A Tiny *Python* Panadapter

Put one of the popular and inexpensive BeagleBone or Raspberry Pi microcomputers to work in your station.

Martin Ewing, AA6E

This digital panadapter[1] software project highlights the important role of programming in modern Amateur Radio work. I used software defined radio (SDR) and digital signal processing (DSP) methods to transform an I/Q (in-phase/quadrature) audio IF to a live spectrum and waterfall. To support these functions, I needed a graphical programming environment, an appropriate programming language, and support libraries. I also needed an appropriate audio capture system. And finally, I wanted to make this run on an inexpensive computer running *Linux*.

This project grew out of the desire to make a panadapter display for the Elecraft KX3 transceiver. The KX3 provides I/Q (in-phase/quadrature) IF outputs for just this purpose. Common computer sound cards will also support 48 kHz sampling, allowing display of up to ±24 kHz around the current RF tuning frequency. Many SDR radios will provide I/Q audio streams for the panadapter, such as the SoftRock[2], FiFiSDR[3], and others.

My aim was to develop software in a high-level, easy-to-learn language (*Python*) to run on small *Linux* microcomputer boards, such as the BeagleBone Black[4] or the Raspberry Pi[5], as well as on any common *Linux* PC. I hope this project will be a way for more hams to learn more about how to use and program these powerful devices.

The common way to input the I/Q IF (up to 48 or 96 kHz band pass) is through a stereo sound card, which is built into many PCs. If a built-in device is not available, an inexpensive USB sound card can be used. Displays range from the typical full-size computer monitor down to cell-phone style screens, such as BeagleBone's optional 4-inch LCD.

The code is intended to be easily adapted to new applications or hardware. Distributed as *Python* source code, you can think of it as much as a development platform as a polished "plug and play" application. It is not highly "tuned" for any particular configuration, making it a useful starting point for learning and developing your new projects.

This software will also support the RTL-SDR[6] "dongle" that is a popular way to get VHF/UHF input for software defined radio.

System Overview

Like many radios these days, you can appreciate a system for its hardware or its software. And more and more of the action is in the software!

Hardware View

The panadapter hardware is straightforward. There are only a few major components — either a stereo sound card or an RTL-SDR dongle, a computer, and a display (see Figure 1). You have the option to attach a keyboard, mouse, and Ethernet connection to make up a complete *Linux* workstation. You can also connect to a receiver CAT (Computer Assisted Transceiver) port for control functions.

A wide range of hardware can provide these functions. An ordinary desktop or laptop PC that runs *Linux* will do fine, and the sound card may be built in. However, it is more interesting to use small *Linux* boards, like the Raspberry Pi or BeagleBone. For these, a USB sound card is likely to be the best choice, and the display can be a 3-, 4-, or 7-inch LCD screen. Some control func-

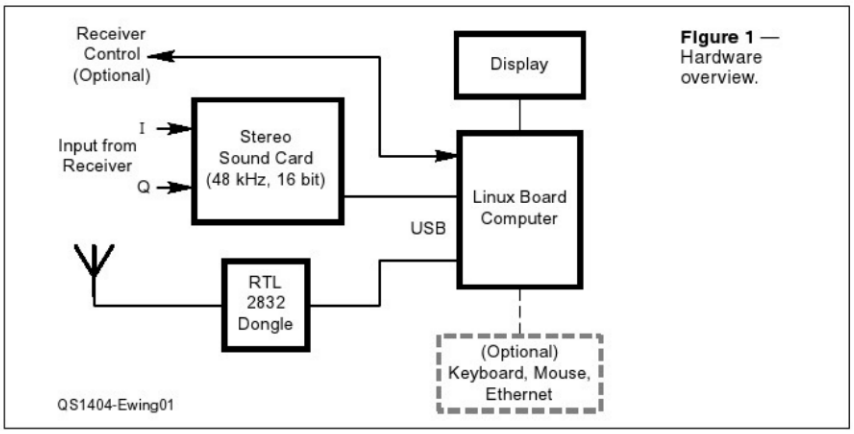


Figure 1 — Hardware overview.

