DttSP: An SDR Core in C

Frank Brickle, AB2KT
Bob McGwier, N4HY

Last year at this conference we presented an overview (*AMSAT OSCAR Echo, SDR-1000, and Higher-speed FSK*, by Frank Brickle, AB2KT and Bob McGwier, N4HY) of the design for a suite of software to implement a Software Defined Radio initially for the FlexRadio SDR-1000. We described the functional breakdown of the software and the overriding design goals and principles that were to guide the development.

The first iteration of that software is complete. It is currently embedded in the new SDR-1000 console running under Windows. However, the DSP core is independent of the Windows version – indeed, it is basically independent of the SDR-1000 hardware as well. We are currently completing the parallel Linux version of the console and are poised to start the inevitable rewrite of the core software. What we would like to describe here are the revised estimates of what the software is good for, and further, where it will be going over the next few months.

I. We have a system that is dynamically reconfigurable.

What do we mean by "dynamic reconfiguration?" At a minimum, this implies the ability to rearrange and recombine individual processing elements such as filters, demodulators, or signal measurements *on the fly* without recompiling any of the soft
ware. In essence, it means the capacity to "patch together" new radios of arbitrary design at will. What is especially important is the capacity to generate new processing chains either interactively or by program – in other words, to enable low-overhead experimentation for users, but also to provide one of the critical functions of a self-aware (cognitive) radio.

We also wanted to make it possible to integrate other tools into the processing chain. For example, there is a host of high-quality audio processing tools for such things as equalization, metering, and offline storage (recording). These tools exist either as plugins or expect functions in the form of plugins, so we are making the SDR components available in this format as well.

There is also a purely utilitarian motive. One of the major design goals of the software was the ability to remote the application – to place it physically at an antenna, for example, so as to minimize feedline loss, etc. This configuration requires considerable flexibility but it also implies having limited hardware in the form of embedded controllers at the far end. Having the software be dynamically reconfigurable places lighter demands on the running environment, much as having loadble kernel modules obviates recompiling the OS kernel itself to add functionality.

II. A single system can handle multiple radios.

This can take a number of forms.

A system can have multiple soundcards, each one of which feeds an independent processing chain. If you're running on a CPU with enough cycles, why not use them? But there are other possibilities. For example, with a common clock on multiple board sets, it is possible to achieve sample-accurate N-channel-combined reception for broadband processing.

On another front, multiple radios make it simple to have several different processing chains attached to a single input. Typical scenarios for this would be having constant signal measurement running alongside ordinary usage; or automatic classification, to achieve something like automated SO2R operation; or background logging and diarization, running independently of typical operation.

III. The software will work with all current Linux sound systems.

At present there are two-and-a-half standards for audio processing under Linux: OSS, ALSA, and ALSA augmented by the jack connection system. Since the DttSDR core is decoupled from hardware input and output, and only requires simple synchronization, it can be integrated into any of those systems. Fundamentally this means that the core can be embedded in either a blocking or a non-blocking configuration, which leads to the next point.

IV. The software will work with a variety of multiprocessing options.

The major contenders at this point are pthreads, primarily a preemptively-scheduled architecture; GNU pth, a highly-portable cooperative (non-preemptive) arrangement; pthreads implemented on top of GNU pth; or even separate, individual user processes, which are fairly lightweight under Linux.

Once again, having flexibility in this regard means having the ability to accommodate a variety of different operating environments, and it also facilitates implementations for a variety of operating systems as well, such as Mac OSX or the various BSDs.

V. Minimal modifications (should) be required to cooperate with other programs.

What we have in mind here is interfering minimally with people's other favorite applications like gMFSK or other digital mode environments. Ideally it should be possible to run the entire SDR environment from within your favorite program. To that end we are trying hard to make it possible for the SDR core to be hidden transparently in the audio-io facilities, at least as far as the other applications are concerned.

VI. Bindings for other languages.

Dynamic reconfiguration has another nice side effect: it is only a short step away from making the full core functionality available to many other popular languages. This ought to make it easier, basically, to play with, and perhaps broadens the applicability of the combination of hardware and software.

Currently there are tools available to automate generation of bindings for python, perl, ruby, guile, and tcl, among the scripting languages; for Common Lisp and ocaml, among the Object-Oriented dynamic languages (a nice match); and, most provocatively, matlab/octave, R, and S. There are some interesting educational possibilities here.

VII. Other data paths and control streams.

One artifact of a Unix-style driver/filesystem configuration is the ease with which various io and networking facilities can be exploited. We expect to be making available USB and FireWire taps at various places in the processing chain.

Another intriguing possibility is making use of less-conventional multimedia facilities like MIDI, which offers a medium-bandwidth, rich, but simple io stream essentially for free. We are looking at smoothing out CW performance of the SDR-1000 by optionally taking keying input from a game port via MIDI software streams.

These then are some of the features of the DttSDR core that are currently at or near completion. The Windows version being distributed is a beta release. The corresponding Linux version, however, is still at the alpha stage, since it is still subject to considerable change.