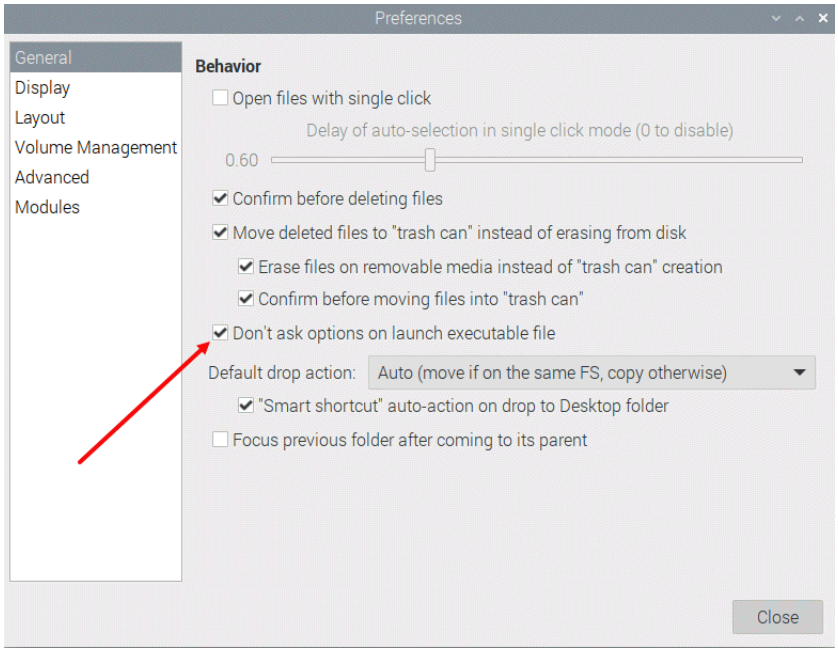

Fig. J.1: Suppress being asked options when clicking the piHPSDR desktop icon.

At this stage, double-clicking the piHPSDR desktop icon should start the program. With the default settings of the RaspPi desktop, normally a window pops up that asks you whether you want to start the program directly (of course you want). To get rid of this question once and forever, open the file manager and go to `Edit`→`Preferences` (Fig. J.1). Activate the check box where it is indicated by the red arrow and that's it. Immediately after running the install script, the piHPSDR desktop icon looks quite generic and does not show the HPSDR logo. Log out and then log in again and the logo is there. With the Ubuntu operating system, double-clicking the piHPSDR Desktop icon also usually fails, and this can be cured by right-clicking the icon and selecting the `Allow Launching` tab.

Proceeding as described above, you just compiled the program with the default compile-time options (see Appendix G), namely `MIDI`, `GPIO`, and `SATURN`, and with `PulseAudio` as audio module. If you want, for example, use SoapySDR hardware such as the AdalmPluto or RTL sticks, you have to enable `SOAPYSDR` and recompile as described in Appendix G.

J.2 piHPSDR and SoapySDR

This section is only relevant for those who want to use piHPSDR with SoapySDR radios such as the AdalmPluto, the LIME-SDR, RTL sticks, etc. All others are already done at this point.

The SoapySDR library, and all SoapySDR modules, must be compiled from the sources. The reason is, that SoapySDR in the standard repositories is quite outdated (version 0.7) while piHPSDR requires a more recent version (0.8) because the SoapySDR API has changed from 0.7 to 0.8. The necessity to compile SoapySDR from the sources may be dropped in the future, if the libraries in all the standard repositories (Debian, Ubuntu, etc.) are all updated to the 0.8 API.