[1]Notes appear on page 38.

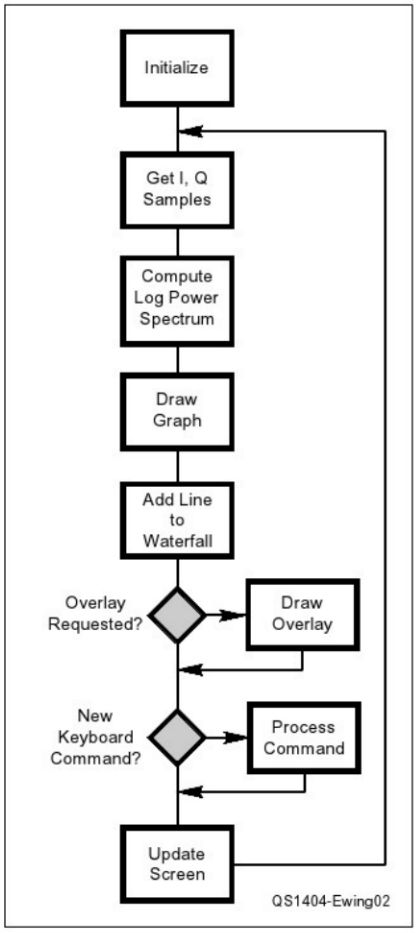


Figure 2 — Software flow chart.

tions are necessary, but we can implement them with dedicated buttons without a keyboard or mouse. Some screens support a touch interface, but I have not implemented touch in the software.

The main challenge of this project is the software, especially when we are aiming for fast audio and graphics features running in as small a processor as possible.

Software View

Figure 2 shows a simplified block diagram of the top level code of the panadapter, called *iq.py*.

After initialization, the program gets a chunk of I and Q IF (audio) data. Typically, this comes from PortAudio calls (and call-back routines) that accept data from an ALSA audio device, such as a USB sound card. PortAudio uses non-blocking I/O so that no data is lost during processing and display, as long as there is adequate processor power.

Alternatively, we can get data from an RTL-SDR VHF/UHF dongle device. This dongle can sample at high rates (up to 2.048 MHz), beyond the real-time capability of smaller systems. For this reason, the RTL mode uses blocking I/O and accepts some loss of data.

Data in hand, we compute the log power spectrum[7] of each buffer in the chunk and accumulate it into the chunk's average log power spectrum — the thing we will display. A power vs. frequency ("2D") graph is always plotted along with a calibrated oscilloscope-like "graticule." Optionally, a waterfall display (power vs frequency vs time, or "3D" plot) is also produced.

If requested, an overlay of diagnostic information and user help is drawn on top of the data displays in a way that works well on small display devices. Several pages of help are available that guide the user to set parameters using four pushbuttons. (I have optimized the process for the BeagleBone's 4-inch LCD4 display card, but it should work on a range of processors and displays.)

If there is a keyboard event (or pushbutton press), it is interpreted as a command to change operating or display parameters.

Finally, the newly constructed graphic is transmitted to the display, and the loop continues.

Software Libraries

The software overview in Figure 3 shows the major code blocks and how they layer on each other and add functionality. These are the resources we will use in the *iq.py* program.

The bottom layer is the *Linux* operating system, which supports the audio, graphics, numerics, and control functions. Each component may be built on simpler functions that result in the desired capabilities. For example, for the audio functions we use *PyAudio* (**people.csail.mit.edu/hubert/pyaudio/**), which is a *Python* form of the *C/C++ Portaudio* library (**www.portaudio.com**), which in turn is based on ALSA, the Advanced *Linux* Sound Architecture.

Graphics is an important but computer-intensive panadapter function. I chose the *PyGame* library (**www.PyGame.org**), which is built on SDL (Simple DirectMedia Library). SDL in turn calls on Xorg or directfb for the most basic screen operations.

