Extracted from:

Explore Software Defined Radio

Use SDR to Receive Satellite Images and Space Signals

The Pragmatic Bookshelf

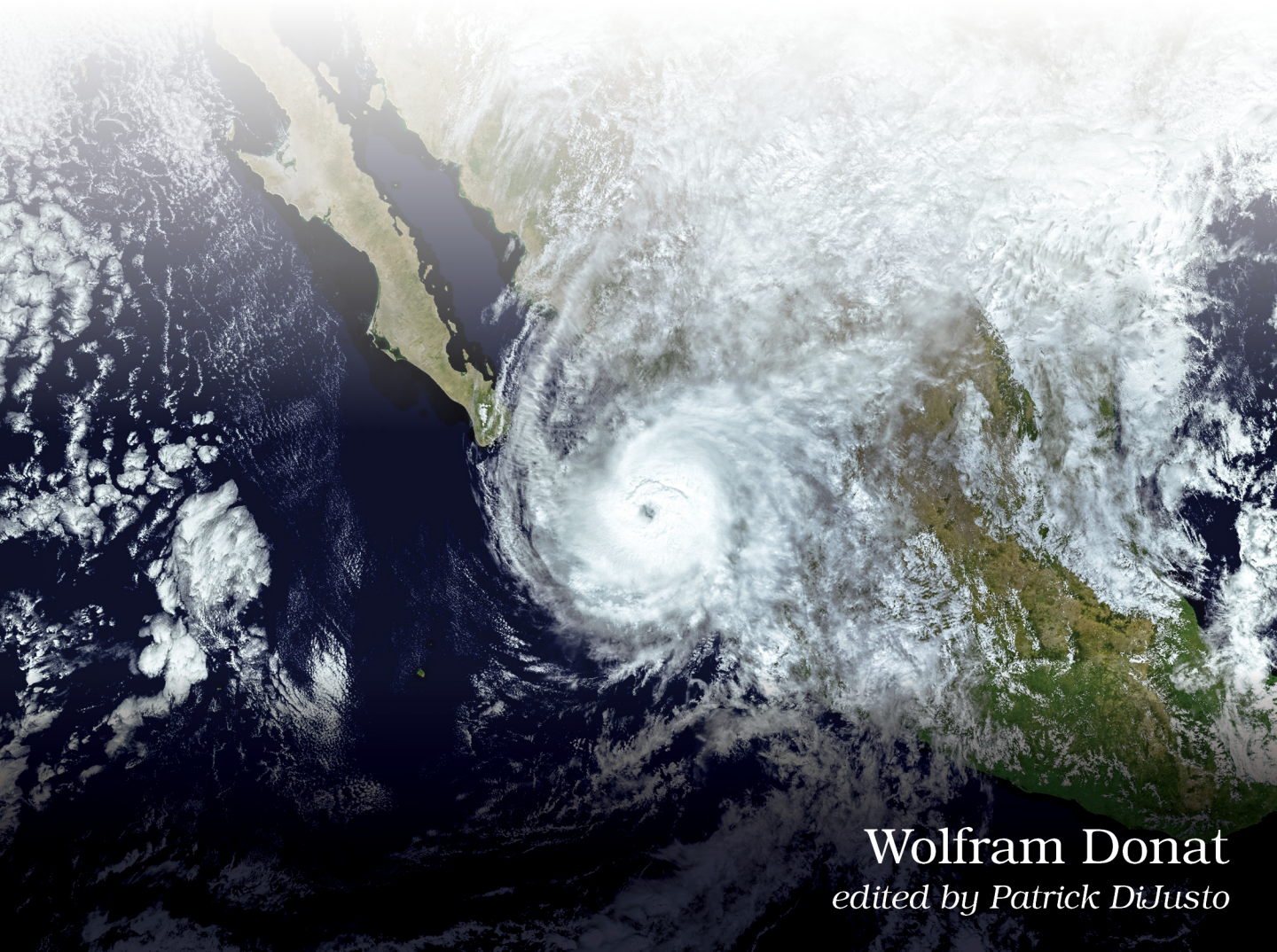
Raleigh, North Carolina

The Pragmatic Programmers

Pragmatic exPress

Explore Software Defined Radio

Use SDR to Receive Satellite Images and Space Signals



Wolfram Donat

edited by Patrick DiJusto

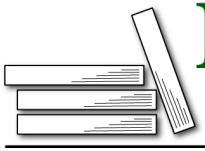
Explore Software Defined Radio

Use SDR to Receive Satellite Images and Space Signals

Wolfram Donat

The Pragmatic Bookshelf

Raleigh, North Carolina



Our Pragmatic books, screencasts, and audio books can help you and your team create better software and have more fun. Visit us at *https://pragprog.com*.

