

## Communication System Simulation

### Introduction

Computer simulations are the most practical way to predict the performance of modern communication systems. These simulations can include the effects of complex processing such as FEC and impairments such as fading.

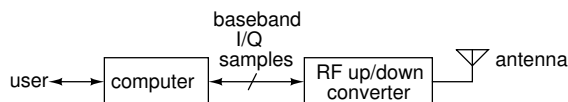
Simulations of communication systems operate on sampled signals. For example, we can generate random bits and random noise samples and apply the same operations on these signals that an actual transmitter, channel and receiver would<sup>1</sup>. Then we can measure the performance (e.g. the BER) of the simulated system.

Such computer simulations are typically written in a programming language such as C or Matlab using libraries of functions that perform various signal processing functions such as signal and noise generation, filtering, modulation, coding, etc.

In this lab you will simulate the operation of a communication system that uses bipolar NRZ signalling over an AWGN channel together with a rate-1/2 convolutional encoder and a Viterbi decoder. You will compare the BER vs.  $E_b/N_0$  curves for systems with and without FEC and determine the coding gain at different BERs.

### GNU Radio

A sufficiently-fast computer can be used to implement the signal processing for wireless communications. This is known as “Software Defined Radio” (SDR).



GNU Radio is an open-source project to develop software for SDR systems. GNU Radio software can also be used to simulate communication systems – which is how we will use it in this lab.

<sup>1</sup>These simulations are sometimes called “Monte-Carlo” simulations because they use randomly-chosen data.

### GNU Radio Companion

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 set up and run simulations.

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

A GNU Radio flowgraph can also send/receive samples to/from radio hardware, files, network connections and other GNU Radio flowgraphs<sup>2</sup>, although we will not use these features in this lab.

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, experienced users tend to prefer programming languages such as Matlab (e.g. using the [Communications Toolbox](#)), because they are more flexible.

### Dataflow Flowgraphs

It may appear from the flowgraph that the blocks operate simultaneously (in parallel). However, since the blocks are implemented in software they must execute sequentially. For efficiency reasons there are buffers between blocks that allow blocks to process multiple samples each time the block is executed. 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.

This “dataflow” architecture means that blocks only execute when they have data to process. As a result we can compare the first bit output by the decoder with the first bit input to the encoder without

<sup>2</sup>Including those written in programming languages such as Python, C++ and Matlab.