QS1404-Ewing03

Panadapter Application, *iq.py*			
Audio	Graphics	Numerics	Rx Control
PyAudio	PyGame	Numpy	Hamlib
Portaudio	SDL		
ALSA	Xorg/directfb		
Linux			

Figure 3 — Software libraries overview.

PyGame is a relatively simple graphics system that is easy to program and reasonably efficient.

The numerical parts of my code are facilitated by the *Numpy* library (**Numpy.org**). This is a large system for scientific and engineering mathematics. I only need to use *Numpy*'s Fast Fourier Transforms (FFTs) and some other efficient array operations.

Finally, the *Hamlib* (**www.Hamlib.org**) library provides my program with the ability to address and control many different types of amateur radios.

Building the Operating System

In this section, we'll see what needs to be done to bring your brand new computer board into a state that is ready for our *Python* application.

This project is meant to run on various hardware platforms, but especially on the Raspberry Pi (Pi) and the BeagleBone Black (BBB). Other small boards, like the older BeagleBone and BeagleBoard, should be okay, and the code should run even better on modern *Linux* PCs.

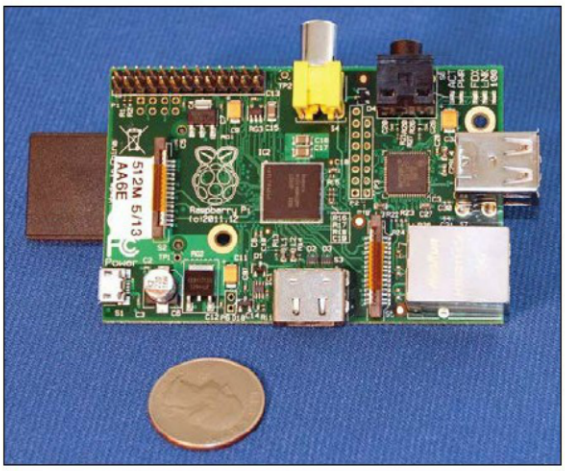
While all these small platforms are *Linux*-based, the main software components — *PyAudio*, *PyGame*, *Numpy*, *Hamlib*, and the *Python* language — are all available for *Windows* and *MacOS*. Porting this project to *Windows* or Macintosh would be a fine thing to do, but we are focusing on the *Linux* OS.

If you're considering getting a new computer for this project, I would recommend the BeagleBone Black or a card of equivalent power. The Raspberry Pi (512 MB Model B) is usable, but it requires more care to select operating modes that do not exceed its CPU capacity. On the other hand, the Pi's *Raspbian* operating system can be easier to prepare for the panadapter project, and the community of Pi users is very large.

I will now describe how to make your Pi or BBB ready for our application. Because the two boards' suppliers configure their systems differently, some preparations are specific to each platform, but others are common to both.

Raspberry Pi: First Steps

The Pi download website (**www.raspberrypi.org/downloads**) recommends that first-time Pi users install the "New Out of Box Software." If you follow this procedure, you will be given a choice of operating systems. Select the *Raspbian* option, and then sit back for a complete *Raspbian* download.



A Raspberry Pi Model B microcomputer.

If you're familiar with the *Linux*/*Debian* way of doing things, you can get more control and possibly save some time by installing *Raspbian* directly from the download page.

The first thing is to set up the Pi's operating system for the panadapter project. Many of these operations are useful for anything you may want to do with your Pi. Detailed step-by-step procedures are in the supplementary files in **www.arrl.org/qst-in-depth**. In summary:

- Download *Raspbian* onto an SD card using a PC. An 8 GB class 4 card or above is suggested. Place the card in your Pi, attach keyboard, monitor, Ethernet, and +5V power. Wait for boot-up.
- Set up a password and other options as desired in the *raspi-config* process. Set up localization for time zone, keyboard, etc.
- Reboot and go to "Continuing with your Pi or BBB" below.

BeagleBone Black: First Steps

The BeagleBone Black (BBB) requires a bit more work than the Pi to configure for our project, but in the end we will have a system that is configured much like the Pi above. I need to bring up a *Debian* 7.0.0 "Wheezy" (or later) *Linux* system on the BBB. The major steps are outlined here, while details are given the supplementary files mentioned above.