The SoapySDR core library has already been downloaded, compiled, and installed with the `LINUX/libinstall.sh` command above. To verify correct installation of the SoapySDR core, use the command (note the capitalization!)

```
SoapySDRUtil -info
```

On my test system, this command produced the following output:

```
######################################################
##     Soapy SDR -- the SDR abstraction library     ##
######################################################

Lib Version: v0.8.1-gc4340a2b
API Version: v0.8.200
ABI Version: v0.8-3
Install root: /usr/local
Search path:  /usr/local/lib/SoapySDR/modules0.8-3 (missing)
No modules found!
Available factories... No factories found!
Available converters...
 -  CF32 -> [CF32, CS16, CS8, CU16, CU8]
 -  CS16 -> [CF32, CS16, CS8, CU16, CU8]
 -  CS32 -> [CS32]
 -   CS8 -> [CF32, CS16, CS8, CU16, CU8]
 -  CU16 -> [CF32, CS16, CS8]
 -   CU8 -> [CF32, CS16, CS8]
 -   F32 -> [F32, S16, S8, U16, U8]
 -   S16 -> [F32, S16, S8, U16, U8]
 -   S32 -> [S32]
 -    S8 -> [F32, S16, S8, U16, U8]
 -   U16 -> [F32, S16, S8]
 -    U8 -> [F32, S16, S8]
```

(Sorry for the tiny font, I did not want to wrap output lines.) Note the Lib/API/ABI version (0.8). If you see version 0.7 here then you probably have installed SoapySDR on your computer from the standard repositories, and this is not going to work. The `libinstall.sh` command has been designed to work (and has been tested on) a plain vanilla operating system, not on a "spoiled" one where the user has manually installed additional software packages that may induce incompatibilities. The best way out of such a problem is to re-install Linux (on a RaspPi, simply use a new SD card to do so).

In addition to the SoapySDR core, you need a SoapySDR module for each specific radio you want to work with. The example just given clearly states that no such modules are yet installed. Support for specific radios is available in the `LINUX` directory in form of installation scripts, these have the name `soapy.`*your-radio*`.sh`. To give an example, in order to install the SoapySDR module for an AdalmPluto, you have to use the commands

```
cd $HOME/pihpsdr
LINUX/soapy.pluto.sh
```

(you may be asked the administrator password at the end of the process, when it comes to actually install the SoapySDR module just compiled). To veryfy that this actually worked, repeat the command `SoapySDRUtil -info`, and in the output you should see that module for the Pluto is now available

```
Search path:  /usr/local/lib/SoapySDR/modules0.8-3
Module found: /usr/local/lib/SoapySDR/modules0.8-3/libPlutoSDRSupport.so (0.2.2-cdce239)
Available factories... plutosdr
```

At the moment, there is support for the Adalm Pluto, for RTL sticks, for AirSpy radios, and for the SDRplay. I have put together and tested scripts for the Pluto and for RTL sticks, since I have the hardware and can test it. The other scripts that install support modules for SoapySDR radios have been contributed by piHPSDR users who have those radios and managed to compose an automatic install script for the required SoapySDR modules. Note that I received a report that the SoapySDR module for SDRplay does not implement the Soapy API correctly, for example when setting the overall RF gain. I cannot confirm this since I do not own such hardware, but be warned and do not blame this on piHPSDR if such problems are encountered.

To verify that SoapySDR can actually "talk" with your radio, connect it to the computer, wait a few seconds, and use the command

`SoapySDRUtil -find`

An sample output that I obtained on my system reads

```
#######################################################
##     Soapy SDR -- the SDR abstraction library     ##
#######################################################

Found device 0
  device = PlutoSDR
  driver = plutosdr
  label = PlutoSDR #0 ip:pluto.local
  uri = ip:pluto.local
```

and this shows that the SoapySDR library has recognised the Pluto. There is no way for piHPSDR to connect to a SoapySDR device that has not been recognised by SoapySDR. If such a problem arises, this stems from missing, corrupt, or outdated SoapySDR libraries which are not part of piHPSDR. I have seen several cases where people have installed SoapySDR stuff from the distro (via `apt-get install`) and then chaos must ensue. The best way to recover from such a situation is to burn a ,,virgin" OS onto the SD card and start from scratch.