The team that produced this book includes:

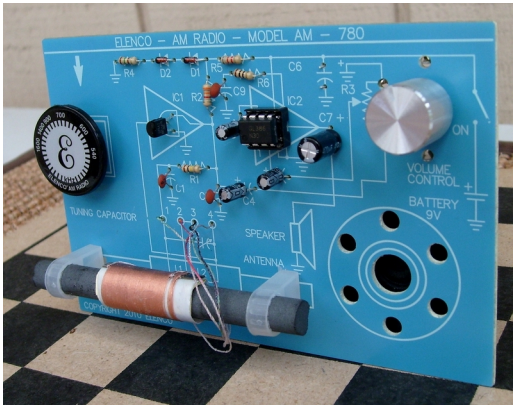
Publisher: Andy Hunt
VP of Operations: Janet Furlow
Executive Editor: Dave Rankin
Development Editor: Patrick DiJusto
Copy Editor: L. Sakhi MacMillan
Layout: Gilson Graphics

For sales, volume licensing, and support, please contact *support@pragprog.com*.

For international rights, please contact *rights@pragprog.com*.

ISBN-13: 978-1-68050-759-1
Encoded using the finest acid-free high-entropy binary digits.
Book version: P1.0—February 2021

Let's revisit, for a moment, the old days before silicon integrated circuits and software packages and newfangled computing machines. Back then, an enterprising hobbyist could use some transistors, a few variable resistors, a diode or two, and a tunable crystal and, with the right antenna and a healthy dose of skill and talent, receive and listen to almost any signal on the air. Back then, all broadcast radio signals were analog. While UHF broadcasting existed, there was almost nothing transmitting in that area of the spectrum, and satellite radio was unheard of.



Nowadays, analog signals are gradually being replaced by digital ones. The day may not be far in the future when all signals on the airwaves are digital. Back in 2009, all U.S. analog television signals were shifted to digital, and in early 2019 the nation of Norway shut off all their national analog FM radio stations in favor of digital audio broadcasting. If and when that happens in your location, the type of radio set shown in the preceding image will no longer be useful. It will still pick up the signals, but they will be unintelligible to the listener.

The difference between those older signals and the newer digital ones is just like the difference between old cassette tapes (remember those?) and CDs. Cassette players work by transforming audio waves back and forth into analog electrical signals, usually by way of a microphone or a speaker. Similarly, analog radio broadcasts work by changing electrical signals into radio waves that are transmitted. The signal directly represents the transmitted sound (or picture) by varying voltages and frequencies.

Digital music storage, such as CDs, on the other hand, works by storing the sound as patterns of ones and zeros, the same way a computer stores data. To listen to a CD, you have to read the pattern of ones and zeros with a laser and then decode them. Likewise, digital radio broadcasts transmit patterns

of numbers rather than analog waves. While your SDR dongle can pick up both types of signals, if someone is broadcasting via a digital signal, you won't be able to understand it—at least not without using some special software to convert digital to analog.

A perfect example of this digital encoding is found on the public safety and law enforcement bands. First of all, you should know that contrary to popular belief, it's *not* illegal to own and operate a police scanner—at least in the United States. (Check with your local laws if you're not reading this in the U.S.) You can purchase them on Amazon. However, should you decide to get yourself such a scanner, you may be disappointed, because many of these public departments have made the switch to digital broadcasting.

The reason many public departments have switched to digital is often because it's much cheaper to broadcast a digital signal than an analog one, and *this* is mainly for two reasons. First, digital radio offers better resistance to interference from other signals nearby on the spectrum. This means it can be broadcast with less power behind it, since you don't have to worry about having to overpower neighboring signals. And second, since it avoids the necessary physical imperfections of analog transmitters, more of the power you put into the broadcast gets translated into the signal rather than into heat energy loss in the broadcast equipment. So less power is needed to broadcast, and you get more bang for the buck with the power you utilize.

What all of this means is that if you're interested in listening to your local fire department or sheriff's station, you'll need to be able to decode the digital signal. Luckily, you can do this by adding some free digital decoding software to your SDR suite of programs.