- Follow instructions at **eLinux.org/BeagleBoardDebian**, building a *Debian* system on an 8 or 16 GB microSD card using a convenient PC.
- Install the microSD card on the BBB. Attach an HDMI monitor, keyboard, and mouse. Attach an Ethernet connection to your local router.
- Install the LXDE desktop package from the network.

Continuing with Your Pi or BBB

Most of the following is common to both boards. Again, more detailed steps are provided in the supplementary files.

- Install the packages *python-pygame*, *python-hamlib2*, *python-dev*, *portaudio19-dev*, and *python-numpy* from your repository.
- Obtain and install the latest *pyAudio* package.
- If desired, obtain, build, and install *rtl-sdr* and *pyrtlsdr* for RTL-SDR support.
- Install optional components, set up user accounts, etc.

Linux PC

If you have an up-to-date *Linux* PC, you can follow the recipe for the BeagleBone, beginning with installation of *python-pygame*, etc. If you are using a distribution other than *Debian* (like *Fedora*), your packages may have different names, and you may need to install them with a different command.

Installing and Running the Program

Table 1 summarizes the suggested hardware to support the panadapter application.

Recent Raspberry Pis are likely to be the Model B with 512 MB RAM. You may be able to use the 256 MB model, but that has not been tested. The Pi offers an NTSC video output, but it will not provide the needed display resolution. Use the HDMI port if possible.

The BeagleBone Black with the LCD4 cape (a 4-inch LCD with five buttons that stack on the BBB) is a good target platform for a compact panadapter. I explored using the BeagleBone audio cape, but at this writing the audio cape is not fully compatible with the BBB and LCD4. Improved audio support may eventually let us have a self-contained BBB cape system. Meanwhile, you must use an external USB sound card.

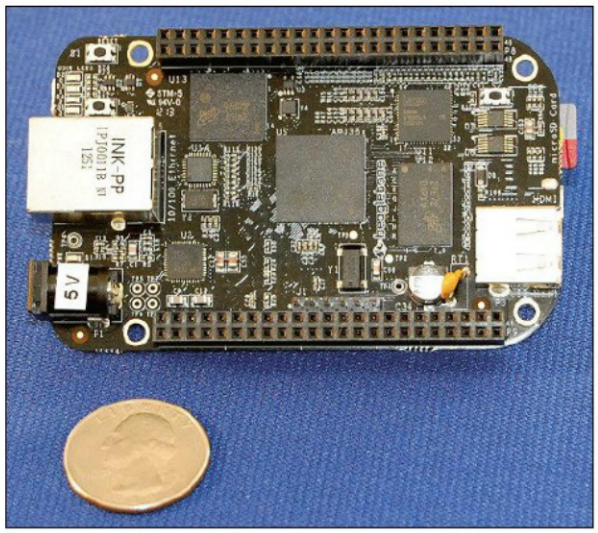
If you are connecting to a receiver that allows remote control (CAT), you may want to use the *Hamlib* support, which will require a USB radio connection or a USB-serial port adapter. If you are only using the RTL-SDR USB dongle receiver, you won't need a USB sound card.

Table 1 — Hardware Configurations

Device (Approx pricing)	Required	Desirable
Raspberry Pi ($35)	Raspberry Pi (Model B) 4 GB SD card Powered USB hub USB sound card HDMI monitor Keyboard and mouse	512 MB RAM 8 GB SD card USB-serial port (rig control) RTL-SDR dongle
BeagleBone Black ($45)	4 GB microSD card Powered USB Hub USB sound card LCD4 display or HDMI display Keyboard and mouse	8 GB microSD card USB-serial port (rig control) RTL-SDR dongle
Personal Computer	Recent *Linux* OS Built-in stereo input sound card or USB sound card	Traditional serial port or USB-serial port (rig control) RTL-SDR dongle

Some points to watch out for with any of the small *Linux* boards:

- Be sure that your +5 V supply has adequate current capacity and your USB cable is good. Especially when driving external USB devices that take power, you can easily get unreliable operation if the voltage droops too much. A 2 A supply or greater is a good bet.
- Use a powered USB hub if you have multiple USB devices. This minimizes the power drain from the CPU board.
- When it comes to powering down, it is preferable to use the *shutdown -h now* command (or equivalent GUI option) and then wait for activity to stop rather than simply switching off the power. If you don't do this (and many of us don't), there is some risk of data corruption.