If the SoapySDR library has been correctly installed, piHPSDR has to be compiled with the SOAPYSDR compile-time option enabled (see Sec. G). This can easily be achieved with the commands

```
cd $HOME/pihpsdr
make clean
echo "SOAPYSDR=ON" > make.config.pihpsdr
make
```

If you want to know what's behind these commands, read Sec. G.

J.3 Upgrading an installation

piHPSDR is a "living" program, which means that regularly error corrections are made, and new features are introduced. After the internet repository has been updated, your installation will (of course) not change until you perform specific actions as described in this section. But before documenting how to do this, some general principles should be made clear:

- If an error is reported to me and if I could correct this, this will be corrected both in the master branch (the current version) and in the most recent release branch.
- New features will only be included in the master branch, while the release branch is "feature frozen". This means it should normally not be necessary to update the manual in the release branch.
- In the following, it will both be explained how to upgrade your current branch, or how to switch to the latest version of another branch (e.g. to move from a release version to the current version).

Which branches do exist?

The following commands simply show which branches do exist.

```
cd $HOME/pihpsdr
git pull
git branch -a
```

On my computer the output is

```
  Rel-2.3
  Rel-2.4
  Rel-2.5
  Rel-2.6
* master
  ...
```

You can ignore lines starting with `remotes/`. The first lines display the names of all existing branches, and the line starting with a star indicates the branch you are currently on. Older branches up to `Rel-2.4` are meanwhile outdated, the previous release branch is named `Rel-2.5` and the current one, at the time of this writing, is `Rel-2.6`.

Which version do I have?

To be sure which version/branch is actually installed on your compute, open a terminal window and type in the commands

```
cd $HOME/pihpsdr
git status
```

The output of the last commands tells you which branch your are using. If the output starts with

```
On branch master
Your branch is up to date with 'origin/master'.
```

then you are using the "current version" (that is, the master branch). If you are using a release version, then the output reads (for the v2.6 release)

```
On branch Rel-2.6
Your branch is up to date with 'origin/Rel-2.6'.
```

which tells you you are on the Rel-2.6 branch. Note that if you committed local modifications, your branch will not be up to date with the repository as indicated in the outputs shown above, and in this case this whole section does not apply to you. But if you have committed local modifications, this usually means you are an expert and know what to do.

Standard update procedure

The standard update procedure will bring piHPSDR on your computer to the latest version of that specific branch (either the master branch or a release branch). Just use the following commands in a terminal window

```
cd $HOME/pihpsdr
git pull
make clean
LINUX/libinstall.sh
make
```

Note the command `libinstall.sh` will take some time and is not necessary in most cases. But occasionally, piHPSDR upgrades require new libraries to be installed and `libinstall.sh` takes care of that.

Switching to the master branch

No matter what your starting point is, the following commands will bring you to the latest "current" version, that is, the latest version of the master branch:

```
cd $HOME/pihpsdr
git checkout master
git pull
make clean
LINUX/libinstall.sh
make
```

Switching to a release branch

No matter whether your starting point is the master branch or another release branch, the following commands will bring you to the v2.6 release branch:

```
cd $HOME/pihpsdr
git checkout Rel-2.6
git pull
make clean
LINUX/libinstall.sh
make
```

Appendix K

Installation of piHPSDR from the sources (MacOS)

This procedure has been tested on several Macintosh models, including a MacBook Air with an M1 Apple Silicon CPU, a MacMini with an M2 pro CPU, and a somewhat older MacMini with an Intel CPU. In the last years, MacOS operating systems from version 10.15 (,,Catalina") through 26.0 (,,Tahoe") have been used. Very old MacOS systems (e.g. Catalina) are no longer supported by HomeBrew. Installing piHPSDR on these is still possible but requires (really) advanced skills, so take care to use a Mac with an operating system still supported by HomeBrew.

If you want to run piHPSDR on an Apple Macintosh computer (iMac, Mac Mini, iBook Air, PowerBook), then you *have to* compile piHPSDR from the sources. Producing a pre-compiled binary that can be distributed and run simply by double-clicking is overly complicated for applications using the GTK graphical user interface (as piHPSDR does). But fear not, the procedure is actually very simple because all the difficult things are avoided by using the HomeBrew toolkit.

