Software Radio

David Malone EI7GYB

November 22, 2009

Last Christmas, with the help of Ian Dowse, I started a simple software radio project that I had seen on the web. It is one of the most simple software radio receiver projects possible, but still has some interesting features and helps show some of the attraction of software radio.

First, let's talk about what the rationale for software radio is. Radio experimenters are very familiar with the construction of a radio: amplifiers, mixers, demodulators and audio circuits are bread-and-butter stuff. The idea of software radio is to replace certain parts of a radio, which have traditionally been circuits (i.e. hardware) with a computer program (i.e. software). You could think of this as replacing the parts of a radio that usually operate at the IF with a computer.

Why would you want to do this? Well, the primary part of a radio that you might replace is the modulator. Particularly in digital communications, there is a lot of interest in choosing good modulation schemes for the current radio conditions. A radio which does its modulation in software is free to choose almost any modulation, or even create new modulations, just by loading new software onto the radio. This is an exciting prospect for researchers, who previously had to design and fabricate circuits to try each new scheme. It is also interesting for manufacturers, who could sell a single radio that could do (say) 3G, WiFi and Bluetooth.

So, what does the most simple software radio look like? Well, any software radio receiver is going to need to convert the analogue radio signal into digital data for the software to work on. So, at a minimum we will need an analogue to digital converter (A2D). Naturally, we will also need an antenna and something to run the software on. The software radio project in question works with this minimum of components.

The project was devised by Juan Domenech Fernandez, and his description of it is available as a sequence of web pages beginning at http://www.domenech.org/homebrew-sdr/. It is centered around a chip called the BT878A, which is found on a number of TV Tuner cards for PCs. This chip is intended to be connected to a TV tuner circuit: it has pins that take the composite video and audio output from the tuner. The BT878A chip then digitises the video and audio and can pass it to the CPU of the PC to be displayed.

What Juan noticed was the speed of the A2D converter used for audio on the BT878A. For example, for CD quality audio, a sampling rate of 44.1kHz is used, but the A2D converter on this card can actually be programmed (with a little trickery) to run at 896kHz! Naturally, this is far above what people can hear, but it is in the range of long-wave radio signals. A sampling rate of 896kHz is enough to reconstruct signals of up to 448kHz.

Juan's project involves taking a standard TV tuner card with the 878A chip (about \in 80), cutting the audio output line from the TV tuner circuit, and instead connecting the chip directly to a long-wire antenna. The details of this simple change are at http://www.domenech.org/bt878a-adc/index-e.htm. Amazingly, the chip is sensitive enough that no RF amplifier is required to receive signals!

The next stage is to get some software going. I used Linux rather than Windows, because it was relatively easy to modify the driver to allow the chip to sample at 896kHz. The chip is supported by a driver called btaudio, and requires a one line modification to allow the high sample rates (the modified driver is available at http://www.maths.tcd.ie/~dwmalone/btaudio.tgz). Since the driver acts as an audio source, we can use



Figure 1: Traditional radio and software radio.

various audio tools to record and process the radio signals. In the rest of this article, I'll use a program called sox to show how this setup can be used as a radio spectrum analyser.

So, how do we use this setup? Well, the first step is to unload any other drivers that might have attached to the card and then load our modified driver, and set it up for recording. This sequence of commands should achieve that:

```
rmmod bt878
rmmod bttv
rmmod snd_bt87x
rmmod btaudio
insmod ~/btaudio/btaudio.ko debug=1
aumix -d /dev/mixer1 -q -2 100:100 -2 R -q
```

The second last command loads the driver from the directory shown with debugging messages enabled. The final command sets the recording levels to 100% - /dev/mixer1 controls the settings for the audio on the TV card.

The next step is to collect some RF samples from the driver. One option is to feed them directly to another sound card, which lets us hear the raw RF signal:

sox -r 44100 -t ossdsp /dev/dsp2 -t ossdsp /dev/dsp

Here /dev/dsp2 represents the BT878A's A2D converter and /dev/dsp represents my PC's regular sound card. I'm only sampling at a rate of 44.1KHz here, because that is all my sound card can output. Listening to the raw RF can be quite informative. For example, turning on a low-energy light bulb near the PC allows us to hear the RF noise produced by these light bulbs.

Rather than outputting the RF samples to a sound card, we can also record them in a .wav file. Sox can do this using the command:

sox -r 896000 -t ossdsp /dev/dsp2 -t .wav /tmp/test.wav

Now I am using the full sampling rate, and sox will keep recording until I hit Control+C (or until I run out of disk space!) From these samples I can then produce a spectrogram using a command like:

sox /tmp/file.wav -n spectrogram -x 40 4 -Z -50 -z 70 -s -o spectrogram.png

The options control the colouring of the spectrogram and the level of detail. Take a look at the example spectrogram. The horizontal lines represent fixed RF frequencies (the frequencies are shown in kHz). The darker the colour, the more power is at that frequency. As you move from left to right, the spectrogram shows how the power changes over ten seconds of time. The particularly dark line at 252kHz is RTE Radio 1, and you can see the symmetric AM modulation on either side of it. You can see the lines for other long wave broadcast stations too, including BBC 4 on 198kHz and TDF on 162kHz.

There are a number of interesting signals in this band which can be seen, including time signals (Rugby MFS 60kHz, Frankfurt DCF 77.5kHz), airport beacons (slow Morse from Dublin at 316kHz and Garristown at 407kH) and maybe even a faint LORAN-C navigation signal around 90–110kHz. It also covers the amateur band around 135.7–137.8kHz. There are a number of spurious signals: for example, I believe the signal with the steps around 133kHz is likely to be generated inside the PC, and harmonics of it can also be seen.

In a sense, we now have a radio with a demodulator that produces spectrograms, instead of audio. Using this, I've produced a number of interesting animated spectrograms, including showing how the radio spectrum changes over the day, showing how the spectrum changes as you rotate a loop antenna to null stations out and showing the MSF transmitter coming and going during maintenance.

Now that we have these signals, we can demodulate any of them in any way we choose, either as we receive them or later (by storing the RF samples). We could choose to AM demodulate them, we could apply different (digital) filters to them, we could even look at the frequency stability of the transmitters or look for out-of-band emissions. There are prepackaged software systems for this, or, alternatively, you can homebrew your software.

I hope this gives a quick flavour of what is possible with a software radio, even with this minimal setup. Naturally, when combined with other standard radio components, such as RF amps and mixers, even more is possible.

©David Malone EI7GYB.



Figure 2: The radio spectrum at 1pm on 11/04/2009.