A BeagleBone Black microcomputer.

Sound Cards

Most USB sound cards have a standard interface that works with the ALSA drivers found in typical *Linux* installations. Board-level plug-ins (with PCI or PCI Express connections) or built-in devices are also supported, and appear the same to higher-level software. It is rare that special drivers are needed, unless you have a complex sound card that includes special processing and mixing options.

For this project, we have evaluated a number of stereo sound cards at different price levels. The good news is that all of them seem to work. The bad news is that you get what you pay for. The less expensive units will not go above 48 kHz sampling and may offer poorer frequency response, less sensitivity, or more noise. Some offer phono or microphone preamplification, and some do not.

The sound cards we have looked at are summarized in Table 2. There are many others on the market, so this is just a sampling.

It is difficult to compare published specs on sound cards, especially the cheaper ones, because usually there are no published specs! *QST* reviewed sound cards in 2007.[9] Those products are probably unavailable now, and there were no USB sound cards considered. Nevertheless, the article is still a useful review of the complications of evaluating sound cards. The test procedures evaluated cards as "audio" devices as if for human listening, using a logarithmic frequency scale. For SDR, we want a flat response nearly up to the Nyquist frequency (½ the sampling rate) and we must think in terms of linear frequency scales, where the last kHz of response is as important as the first.

I do not have precise measurements for sound cards, although there is some relevant data on the Internet.[10] The less expensive options are certainly usable for casual work as a panadapter, but you can expect high-end sound cards to perform better for software defined radio where the best image rejection and dynamic range is desired.

The iConnex and UCA202 both appear to use the TI/Burr-Brown PCM2902 chip, which has an interesting quirk. In at least some versions, the left channel is delayed by exactly one sample relative the right channel. I have provided the "*—LAGFIX*" option in *iq.py* to correct for this "feature" (bug!), which would otherwise make these sound cards unusable for SDR work.

To determine the suitable operating levels for your sound card, in general, you'll want to adjust receiver gain (or sound card input gain) to a level just enough that you get reasonable "counts" on receiver noise, but no more. This will allow maximum dynamic range (headroom) for receiving. In practice, if your typical noise level sample values peak at ~ ±10 counts (out of full scale ±32,767 in a 16 bit converter) that should be enough. (Be sure your noise is coming from your antenna, not your electronics! Verify that the noise goes down when you disconnect the antenna.)

If you need more gain, you may be tempted to use the sound card's preamplifier, if it has one. This may or may not help. If you have a "phono" preamp, it probably applies RIAA equalization to match a phonograph's frequency response. That will attenuate high frequencies severely (by ~20 dB at 20 kHz). A "mic" preamp should not have this problem. Indeed, the iMic's mic preamp seems to improve the overall frequency response of the iMic.

Depending on your receiver and its grounding arrangements, it may be helpful to use audio isolation transformers for the I and

Table 2 — Some Inexpensive USB Sound Cards with Stereo Inputs

Unit	Brief Specs	Approx Pricing	Comments
iKey-Audio iConnex	48 kHz, phono preamp	$40	PCM290x based (may require lag correction)
Behringer UCA202	48 kHz, no preamp	$30	Similar to iConnex without preamp
Griffin iMic	48 kHz, mic preamp	$29	Spurious peaks at 5 kHz and multiples on BBB. 114 dB SNR claimed

Q signals between the radio and the sound card. A convenient commercial unit is the "Cables to Go 40000 3.5mm Extension Stereo Audio Isolation Transformer," which can be found by Internet search. Alternatively, you can wire your own if you have a 1:1 audio transformer.