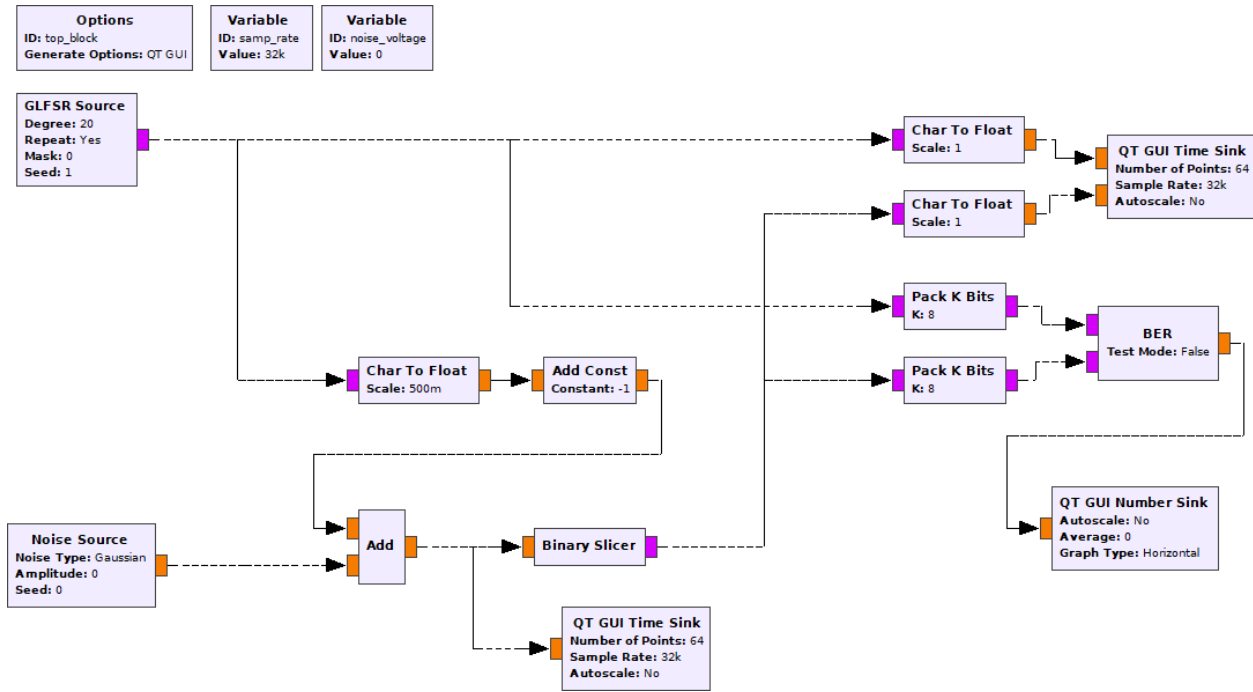


Figure 1: GNU Radio Companion flowgraph for bipolar NRZ over AWGN channel without FEC.

having to compensate for the processing delay of the blocks.

Note that this buffering of samples increases latency and can prevent proper operation of certain delay-sensitive applications such as those using feedback control.

## $E_b/N_0$

$E_b/N_0$  is the “metric” (numerical value) used to compare the power efficiency of communication systems operating over AWGN channels. It is the ratio of the energy per information bit<sup>3</sup>, ( $E_b$ ) to the noise power spectral density ( $N_0$ ).

Comparing systems on the basis of  $E_b/N_0$  rather than the signal-to-noise power ratio (SNR) allows us to compare systems that may transmit at different rates (e.g. due to using FEC or multi-level signalling) or that use different bandwidths (e.g. due to using different line codes or modulation).

Assume that both the signal and noise are strictly bandlimited to a bandwidth  $B$  and that we sample at the Nyquist rate  $f_s = 2B$ . We can convert the signal power  $S$  to bit energy  $E_b$  by multiplying by the information bit period or by dividing by the information

bit rate  $f_b$ . We can compute the noise power spectral density,  $N_0$  by dividing the noise power,  $N$ , by the bandwidth  $B = f_s/2$ . Thus:

$$\frac{E_b}{N_0} = \frac{S}{N} \cdot \frac{f_s}{2f_b}$$

With one sample per information bit ( $f_b = f_s$ ),  $E_b/N_0$  is half of the SNR ( $-3$  dB). When using a rate- $1/2$  code the information bit rate ( $f_b$ ) is halved and  $E_b/N_0$  is equal to the SNR.

For bipolar NRZ with signal levels of  $\pm 1$   $S = 1$  and for AWGN  $N = \sigma^2$ .

## Procedure

You will build two simulation flowgraphs. The first generates a PRBS, converts the 0,1-valued bits to  $\pm 1$ , adds zero-mean AWGN with a configurable standard deviation ( $\sigma$ , the RMS voltage), applies a threshold (at 0 V) to the sum of the signal plus noise, and compares the transmitted and received bits to measure the error rate. The diagram in figure 1 shows the flowgraph.

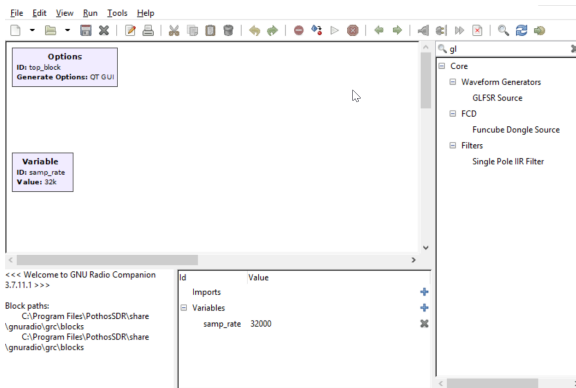
The second flowgraph adds a convolutional FEC encoder before the channel and a Viterbi FEC decoder after the channel. The decoder corrects some errors thus reducing the BER. This provides a “coding gain” as explained in the lectures.

<sup>3</sup>Information bits do not include the parity bits used for FEC.

## 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:



then type part of the block name and double-click on the desired block name. You can move a block by dragging it (any connections will follow).

To connect 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 bytes, violet for float).

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. Another example is the FEC encoder and decoder definition

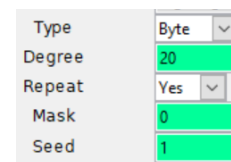
blocks. These set up data structures that define the operation of separate encoder and decoder blocks.