A note on administrator privileges

The whole installation procedure depends on that your user account is an administrator account, such that you can execute ,,sudo" commands in the terminal window. You may be asked the administrator password. If your user account does not qualify for administrator work, then you are simply not allowed to install the ,,HomeBrew" universe which is the prerequisite for compiling piHPSDR. In normal cases (you own the Mac) you should have administrator privileges.

K.1 Fresh install from the internet

Some users have reported that it seems to be necessary to first install the X Window manager on the Macintosh. While I cannot confirm this, it also does no harm if you install X-Windows on your Mac. To the end, in a web browser open the link

`www.xquartz.org`

and install the most recent package file found there. At the time of this writing, this is version 2.8.5 released in January 26, 2023. The package is good both for Intel and Silicon Macs. Download the package file (e.g. XQuartz-2.8.5.pkg) to your desktop and run it by double-clicking.

Compiling piHPSDR on a Mac requires some basic LINUX/Unix skills. The most important program to use is the Terminal application (`Terminal.app`), that can be found in the Utilities folder that resides in the Applications folder. It is suggested to drag this application into the ,,dock" so you have quick access.

On a plain vanilla MacOS, even the most basic commands for doing programming are not installed by default. For example, the `git` command used below as the first step to download piHPSDR is not present by default. But this is easily cured. Simply open a terminal window and type in the command

```
xcode-select --install
```

(you will need internet access to perform this task). This installs the so-called Command Line Tools, which we absolutely need to proceed. But even

with the Command Line Tools installed, additional commands and libraries are needed. Fortunately, there exist packages that allow for easy installation of such additional libraries. The most popular such ,,Unix enabler" packages are `MacPorts` and `HomeBrew`, I am using `HomeBrew` but I know that piHPSDR can be compiled and run successfully with `MacPorts` as well. If you use `MacPorts` then you are on your own, or have to rely on piHPSDR installations done by the `MacPorts` folks. For `HomeBrew`, I provide semi-automatic installation scripts that do all the complicated stuff for you. The choice, `HomeBrew` vs. `MacPorts`, is up to you, but my recommendation is obviously to go for `HomeBrew`.

To load all required libraries and commands, open a terminal window. If you do so, you get a prompt and are in your home directory. It is assumed that no directory named `pihpsdr` exists. If it does, then this is likely a left-over from some previous attempt to install the program and then it should be moved elsewhere, since it may contain files with saved preferences (`*.props`) which you may want to re-use.

The piHPSDR repository is downloaded, and support libraries are installed, with the commands

```
cd $HOME
git clone https://github.com/dl1ycf/pihpsdr
cd pihpsdr
MacOS/libinstall.sh
```

The `git clone` command loads the piHPSDR repository, and with the command `MacOS/libinstall.sh` (which is then executed within the just-created `pihpsdr` directory) installs the `HomeBrew` universe. Installing `HomeBrew` starts with asking you for the administrator password, tells you what it wants to do and requires hitting `Enter` to proceed. Do not worry if `HomeBrew` was already installed on your computer, the installation procedure works in this case as well and does no harm. In addition to the `HomeBrew` core, additional libraries that piHPSDR depends on (e.g. GTK+3), the SoapySDR core and SoapySDR modules for several radios are installed.

In case something went wrong, or simply to check that `HomeBrew` has been correctly installed, open a *new* terminal window and type the command

```
brew config
```

—— Release and Development version ——

Proceeding as shown above will provide an up-to-date ,,snap shot" of the piHPSDR repository. This ensures that you have the latest (development) version which corresponds to the present version of the manual, but there is a chance that there is a recently introduced bug which has not yet been found and fixed.
There is also a ,,release" version which is the previous version. The ,,release" version lags behind the current development since the very latest features are not included there. This means that any ,,release" version is stable at least from the end user's view point (bug fixes are still occasionally made). To get the release version, you must, after typing in the the command `cd pihpsdr`, insert the command

```
git checkout Rel-2.6
```

and this will then switch your local repository to the release version 2.6. No more details on how to switch between version with the `git` command can be given in this manual.