The RTL-SDR Dongle

As a byproduct of the DVB-T (digital video broadcast — terrestrial) service in use in many countries (but not in North America), an inexpensive USB "dongle" receiver has become available for Amateur Radio applications in the VHF-UHF range. For our purposes, it consists of a tunable front-end and an 8-bit I/Q sampler, capable of running at over 2 MHz sampling rate. Because it is based on the Realtek RTL2832 quadrature decoder chip, it is often known as the "RTL-SDR" receiver. These devices are available on eBay (**www.ebay.com**) for $10 and up.

To our *Linux* boards, the RTL-SDR looks like a USB device. It has *Linux* support for SDR, so it was a natural addition to the panadapter project. While our software will display up to a 2 MHz spectrum slice with the RTL-SDR, we probably do not have the CPU resources to keep up with a continuous 2 MHz sample rate — at least not without extra programming. I manage by taking data in bursts only when the computer is ready. Discarding some incoming data reduces sensitivity, but we still can get a useful spectrum display.

I should note that the RTL-SDR is a very simple receiver, with large instantaneous bandwidth and limited sensitivity. It is easily overloaded by strong out-of-band signals. Serious amateur work may require a good antenna, a preamplifier, and a bandpass filter.

Downloading and Installing the Project

At this point I assume you have set up your Raspberry Pi, BeagleBone Black, or other small *Linux* board according to the vendor's directions and the hints we have given above. You will have a *Linux* environment that includes *Python* and necessary packages. The job now is to download and install the panadapter *Python* files.

All the files for the project can be downloaded as a compressed zip or tar archive at **www.arrl.org/qst-in-depth** or at **www.aa6e.net**. Download this file to a convenient directory and then use the command `unzip iq.zip` or `tar xzf iq.tar.gz`, which will expand the archive into a new "iq" directory.

Setting Up a Sound Card

The *iq.py* code needs to know which *PyAudio* index number to use for audio input. If you have only one audio device, things are simple. You can use the default setting (–1). If this doesn't work, or if you have multiple audio devices, you may need to do some more work.

I provide the special program *pa.py* so that you can see the sound devices that *PyAudio* sees. Type `python pa.py` in a terminal window, and examine the output. The following is a partial listing of *pa.py*'s output for a Raspberry Pi with a Sound Blaster SB1240 USB sound card:

```
DEVICE: 0; NAME: 'bcm2835 ALSA: bcm2835 ALSA (hw:0,0)'
defaultSampleRate : 44100.0
maxInputChannels : 0
maxOutputChannels : 2

DEVICE: 1; NAME: 'USB Sound Blaster HD: USB Audio (hw:1,0)'
defaultSampleRate : 48000.0
maxInputChannels : 2
maxOutputChannels : 2

DEVICE: 2; NAME: 'USB Sound Blaster HD: USB Audio #1 (hw:1,1)'
defaultSampleRate : 48000.0
maxInputChannels : 2
maxOutputChannels : 2

DEVICE: 3; NAME: 'USB Sound Blaster HD: USB Audio #2 (hw:1,2)'
defaultSampleRate : 44100.0
maxInputChannels : 0
maxOutputChannels : 2
```

PyAudio's device 0 is the Pi's on-board bcm2835 output-only "sound card." Devices 1 – 3 correspond to the three logical devices internal to the SB1240. Because this card's internal addressing is not documented, we use trial and error to find that *PyAudio* device (index) #2 gives us the SB1240 input channels we need for the panadapter.

I can get similar information by using the *Linux* and ALSA command line. The virtual directory /proc/asound/ describes the current ALSA status. For example, you can type `cat /proc/asound/cards` to get a listing of all attached sound cards. Other useful commands include `alsamixer`, `amixer`, `arecord`, and `aplay`. Check their pages for details.

When you use the RTL-SDR dongle for input, it will not show up as a sound card. No index setting is required, because we do not use *PyAudio*. The code finds the RTL device automatically.