One thing I find necessary to mention: digital speech or signal *encoding* is a completely different animal than signal *encrypting*. Digitally encoded speech signals can be decoded and listened to pretty easily with free software. Encrypted signals, on the other hand, are often official channels and frequencies that are not meant to be decoded or listened to by the general public. While software exists to decrypt these signals, it's often illegal for civilians to use. In other words, stick to your local police department broadcasts and stay away from broadcasts by the NSA or the CIA.

Finally, a word of notice from your author: although I managed to get digital speech decoding up and running on Windows, it was tricky. I had much more luck with my Linux installation, though it may have to do with the difficulty I had in locating suitable DSD-encoded broadcasts. If you're trying this on a Windows box and are having absolutely no luck, you may want to consider

taking the leap (if you haven't already) and transitioning to a Linux system, at least for your SDR experimentation.

Hardware

Luckily, no matter which operating system you're using, no additional hardware is required beyond what you were using to listen to standard FM radio signals. That being said, however, you stand a much better chance of receiving some good signals if you have a good antenna. In preparation for listening to weather satellites, it might be a good idea to find or purchase an old TV rabbit ear antenna setup.

These are available on Amazon or your local big box store for around ten dollars. I would suggest that you choose a set that terminates to a coax connection. You can then follow up your antenna purchase with two things: a longer cable for your dongle and an adapter to connect that longer cable to the coax cable on the antenna. Both of these are available on Amazon—look for an MCX male-to-female extension cable (often used for GPS systems) and an MCX-to-coax adapter. Once you've got your antenna and cable setup, you're ready to receive digital signals, as well as the older analog signals.

Software

Once again, I've separated instructions for Windows and Linux. If you're using Linux, feel free to skip ahead.

Windows

In addition to `SDRSharp`, you'll need both a program called `dsd` (which stands for digital speech decoding) and a way of sending the output from `SDRSharp` to the `dsd` application, via software running inside your computer.

Ordinarily, when you tune your dongle and listen to the radio frequencies with a program like `SDRSharp`, the output is being piped (obviously) to your

computer's speakers. It's really no different than listening to the radio in your car. If you want to decode digital speech signals, however, you need a way of sending the output of SDRSharp to a digital speech decoding program instead, and then sending *that* output to your speakers.

This is where a device called a *virtual audio cable* comes in. Picture it as connecting a cable to the "output" of SDRSharp and plugging it into the "input" of dsd—except that it's all happening virtually, completely in software. Two commonly used programs for Windows users are Virtual Audio Cable and VB Cable. Virtual Audio Cable has both a free and a paid version, while VB Cable is completely free.

Virtual Audio Cable can be found at http://software.muzychenko.net/eng/vac.htm, and VB Cable can be downloaded from https://www.vb-audio.com/Cable/index.htm. I experimented extensively with both programs, and while I had no luck at all with Virtual Audio Cable, I had no problems with VB Cable. I must specify here that I didn't try the paid version of Virtual Audio Cable, so it's possible that the paid version works just fine. Thus, your results may vary if you decide to try it, but my instructions going forward will be for VB Cable.

Once you've downloaded VB Cable, extract everything from the downloaded .zip file. Navigate inside the resulting folder and you should see, among a bunch of other files, a VBCABLESetup and a VBCABLESetupx64 application.

Choose the version right for your architecture, right-click it and choose Run as Administrator. All of the drivers should install, and if you check your Windows settings, you should see a new device, Cable Input, listed under your sound playback devices, and a Cable Output device under the recording tab.

While you have these settings open to check, set the VB-Audio Virtual Cable as your default recording device. This is because dsd—the program we're installing next—will automatically use the default recording device as its input.

Once VB Audio is installed, you'll need to download and install dsd. Ordinarily, this might be a tricky situation because the program is designed to run on Linux, and to run it on a Windows machine it needs to be compiled and installed using a Windows-based Linux emulator called Cygwin. Installing and running Cygwin can be instructive if you're interested in compiling and running dsd yourself, but it can also be problematic if you don't install it with all of the correct libraries, extensions, and compilers. It's a rabbit hole that many hobbyists may not want to follow.

We're fortunate that there are enough enthusiasts around that someone has done the hard work for us by compiling all of the necessary libraries and dsd

itself, and then releasing the resulting Windows binary. This allows you to avoid the entire Cygwin-based rigamarole. You can download the binary from my github repo here: https://github.com/wdonat/jumpstarting_sdr.