This should give you lots of information on the currently installed version of `HomeBrew`, but also on the CPU of your machine, the compilers and MacOS version info, etc. If the command fails stating that the "brew" command has not been found, something is wrong, since the automatic installation procedure should have taken care to update your shell profiles such that the "brew" command is found. This is the reason why you have to open a new terminal window to make this test, since in the originally opened window the update of the command search path is not yet effective.

If anything went wrong with the SoapySDR installation, you can safely ignore this is you do not plan to compile piHPSDR with the SOAPYSDR option. You also do not need to manually disable the GPIO option since on MacOS, the GPIO and SATURN options do not apply and are automatically deactivated. Without changing the compile time options (see Appendix G) the MIDI option is the only one you get on MacOS. Changing compile-time options is described in detail in Appendix G and involves the creation of a file `make.config.pihpsdr` within the `pihpsdr` directory.

Compiling piHPSDR then only requires two commands

```
cd $HOME/pihpsdr
make app
```

and that's it! The first command is even not necessary at this point, since you are already in the `pihpsdr` directory. But if you make changes in the future (e.g. changing the compile time options as described in Appendix G, or apply some code modifications) and want to re-compile later, this sequence of commands is the safest way to create a new binary.

Note that saying `make app` instead of just saying `make` has the bonus that a MacOS ,,app" bundle is automatically created within the piHPSDR directory. You can drag this bundle in the Finder, say, from the piHPSDR directory in your home directory, to the Desktop, this can also be accomplished with the additional command

```
mv pihpsdr.app $HOME/Desktop
```

(take care that any older piHPSDR application on the Desktop is moved or deleted first).

Checking the SoapySDR installation.

If you want to use SoapySDR, please check the Soapy installation. Open a new terminal window and simply type the command (be careful to use the correct capitalisation!)

```
SoapySDRUtil -info
```

On my test system, this command produced the following output:

```
######################################################
##     Soapy SDR -- the SDR abstraction library     ##
######################################################

Lib Version: v0.8.1-release
API Version: v0.8.0
ABI Version: v0.8
Install root: /usr/local
Search path:  /usr/local/lib/SoapySDR/modules0.8
Module found: /usr/local/lib/SoapySDR/modules0.8/libHackRFSupport.so   (0.3.3)
Module found: /usr/local/lib/SoapySDR/modules0.8/libLMS7Support.so     (20.10.0)
Module found: /usr/local/lib/SoapySDR/modules0.8/libPlutoSDRSupport.so (0.2.1)
Module found: /usr/local/lib/SoapySDR/modules0.8/libRedPitaya.so       (0.1.1)
Module found: /usr/local/lib/SoapySDR/modules0.8/libairspySupport.so   (0.2.0)
Module found: /usr/local/lib/SoapySDR/modules0.8/libairspyhfSupport.so (0.2.0)
Module found: /usr/local/lib/SoapySDR/modules0.8/librtlsdrSupport.so   (0.3.2)
Available factories... airspy, airspyhf, hackrf, lime, plutosdr, redpitaya, rtlsdr
Available converters...
 -  CF32 -> [CF32, CS16, CS8, CU16, CU8]
 -  CS16 -> [CF32, CS16, CS8, CU16, CU8]
 -  CS32 -> [CS32]
 -   CS8 -> [CF32, CS16, CS8, CU16, CU8]
 -  CU16 -> [CF32, CS16, CS8]
 -   CU8 -> [CF32, CS16, CS8]
 -   F32 -> [F32, S16, S8, U16, U8]
 -   S16 -> [F32, S16, S8, U16, U8]
 -   S32 -> [S32]
 -    S8 -> [F32, S16, S8, U16, U8]
 -   U16 -> [F32, S16, S8]
 -    U8 -> [F32, S16, S8]
```

In contrast to the RaspPi installation, SoapySDR is already at version 0.8 in the `HomeBrew` repository so the modules can very simply be installed and need not be compiled from the sources. I have added (as you can see) support for quite a few SoapySDR radios, and most likely even more SoapySDR modules are available in the `HomeBrew` repository.