Setting Up an RTL-SDR Dongle

Using the RTL-SDR dongle is simple. You attach an antenna, and you plug in your

In-phase and Quadrature IFs

In-phase and quadrature (I/Q) sampling refers to the technique of mixing (multiplying) an RF or IF signal with a local oscillator that is provided in 0 degree and 90 degree phase shift versions. This produces two IF streams, which may be treated as real and imaginary parts of a complex signal. With I/Q IFs, you can distinguish positive and negative sidebands relative to the local oscillator. It is the same principle that is used in the "phasing method" for single sideband generation and detection. The IFs are sampled and converted to digital form with an analog to digital converter. For more information, consult "DSP and Software Radio Design," Chapter 15 in recent editions of *The ARRL Handbook for Radio Communications*.

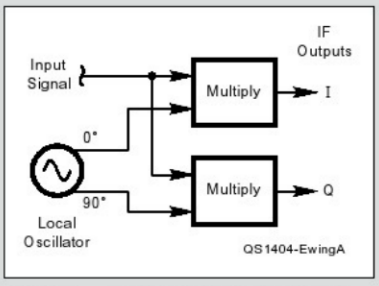
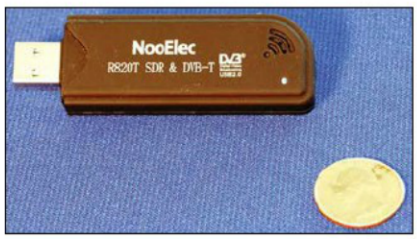


Figure A — In-phase and quadrature IFs.



An RTL-SDR dongle receiver.

dongle to your USB hub. The initial operating frequency and sample rate are set from the command line as shown below.

Setting Up a BeagleBone Black and LCD4

The LCD4 is one several LCD capes that can stack on the BBB. You need to plug it in to the CPU board, but be sure to use the right orientation: If the BBB Ethernet jack is at the left, the LCD4 pushbuttons should be along the right side.

With the LCD4 installed, the BBB should boot up to an LXDE desktop on the LCD. The HDMI video output is disabled automatically. Other sizes of LCD cape (eg, 3 inch or 7 inch) can probably be used, but the SCREEN_SIZE value in *iq.py* should be adjusted appropriately.

The buttons on the LCD4 are labeled left, right, up, down, and enter. Pressing one of these buttons is equivalent to pressing the corresponding arrow keys (or enter) on a standard keyboard. Each press of the ENTER button produces a new help overlay on the LCD4. An example is shown in Figure 4.

Command Line Options

Because it is designed for operating on different platforms for different kinds of operating, the *iq.py* code has many user-settable parameters and modes. You can change most of them from the *Linux* command line. Once you find a combination you want to use repetitively, you can put it into a shell script (a file) or define a shell alias that has everything set for you. Or, you can edit the source code *iq_opt.py* and change the default values.

The code is normally started from the *Linux* command line, using the *python* command. I assume you have started a terminal window and changed directory (*cd*) to the directory where you placed *iq.py*. If you feel lucky, you can try the command: `python iq.py`.

This will use all the default values. If you have a complicated sound card or multiple sound cards, you will have to use *pa.py* (or trial and error) to find the appropriate index value, as discussed above.

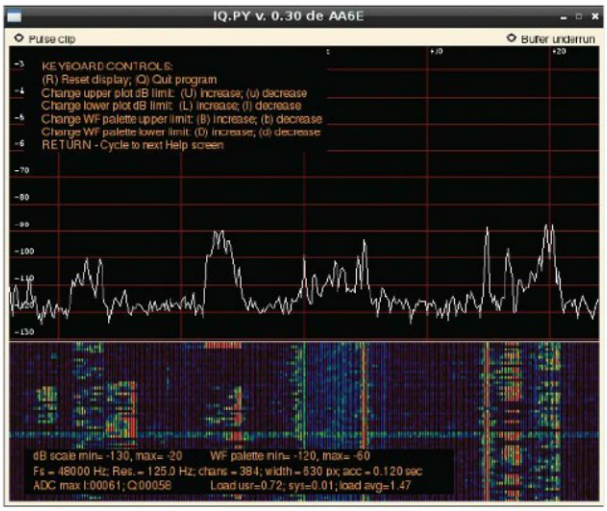


Figure 4 — Screen shot of *iq.py* with help overlay.

