# Introduction to SDR and GNU Radio

Introduction

Modern wireless communication systems process signals as sampled digitized signals rather than as analog signals. This is known as "Digital Signal Processing" (DSP) and is used because of the lower cost and lower power consumption of digital compared to analog electronics.

Exceptions include the Analog Front End (AFE) that converts between digital baseband signals and the passband RF signal, RF power amplifiers and DC power supplies, all of which are necessarily analog.

Most DSP basebands are implemented using application-specific digital circuits to minimize costs and power consumption. However, a sufficiently-fast computer can also implement the baseband signal processing. This is known as "Software Defined Radio" (SDR):
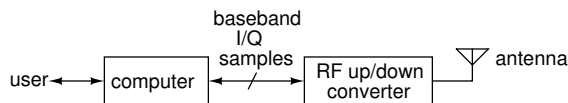
GNU Radio is an open-source project that develops software for SDR systems. GNU Radio Companion (GRC) is a graphical design tool that lets a user create GNU Radio software without having to write code. Blocks representing signal processing functions are added to a diagram, configured, and interconnected using a graphical user interface. This "flowgraph" is then turned into an executable script that executes the processing blocks in the required order to process the samples. In this lab we will use GRC to configure and run SDR software.

The advantages of GNU Radio and similar tools such as MathWorks's Simulink and National Instrument's LabView are that they don't require programming and that the flowgraphs can be self-documenting. However, many users prefer programming languages such as Matlab (e.g. using the Communications Toolbox), because they are more flexible.

GNU Radio and similar tools are also used to simulate communication systems. The simulations can be used to optimize designs and to generate test vectors for verification of DSP hardware.

A GNU Radio flowgraph can send/receive samples to/from radio hardware, files, network connections and other GNU Radio flowgraphs[1].

An example of a flowgraph created by GRC is shown in Figure 1.

Dataflow Flowgraphs

It may appear from the flowgraph that the blocks operate in parallel (simultaneously). However, since the blocks are implemented in software they must execute sequentially. This "dataflow" architecture means that blocks execute only when they have data to process. The order in which blocks are executed is determined by a scheduler that examines the amount of data available on the input ports of each block.

For efficiency reasons there are buffers between blocks that allow blocks to process multiple samples each time the block is executed. However, this buffering increases latency and can prevent proper operation of certain delay-sensitive applications such as those using feedback control (e.g. PLLs). Many blocks allow the maximum buffer sizes to be adjusted to minimize latency.

Complex Baseband Representation

It is possible to down-convert a real bandpass signal centered around a carrier frequency into a complex baseband signal centered about zero (DC). The complex baseband signal is centered on zero and, unlike real signals, the frequency components are not constrained to have even (or odd) symmetry.

The complex baseband architecture reduces implementation costs because receivers and transmitters can be implemented using only low-pass filters that can be integrated into ICs. This reduces costs by reducing parts count and board space.

The baseband interface of the SDR hardware uses sampled complex baseband signals in I/Q

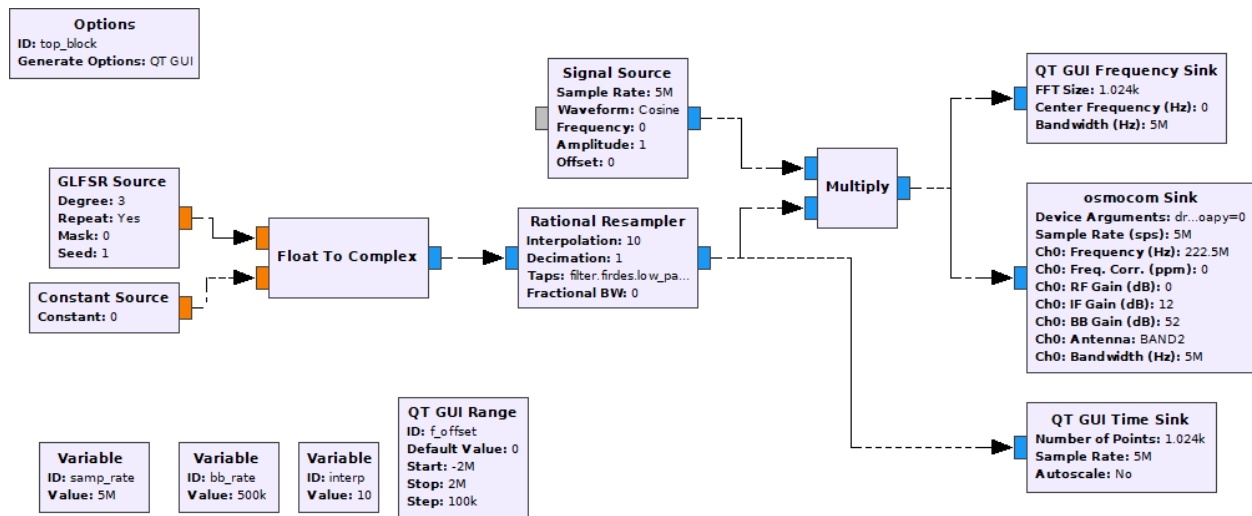[1]Including those written in programming languages such as Python, C++ and Matlab.