## Configure the First Flowgraph

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

**Variable** Add a variable with ID **noise\_voltage**. This will be used to set the RMS voltage (which is the standard deviation,  $\sigma$ ) of the noise.

**GLFSR Source** This block generates a PRBS. The implementation is different than that described in the lecture notes to allow more efficient implementation in software, but the PRBS has the same properties. Configure this block as follows:

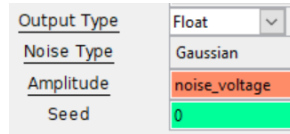


**Degree** is  $N$ , the number of bits in the shift register. The sequence repeats after  $2^N - 1$  bits. Use a value that is large enough to test the blocks in your simulation. The value used here is  $N = 20$  which will generate a PRBS of about 1 million bits and whose longest run of 1's is 20 bits. **Mask** defines the LFSR taps. If left as 0, a suitable generator polynomial for the specified degree will be chosen.

**Type Conversions** Various conversions from bits to floating point values will be required. For example, the **GLFSR Source** is configured to output bits (stored in the least-significant bit of each byte) but these need to be converted to Float values for transmission over an AWGN channel. Add a **Char To Float** block wherever this conversion is required.

Since the input to the BER counter is packed bytes (8 bits/byte), a block is required to pack 8 1-bit values to one 8-bit byte. Add and configure **Pack K Bits** blocks with  $K = 8$ .

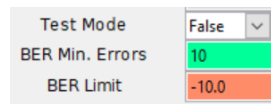
**Noise Source** Add a Gaussian noise source with a Float output whose amplitude is set by the variable `noise_voltage`. It is configured as follows:



**AWGN channel** The AWGN channel is composed of three blocks: **Char To Float** converts 0/1 values to Float values and scales by  $\frac{1}{2}$  to produce values of 0 and 2 (the scaling value is in the denominator). The **Add Const** adds -1 to produce values of -1 and 1. The **Add** block adds the noise to the signal. These blocks should have types set to Float and Vec(tor) lengths of 1 sample.

**Binary Slicer** This block compares the input to a threshold at 0 and outputs bits that are 0 or 1.

**BER** This block compares sequences of byte inputs and computes the number of bits that differ. The output is the log of the bit error rate as a floating point number. The computation includes all bits since the start of the simulation.



**QT GUI Time Sink** This block is similar to an oscilloscope and displays a graph of the input sample values as a function of time. These can be used to verify that the simulation is operating as expected. Two of these are included in the flowgraph below. One displays the output of the channel (signal plus noise). The second displays the bits into the transmitter and out of the receiver.

**QT GUI Number Sink** This block displays a single number while the simulation is running. It can show the value as bar graph.

Both 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 within tabs. For example, a hint of [ 1 , 2 ] means to place that GUI element on the first row, second column.

### Measure the BER

Double-click on the `noise_voltage` block or that line on the variables sub-window and set the noise RMS voltage to the first value in your spreadsheet.

Select Run ► Execute, press F6, or press on the run (▶) icon. This will convert the flowgraph to a Python script and run it.

The GUI will show the bits at the input to the transmitter, the output of the receiver and the output of the channel. The GUI Number Sink will display  $\log_{10}$  of the BER.

Wait until the BER appears stable and record the value. Take a screen capture of the GUI window, then close the GUI window to stop the simulation.

Repeat the steps above, except for the screen capture, for the other  $E_b/N_0$  values in your spreadsheet and verify that the results match your predictions.

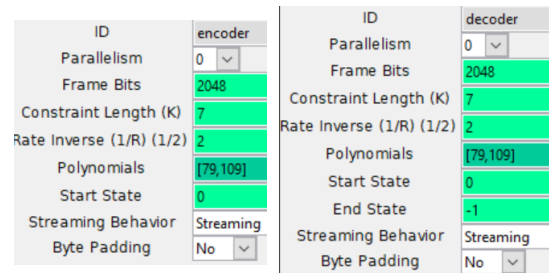
Select File ► Screen Capture to save an image file showing your flowgraph.

### Add FEC

Save and copy the `.grc` file to a new file and modify the flowgraph by adding the following FEC components as shown in Figure 2.

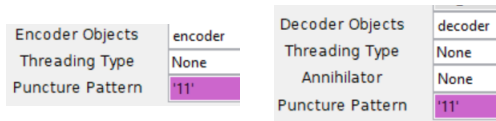
**CC Encoder/Decoder Definition** These blocks define the convolutional code used by the encoder and decoder. You can define constraint length ( $K$ ), rate ( $R$ ) and the generator polynomials used to generate the two output bits. The current implementation is limited to the standard rate- $\frac{1}{2}$  convolutional encoder described in the lecture notes.