Once you've downloaded the .zip file, extract the contents, which will give you a directory called dsd-1.7.0. Inside that folder you'll find the dsd application; don't open it just yet, as it's not something you just double-click and open. Just remember where you put it.

Now, open up SDRSharp and tune to a digitally encoded channel. In my experience, this can be one of the most problematic portions of the project, as there doesn't seem to be a central listing of broadcast frequencies—digital or otherwise—sorted by area. Try Googling your local police department and public safety organizations, or http://www.radioreference.com has a pretty comprehensive database. If all else fails, you may need to (as I did) simply start scrolling through the frequency dial and looking for a digital signal. They're pretty easy to distinguish, as they tend to be a digital "hum" sound and they're often broadcast either sporadically or in regularly spaced bursts. However, it's a large spectrum, and scrolling through it can take a long time, so you may want to save this option as a last resort. I wish there was a way to narrow it down, but all of my research up to now hasn't revealed a general area of the spectrum where these frequencies tend to reside. If you're aware of any such area, please let me know! I can say that in my area of southern California, I had my best results around 500 MHz. Obviously, your results may vary greatly.

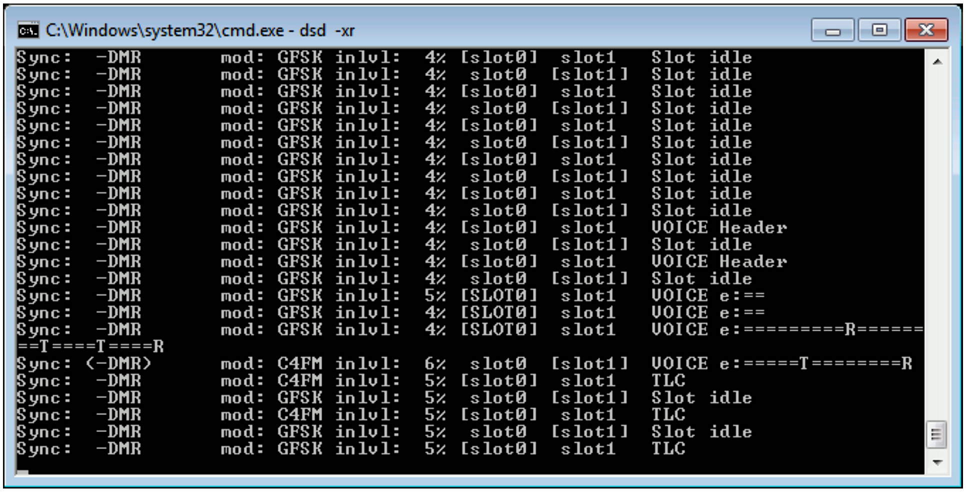
Once you've found a digitally broadcasting candidate frequency, you'll need to adjust your SDRSharp settings. First, set the audio output to VB Audio. Then set the receiving mode to NFM (Narrow Band FM), and then set the bandwidth to about 12 KHz. When you've tuned to the strongest part of the signal, press the Play button.

Of course, you shouldn't hear anything, because instead of sending the output to your speakers as you did before, you're piping the output to your VB Audio installation, which in turn is piping to the dsd application (which hasn't started yet). Open a windows command prompt, and in that window, navigate to your extracted dsd-1.7.0 directory. Once you're inside that folder, enter

```
dsd -i /dev/dsp -o /dev/dsp -fa
```

This command tells dsd to listen to the default audio recording device, which you've set to be the VB Audio output, and pipe it to the default audio output device, which is most likely your speakers. Finally, the -fa flag tells it not to discriminate and to scan for all sorts of encodings.

If you've found a dsd-encoded frequency, the terminal window should begin scrolling text, which will change when speech is detected. The text scrolls (adds an additional line) every time the signal updates; for example, when a user presses their TALK button, that will show as a status update of text.



If nothing happens, there are two options: either the frequency you're listening to is not dsd-encoded or your settings are wrong. Double-check your settings and keep trying, including scanning the dial for likely signals.

One of the most important settings to play with is your gain—both in SDRSharp and in your Windows sound settings. Try different values in your SDRSharp output, and then open your Windows sound recording settings and adjust the microphone sensitivity. Once you've found a good digital speech frequency, it's unmistakable, and you'll really be able to hear a difference when you adjust either one of those settings.