Figure 1: GNU Radio Companion flowgraph for PRBS-modulated BPSK (2-QAM).

(real/imaginary) format and most of the signal processing operates on these complex signals.

As an example, in this lab we will translate the frequency of signal by multiplying it by a complex sinusoid. GNU Radio also includes blocks that convert between signal of different types (e.g. byte, float and complex).

ML PRBS

In evaluating, testing and implementing communication systems we often need a signal that has statistics similar to that of a random binary source but which is deterministic. This is called a pseudo-random bit sequence (PRBS).

The most common type of PRBS is the Maximal-Length PRBS (ML-PRBS) which has certain desirable properties, including having the maximum possible period for a given generator complexity.

An ML-PRBS can be generated using flip-flops and xor gates to compute feedback in a circuit called a linear-feedback shift register (LFSR).

Filters

The radio spectrum is divided into bands which are allocated to different uses (broadcasting, cellular, radio-navigation, etc.). Each band is typically divided into smaller ranges, called channels with each channel allocated to a different set of users.

Filtering is a important part of most communication systems. Transmitters use them to limit out-of-band power that might interfere with users of other channels and receivers use them to reject signals present on other channels.

Low-pass filters are also used when changing the sampling rate of digitized signals. Increasing ("interpolation") or decreasing ("decimation") the sampling rate requires post- or pre-low-pass filtering the signal to avoid aliasing.

There are many digital filter types (IIR, FIR), architectures (parallel, cascade, FFT) and design methods (window, mini-max). In this lab we will use a simple window-based design of an FIR filter to interpolate a low-bandwidth baseband signal's sampling rate to the higher rate required by the SDR transmitter.

Variables and GUI Widgets

A GNU Radio flowgraph is actually a Python script. This allows us to embed variables (using the "Variable" block) and arbitrary Python code into a flowgraph.

It's also possible to include blocks that create user interface components ("widgets") that are active while the block is running. These can be used to display waveforms or frequency spectra and to change values of variables [2]. These variables can be used in expressions that configure the operation of blocks. You saw examples of this in the UI presented by the

---

[2]Parameters whose names are underlined in a block's properties dialog box can be updated during flowgraph execution.

previous lab which was implemented using GNU Radio.

Procedure

In this lab you will practice using GNU radio by building a transmitter that incorporates some typical blocks. The flowgraph generates a ML-PRBS, multiplies it by a complex exponential to shift the frequency and then interpolates to increase the sampling rate.

The diagram in figure 1 shows the flowgraph.

Using GRC

Start GRC. Under the View menu enable the Block Tree Panel, Console Panel and Variable Editor. Make sure 'Hide Variables' is not checked.

The easiest way to add a block is to search for it by name. Click on the magnifying glass icon (🔍) on the menu bar or type Control-F. Type part of the block name and double-click on the desired block name. This adds it to the flowgraph. You can them move the block by dragging it (any connections will follow).

To connect block outputs to inputs click on the output port and then click on the input port you want to connect it to.

As in any diagram, the blocks should be arranged so that the logical flow is left-to-right and top-to-bottom whenever possible.

Samples flowing between blocks can be of different types. Bits are typically transferred in the least-significant bit of a Byte type and signal samples are typically Float or Complex types. In GRC the color of the input and output connectors reflects the signal type (e.g. orange for floats, blue for complex).

If there is a mismatch between the signal types the connector will be drawn in red.