To verify that a given hardware (e.g. an RTL stick, or an AdalmPluto) is properly detected by the SoapySDR library, type in the command

```
SoapySDRUtil -find
```

On my Macintosh (with an Adalm Pluto attached), the output is

```
#######################################################
##     Soapy SDR -- the SDR abstraction library     ##
#######################################################

WARNING: Unknown parameter '0' in <context>
WARNING: Unknown parameter '23' in <context>
WARNING: Unknown parameter 'v0.23' in <context>
Found device 0
  device = plutosdr
  driver = plutosdr
  uri = usb:20.14.5
```

and this shows that the SoapySDR library has recognised the Pluto. There is no way for piHPSDR to connect to a SoapySDR device that has not been recognised by the SoapySDR library. If such a problem arises, this stems from missing, corrupt, or outdated SoapySDR libraries which are not part of piHPSDR.

K.2 Upgrading an installation

piHPSDR is a "living" program, which means that regularly error corrections are made, and new features are introduced. After the internet repository has been updated, your installation will (of course) not change until you perform specific actions as described in this section. But before documenting how to do this, some general principles should be made clear:

- If an error is found, it will be corrected both in the master branch (the current version) and in the most recent release branch.
- New features will only be included in the master branch, while the release branch is "feature frozen".

- In the following, it will both be explained how to upgrade your current branch, or how to switch to the lastest version of another branch (e.g. to move from a release version to the current version).

Which branches do exist?

The following commands simply show which branches do exist.

```
cd $HOME/pihpsdr
git pull
git branch -a
```

On my computer the output is

```
  Rel-2.3
  Rel-2.4
  Rel-2.5
  Rel-2.6
* master
  remotes/origin/HEAD -> origin/master
  remotes/origin/Rel-2.3
  remotes/origin/Rel-2.4
  remotes/origin/Rel-2.5
  remotes/origin/master
```

You can ignore the lines starting with `remotes`. The other lines display the names of all existing branches, and the line starting with a star indicates the branch you are currently on. Older branches up to `Rel-2.4` are meanwhile outdated, the previous release branch is named `Rel-2.5` and the current one, at the time of this writing, is `Rel-2.6`.

Which version do I have?

To be sure which version/branch is actually installed on your compute, open a terminal window and type in the commands

```
cd $HOME/pihpsdr
git status
```

The output of the last commands tells you which branch your are using. If the output starts with

```
On branch master
Your branch is up to date with 'origin/master'.
```

then you are using the "current version" (that is, the master branch). If you are using a release version, then the output reads (for the v2.6 release)

```
On branch Rel-2.6
Your branch is up to date with 'origin/Rel-2.6'.
```

which tells you you are on the Rel-2.6 branch. Note that if you committed local modifications, your branch will not be up to date with the repository as indicated in the outputs shown above, and in this case this whole section does not apply to you. But if you have committed local modifications, this usually means you are an expert and know what to do.

Standard update procedure

The standard update procedure will bring piHPSDR on your computer to the latest version of that spefic branch (either the master "current" branch or a release branch). Just use the following commands in a terminal window

```
cd $HOME/pihpsdr
git pull
make clean
MacOS/libinstall.sh
make
```

Note the command *libinstall.sh* will take some time and is not necessary in most cases. But occasionally, piHPSDR upgrades require additional libraries to be installed and the commands shown here take care of that.

Switching to the master branch

No matter what your starting point is, the following commands will bring you to the lastest "current" version, that is, the lastest version of the master branch:

```
cd $HOME/pihpsdr
git checkout master
git pull
```

```
make clean
MacOS/libinstall.sh
make
```

Note the command `libinstall.sh` will take some time and is not necessary in most cases. But occasionally, piHPSDR upgrades require new libraries to be installed and `libinstall.sh` takes care of that.