Configure these blocks as follows:



The mode should be set to “streaming” (the other modes are for packet-based systems where the shift-register needs to be flushed).

**FEC Extended Encoder/Decoder** These blocks perform the encoding and decoding as configured by the encoder and decoder definitions. The encoder has unpacked bits as input and output. The decoder expects float input ( $\pm 1$ ) and generates unpacked bits as output. Configure these blocks as follows:

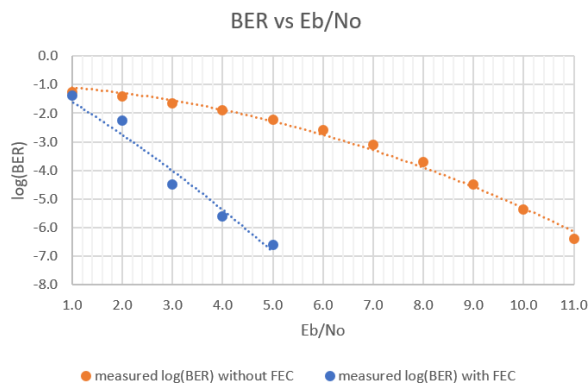


### Re-Measure the BER

Re-measure the BER vs  $E_b/N_0$  and add the data to your spreadsheet.

Select File ► Screen Capture to get an image file showing your second flowchart.

Plot BER vs  $E_b/N_0$  with and without FEC on the same graph. Your graph might look something like:



### Lab Report

Prepare a spreadsheet showing:

- The signal RMS voltage: 1 for NRZ with levels of  $\pm 1$ . In this case  $\mu = \pm 1$  since  $\sqrt{1^2} = \sqrt{-1^2} = 1$ .
- The noise RMS voltage:  $\sigma$ , the setting of **noise\_voltage**.
- The SNR in dB:  $20 \log\left(\frac{\mu}{\sigma}\right)$ . You can also specify the SNR and compute  $\sigma$  from the SNR as was done here.

- The value of  $E_b/N_0$  without rate-1/2 FEC. The conversion from SNR to  $E_b/N_0$  is described in the introduction. Note that the  $E_b/N_0$  with rate-1/2 FEC is numerically equal to the SNR (column C).
- The log of the *predicted* BER without FEC as a function of SNR. You can use spreadsheet functions  $\log_{10}(\text{normdist}(0,\mu,\sigma,1))$  to compute the log of the BER.
- The log of the *measured* error rate –  $\log_{10}(\text{BER})$  – without FEC.
- The log of the *measured* error rate –  $\log_{10}(\text{BER})$  – *with* FEC<sup>4</sup>.

Include values of SNR in your spreadsheet from about 0 to 14 dB<sup>5</sup>.

	A	B	C	D	E	F	G
1	signal voltage	noise voltage	SNR and $E_b/N_0$ with FEC	$E_b/N_0$ without FEC	predicted $\log(\text{BER})$ without FEC	measured $\log(\text{BER})$ without FEC	measured $\log(\text{BER})$ with FEC
2	$\mu$	$\sigma$	$t$				
3	Vrms	Vrms	dB	dB			
4	1	1.000	0.0	-3.0	-0.8		
5	1	0.891	1.0	-2.0	-0.9		
6	1	0.794	2.0	-1.0	-1.0		
7	1	0.708	3.0	0.0	-1.1		
8	1	0.631	4.0	1.0	-1.2		
9	1	0.562	5.0	2.0	-1.4		
10	1	0.501	6.0	3.0	-1.6		
11	1	0.447	7.0	4.0	-1.9		
12	1	0.398	8.0	5.0	-2.2		
13	1	0.355	9.0	6.0	-2.6		
14	1	0.316	10.0	7.0	-3.1		
15	1	0.282	11.0	8.0	-3.7		
16	1	0.251	12.0	9.0	-4.5		
17	1	0.224	13.0	10.0	-5.4		
18	1	0.200	14.0	11.0	-6.6		

Plot the predicted and measured BER without FEC and the measured BER with FEC against  $E_b/N_0$ .

Use the graph to estimate the coding gains at BERs of  $10^{-2}$  and  $10^{-6}$ .