You can double-click on a block to open up its properties dialog box. This allows you to configure the input and output types and other aspects of the block's operation. The block's title and some of the block's properties are displayed inside the block in the flowgraph.

Some blocks define data structures rather than process samples. An example is **Variable** blocks that set the value of a variable. This variable can be used when configuring other blocks. By default a variable, `samp_rate`, is included in a new flowgraph.

Configure the Flowgraph

Use the File ▶ Open menu item to start a new QT GUI flowgraph. Then add, configure and connect the following blocks:

Variable Add a second variable with ID `interp`. This will be used to set the interpolation ratio between baseband samples and RF samples.

Add a third variable with ID `bb_rate` that has the value `samp_rate/interp`. Note that the value of a variable can be set to an arbitrary Python expression. The code generated for each block is shown in the "Generated Code" tab.

GLFSR Source This block (Galois LFSR) generates a PRBS. Configure this block as follows:
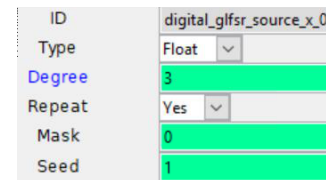
Degree, N, is the number of bits in the shift register. The PRBS period is $2^N - 1$ bits. Configure $N = 3$ which will generate a PRBS with a period 7 bits. Mask defines the LFSR taps. If specified as 0 a suitable generator polynomial for the specified degree will be chosen. Note that float outputs have values ± 1 while byte outputs have values 0 or 1.

Type Conversions Type conversions are sometimes required. For example, the GLFSR Source above is configured to output floats but these need to be converted to Complex values. Add a **Float To Complex** block that sets the imaginary component to zero by connecting a **Constant Source** with value 0.

Signal Source Add a [Co-]Sinusoidal signal source with a Complex output whose amplitude is 1 and whose frequency is set by the variable `f_offset`. It is configured as follows:

Rational Resampler Add an interpolator block configured as follows:

This interpolates by a factor `interp` and the low-pass anti-aliasing filter is set to the following expression:

```
filter.firdes.low_pass_2(1,samp_rate,
  0.5*samp_rate/interp,0.02*samp_rate,60)
```

This Python expression computes the FIR filter taps for a low-pass filter. The function parameters specify the gain, sampling frequency, -6 dB cutoff frequency, width of transition band and required stopband attenuation (in dB).

Multiplier This block does a complex multiplication of two inputs.

QT GUI Frequency Sink This block is similar to a spectrum analyzer and shows the power spectrum when the flowgraph is running.

QT GUI Time Sink This block is similar to an oscilloscope and shows the real and imaginary components versus time when the flowgraph is running.

QT GUI Range This block defines a GUI interface item that allows run-time changes to variables when the flowgraph is run. Set ID to `f_offset` to allow run-time control of the frequency of the complex [co]sinusoidal signal source block added above.

The GUI blocks have configuration options for how they should be displayed in the window that is created when the flowgraph executes. In addition to input types, ranges, labels, etc, the GUI Hint provides some control over the positioning of the GUI elements (and use of tabs). For example, a hint of [1,2] means to place that GUI element on the first row, second column.

osmocom Sink This block represents the interface to the transmit portion of the SDR hardware. It is configured as follows:


The Device Arguments `drive=lime,soapy=0` selects a driver, a Sample Rate of `samp_rate` sets the DAC sample rate, the Ch0 Frequency sets the RF center frequency[3], the Gain values set the gains of the digital and analog portions of the transmitter, Ch0 Antenna selects one of RF matching networks on the PCB, Ch0 Bandwidth selects the (two-sided) baseband bandwidth. Note that in this case we've set the bandwidth and the sampling rate to be the same (5 MHz) which is appropriate for complex baseband signals.

Measurements

Connect the LimeSDR Mini SDR board to the lab PC's USB 3.0 ports (on the front of the case) using the supplied USB 3.0 cable. Connect the SDR boards's **TX** output to the spectrum analyzer's **RF In** input using the supplied SMA-to-BNC cable:

[3]This covers an amateur radio band although we will not be transmitting "over the air" in this lab.
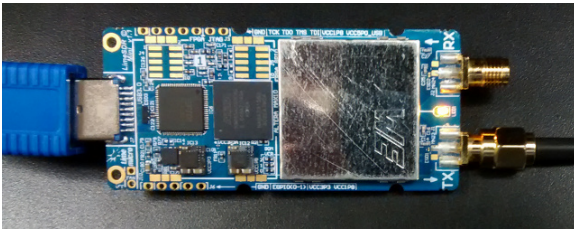
Select Run ▶ Execute, press F6, or press on the run (▷) icon. This will convert the flowgraph to a Python script and run it.

