An introduction to BasicDSP

Pieter-Tjerk de Boer PA3FWM and Niels Moseley PE1OIT

(published in SPRAT nr. 133, winter 2007/8)

In this article, we present the BasicDSP software, which makes building and modifying simple radios in software as easy and accessible as in hardware.

Background

In recent years, there has been a growing interest among radio amateurs for Software Defined Radio, SDR: radio signal processing done in software on a computer, rather than in analog hardware. This digital signal processing (DSP) has several advantages over analog signal processing: it does not suffer from the non-idealities of practical components (temperature sensitivity, limited Q, etc.) allowing e.g. steeper filters, and is very flexible (adaptive filters, new modulation schemes, etc.)

A typical amateur SDR setup consists of a circuit that mixes a part of the HF spectrum down to frequencies in the audio range, which are fed to a PC sound card for conversion from analog to digital for further processing (tuning, filtering, demodulation) in software. The external hardware is often very simple, e.g. just a few ICs in the SoftRock kits; however, the software is typically rather complicated and not very accessible to experimenters.

At the G-QRP Mini-Convention, Jan Verduyn, G0BBL/PA5D, demonstrated the BasicDSP program, which we wrote as a companion to PA3FWM's series of SDR articles in the Dutch magazine `Electron'. In this article, we give a brief tutorial of this program, ending with a minimal but functioning SDR program, and covering some elementary digital signal processing theory along the way. The software can be downloaded from ***http://wwwhome.cs.utwente.nl/~ptdeboer/ham/basicdsp/***

Digital Signal Processing in BasicDSP

The first thing to understand about digital signal processing, is that it happens on a sample-by-sample basis rather than continuously. The A/D converter measures its (analog) input voltage periodically, e.g., 8000 times per second; this rate must be at least twice the highest input frequency. Each measurement is called a sample and must be processed by the software. Similarly, the D/A converter that drives the loudspeaker also expects to get a new sample periodically. Thus, we see what a DSP program must do: get a sample from the A/D converter, do some computations on it, send the result to the D/A converter, and wait for the next sample.

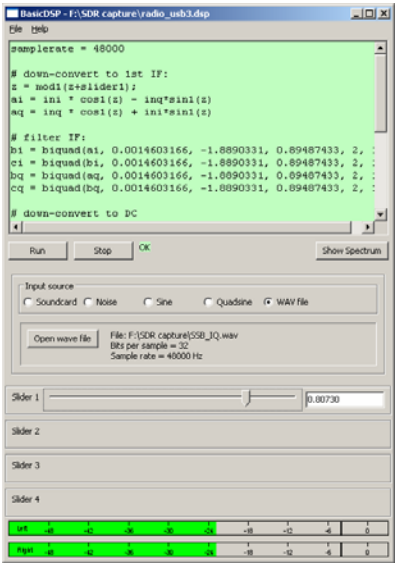
In BasicDSP, you can specify the computations using a few lines of programming code, to be executed once for each sample. All other tasks, such as interfacing with the soundcard, are taken care of automatically by BasicDSP.

Getting started

Try first the following minimal program:

```
out = in
```

Type this into BasicDSP's text entry field and press the 'Run' button. The text entry field will turn green to confirm that the program contains no syntactical errors and is now running.



But what does it do? It simply copies the content of the variable called 'in' (which contains the input sample) into the variable called 'out', the content of which is sent through your soundcard to your computer's speakers. The input sample can be read from either the line/mic input of your soundcard, or a .WAV-file, or a locally generated sine wave or white noise signal. Thus, you can now hear either of those sources on your PC speakers.

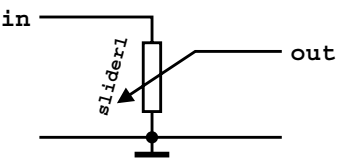
Perhaps you find the sound to loud?
The following program attenuates it by 20 dB, which is the same as multiplying by 0.1:

```
out = in * 0.1
```

The attenuation can also be made variable:

```
out = in * slider1
```

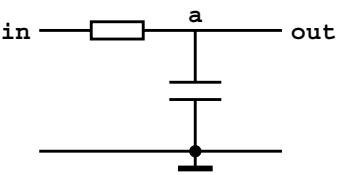
'Slider1' refers to the setting of the first of four controls that are located in the lower half of the BasicDSP window.



A first-order low-pass filter can be programmed as follows:

```
a = a + slider1*(in-a)
out = a
```

We see a new variable being used here, arbitrarily called 'a'. Each such a name is a reference to a location in the computer's memory, in which numbers can be stored temporarily; thus, the output sample may depend on both the current input sample, and on previous samples through information stored in such variables.



The program does the following: for every input sample read, it adds to 'a' a contribution that is proportional to the difference between 'in' and 'a'.
Compare this with the circuit sketched: the voltage on the capacitor changes at a rate which is proportional to the current through the resistor, which in turn is proportional to the difference between the input and output voltages. We know the circuit is a low-pass filter: slow changes of the input voltage are tracked by the output voltage, fast changes are

not because the capacitor cannot charge and discharge quickly enough. Precisely the same happens in the computer program: fast changes of 'in' are not tracked by the variable 'a', slow changes are. The setting of 'slider1' in this example determines the cut-off frequency, just like the resistor's value does in the analogue circuit.

We can also build an oscillator in BasicDSP. The first step for this is building a saw-tooth generator:

```
sawtooth = mod1(sawtooth + slider1)
out = sawtooth
```

At every sample instant, this program adds the value of 'slider1' to the variable 'sawtooth'. The function mod1 however leaves only the fractional part of this sum, so when the sum reaches or exceeds 1, 1 is subtracted from it. Thus, if e.g. slider1 is set to 0.2, sawtooth will successively get the values 0, 0.2, 0.4, 0.6, 0.8, 0.0 (not 1.0!), 0.2, and so on. This is an oscillation with a period of 5 samples, which corresponds to a frequency of 8000/5=1600 Hz if the default samplerate of 8000 Hz is used.

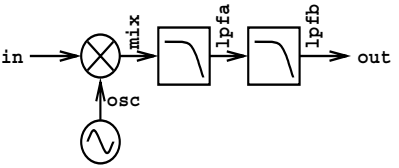
To turn this into a sine wave, we use the sin1() function:

```
sawtooth = mod1(sawtooth+slider1)
osc = sin1(sawtooth)
out = osc
```

sin1() is the sine function taught in school, except that its argument just needs to cover 0 to 1 for a complete period of the sine wave, as opposed to 0 to 360 degrees for the regular sin function. Thus, it converts our sawtooth into a sine wave.

Now, we can build a very simple direct-conversion radio-receiver:

```
samplerate = 48000
sawtooth = mod1(sawtooth+slider1)
osc = sin1(sawtooth)
mix = osc * in
lpfa = lpfa + slider2*(mix-lpfa)
lpfb = lpfb + slider2*(lpfa-lpfb)
out = lpfb
```



The first line sets the sample rate to 48000 Hz, the highest supported by most soundcards. The next lines are our sine-wave oscillator, followed by a mixer (the mixing operation is a multiplication), and finally two first-order low-pass filters.

The above program needs an input signal. That can either be taken from one of the popular SDR downconverters such as the SoftRock kits, or from a recording of the output of such a downconverter. A suitable .WAV file containing SSB signals is available at the Flex Radio website: http://support.flex-radio.com/Downloads.aspx?id=59

This receiver program works, but it is very simple. Further improvements are possible, such as a steeper filter, and rejection of the image frequency (using the quadrature mixer found in most SDR front-ends).