Submit a report in PDF format to the appropriate dropbox on the course web site containing the following:

- your two flowchart diagrams
- your screen capture of the run-time GUI window showing the signal on the channel, the input and output bits and the  $\log_{10}(\text{BER})$  value

<sup>4</sup>There is no close-form formula for the BER with FEC.

<sup>5</sup>At lower SNR's the BER will remain at 0.5, at higher SNR's the error rate will be too low to measure.

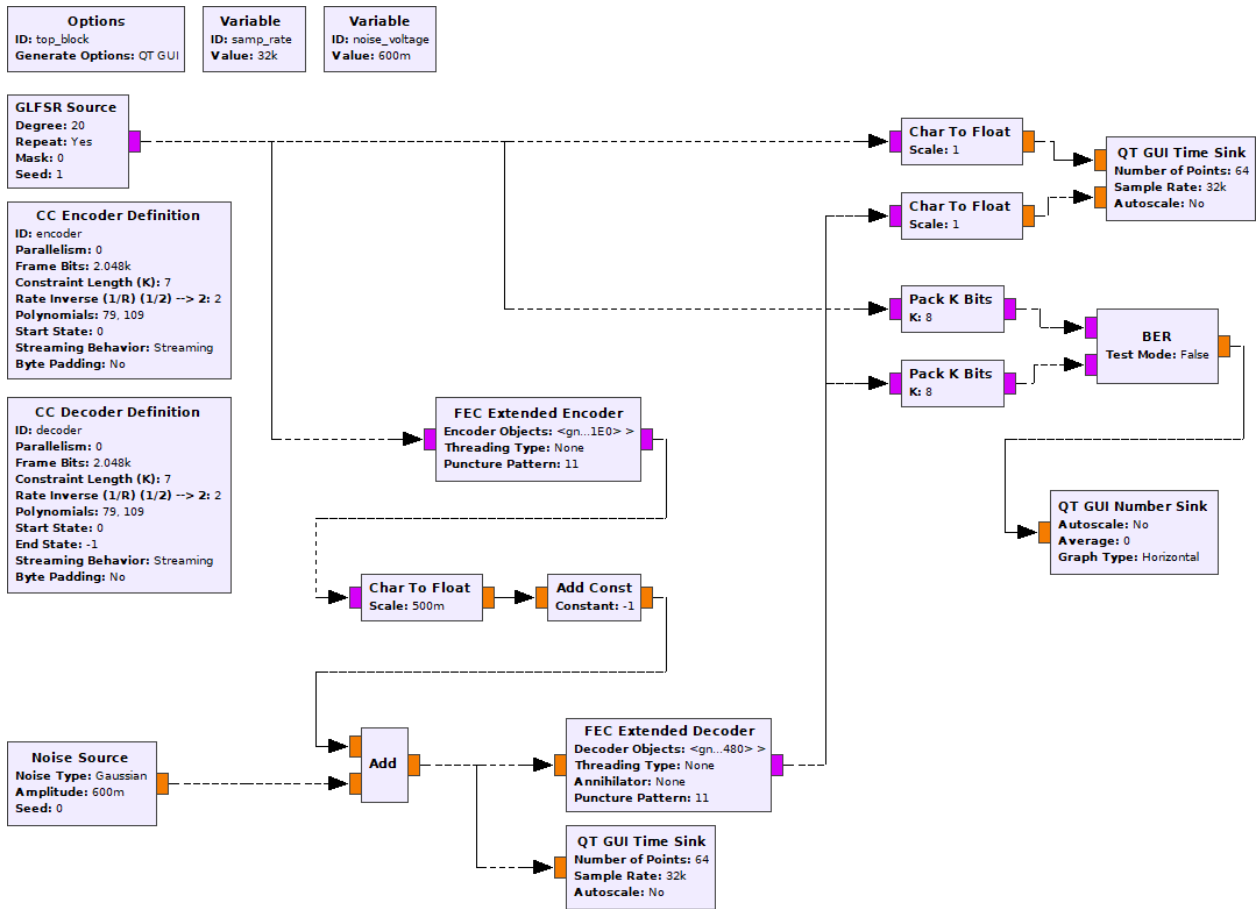


Figure 2: GNU Radio flowgraph for bipolar NRZ over AWGN channel with convolutional FEC code and Viterbi decoder.

- the spreadsheet showing your data
- the plot with two curves showing BER vs  $E_b/N_0$  with and without FEC
- your calculations of the coding gains for BERs of  $10^{-2}$  and  $10^{-6}$ .