View the RF Spectrum

Configure the spectrum analyzer for a center frequency of 222.5 MHz, a span of 5 MHz, reference level of 10 dBm and resolution/video bandwidths of 10 kHz and 3 kHz respectively.

Capture the baseband spectrum shown on the GNU Radio Companion run-time window ("Top Block") and the RF spectrum as displayed on the spectrum analyzer:
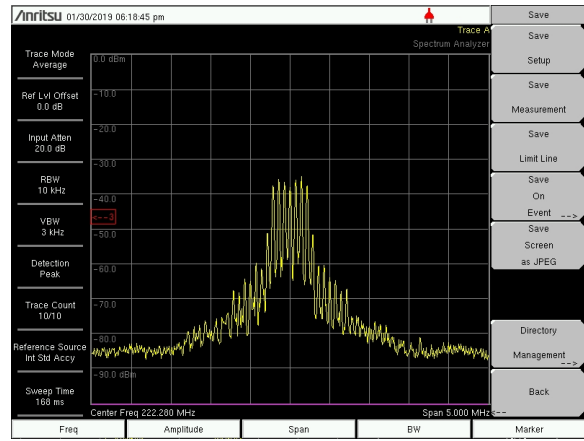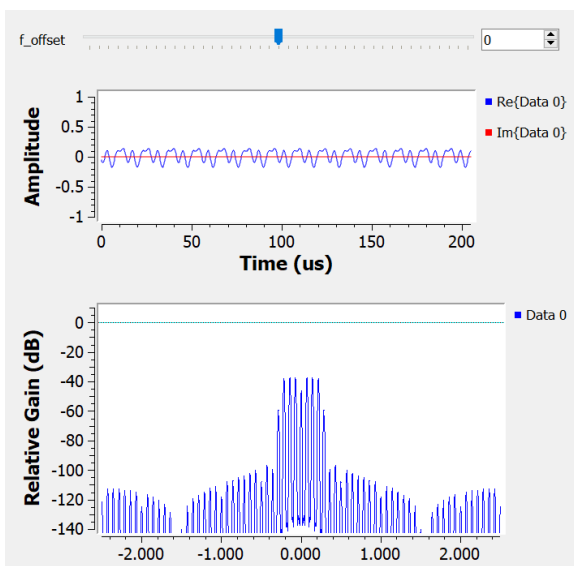
To capture the GNU Radio Companion GUI you can use the Windows Snipping Tool. To capture the spectrum analyzer display plug a USB flash drive into the connector on the analyzers front panel[4]. Select File (Shift-7) ▶ Directory Management and select USB as the Current Location. Select Back and Save ▶ Save Screen as JPEG to save an image file showing the spectrum analyzer display.

Additional Test Cases

Make the following changes and capture the resulting baseband signal and RF spectrum:

- shift the signal up in frequency by 1 MHz using the GUI slider

- change the PRBS period from 7 to 255 bits

- change the filter Taps to [1] to disable the interpolator's anti-aliasing filter

Lab Report

Submit a report in PDF format to the appropriate dropbox on the course web site.

Your report should include the four GUI and spectrum analyzer captures and answers to the following questions:

---

[4]Spectrum analyzers with older (version 1.x) firmware may show errors when trying to save to USB drives lager than 2 GB. The instructor may be able to lend you a small USB drive or (worst-case) you can crop a photo of the screen taken with your phone.

- What is the result of multiplying $e^{-j\omega_1 t}$ by $e^{-j\omega_2 t}$? What is/are the resulting frequency(ies)? How does this differ from multiplying two real sinusoids?

- How does the baseband spectrum shown in the frequency sink GUI differ from the one shown by the spectrum analyzer?

- What is the period of the PRBS sequence for $N = 3$? What is the spacing of the frequency components? What are the period and frequency spacing for $N = 8$?

- What happens when you disable the interpolator's low-pass filtering? What is the name of this effect?