On the Raspberry Pi, for example, we might use:

```
python iq.py --index=1 --buf-
fers=15 --taking=4 --size=256.
```

The index setting is okay for a typical input sound card.

A typical command to start receiving with an RTL-SDR dongle would be:

```
python iq.py --RTL --rate=
2048000 --rtl_freq=144200000.
```

This would set up a 2.048 MHz sample rate centered at 144.2 MHz.

A complete table of options is provided in the supplementary files.

Conclusion

The Tiny *Python* Panadapter is usable software that runs on very inexpensive hardware. It is implemented in *Python*, which is a very capable programming language that is relatively easy to work with, although lower-level approaches (like *C* or *C++*) could offer greater performance at the cost of more programming effort. The *Python* approach, supported by *PyGame*, *PyAudio*, etc. is a good choice if we want the code to be accessible to as many experimenters and learners as possible.

There are many directions this project could take. It could readily support recording and playback of the I/Q IF signal. It could support full SDR transceiver operation, providing demodulated audio on receive, and generating I/Q IF signals on transmit. For experimental work, we might prefer to run on a modern *Linux* PC, which will usually have more computing and graphics power than the small cards offer. Squeezing the application into a small device is fun, but it does limit the functionality.

My thanks to Leigh Klotz Jr, WA5ZNU, for helpful discussions during the development of this article.

Notes

[1]A panadapter is a spectrum analyzer that displays of power vs frequency, generally showing the IF passband surrounding an amateur's current operating frequency.

[2]SoftRock is a series of kit-form receivers and transceivers developed by Tony Parks, KB9YIG. They are available from **flvedash.com**. There is an active "softrock40" Yahoo! group for user support.

[3]FiFiSDR is a kit receiver described at **o28.slscha.net/fifisdr/trac** and **www.df3dcb.de/FIFI-SDR_FA1110.pdf** (in German). A partially assembled version is sold by *FunkAmateur* magazine at **www.box73.de/product_info.php?products_id=2425**. The receiver was reviewed in the September 2013 issue of *QST*.

[4]The BeagleBone Black is developed by the BeagleBoard.org Foundation. See **beagleboard.org**.

[5]The Raspberry Pi is developed by the Raspberry Pi Foundation (UK). See **raspberrypi.org**.

[6]See Robert Nickels W9RAN, "Cheap and Easy SDR," *QST*, January 2013, p 30, **sdr.osmocom.org/trac/wiki/rtl-sdr**.

[7]The Fourier transform (*spectrum*) provides a complex voltage-like value in each frequency channel. The *power spectrum*, the power in each channel, is the square of voltage (really the complex amplitude squared). Finally, the *log power spectrum* is the logarithm (usually given in dB) of power in each channel.

[8]Jonathan Taylor, K1RFD, "Computer Sound Cards for Amateur Radio," *QST*, May, 2007, p 63.

[9]For example, data from Larry Phipps, N8LP, **www.telepostinc.com/sound_cards.html**.

[10]Current peak values for I and Q samples are displayed on the overlay panels of the *iq.py* program.

Martin Ewing, AA6E, was first licensed in 1957, when *Sputnik* went up and SSB was still new. After studying physics and astronomy, he had careers in radio astronomy and academic computing until retiring in 2002. He currently serves as an ARRL Technical Advisor and as a volunteer in the ARRL Laboratory, with particular interests in *Linux* software and software defined radio. You can reach Martin at 28 Wood Rd, Branford, CT 06405, or via e-mail at **aa6e@arrl.net**.

For updates to this article, see the *QST* Feedback page at www.arrl.org/feedback.