Switching to a release branch

No matter whether your starting point is the master branch or another release branch, the following commmands will bring you to the v2.6 release branch:

```
cd $HOME/pihpsdr
git checkout Rel-2.6
git pull
make clean
MacOS/libinstall.sh
make
```

Appendix L

Connecting Computer and Radio

This section applies if you run piHPSDR on a RaspPi or a Desktop or Laptop computer running Linux or MacOS, and if this computer connects to the radio via ethernet, and the radio is running HPSDR protocol-1 or protocol-2. This class of radios encompasses the original HPSDR radios but also the HermesLite-II (which runs a variant of the HPSDR protocol-1) and RedPitayas (if it runs a HPSDR radio emulator).

This section also does not apply to SoapySDR radios (Adalm Pluto, RTL sticks, etc.). In most cases they use an USB connection so you can just plug and play.

This section further does ***not*** *apply to the new Saturn/G2 radios when running piHPSDR on the internal RaspPi compute module. In this case, the internal RaspPi is directly (and automatically) connected to the radio via an XDMA interface.*

The easiest way: DHCP-based connection.
Every ethernet interface needs an IP address assigned to, otherwise it will not be operative. This applies both to computers and radios. The standard way of assigning an IP address to an interface is, that a so-called DHCP server is running ,,at the other side of the cable". Then the computer or the radio requests an IP address and gets it from that server. For example, if you connect both the computer (RaspPi) and the radio to an internet router via

cable, then the router (assuming that it runs a DHCP server) will provide an IP address to both devices and they can start to communicate. This is not only the easiest way to set up the communication between computer and radio, but also lets the computer connect to the internet (if there is upstream internet connection ,,at the other side" of the router). If you just want to connect computer and radio (e.g. because the computer has a separate WiFi connection to the internet), just get a WiFi router without using its antenna and its upstream port. These low-cost devices usually have four downstream ports, into two of which you plug the cable from the computer and the cable from the radio.

The most stable way: direct cable connection.
A setup that works very well but is more difficult to set up, is to have a *direct cable connection* between the RaspPi and the radio. Then you only need a single cable, where one end is plugged into the computer and the other into the radio. The computer can then still be connected to the web via its WiFi interface. In this direct cable connection, neither the computer nor the radio gets an IP address for the wired interface simply because there is no DHCP server at either side of the cable.

To make the communication work, both the computer and the radio must have a fixed IP address in the same subnet. Choose two different addresses for the computer and the radio. I usually use IP addresses of the form 192.168.8.xx, where xx is a number between 10 and 30. This choice avoids conflicts with automatic DHCP addresses e.g. used for the WiFi connection of the computer. The subnet mask for such addresses (in case you are asked) is 255.255.255.0.

The bad news is that the procedure of assigning a fixed IP address differs greatly between different radio, and different operating systems (e.g. Linux and MacOS). For HPSDR radios, a fixed IP address can most easily be set with a so-called bootloader program. For RedPitaya radios, this can often be done via the web interface of the RedPitaya. For the HermesLite-II radio, the SparkSDR software is recommended to set a fixed IP address. There is a chicken-and-egg problem here: some of these programs want to ,,discover" the radio first before doing anything, and this means you are back to DCHP-supported connection at least for burning a fixed IP address into the radio.

Information on how to assign a fixed IP address to a wired ethernet interface for Linux computers is relatively easy to locate on the internet. For MacOS

it is even easier since it is an option that can be chosen in the network setup tool.

If you plan to run your own DHCP server.
In principle, one can have a DHCP server running on the computer that also runs piHPSDR, and this will then assign an IP address to the wired interface of the computer and to the radio. Information how to do this can relatively easy be found on the internet. For MacOS, this works very well for me, but for my RaspPi, it is somewhat involved (the DHCP server has to be run on a ,,bridge" device) and this is beyond the scope of this manual.

—— End of the piHPSDR manual ——