Final Project: Transmitters and Receivers

Part A: Receiver Side (40%)

Submission deadline. Wednesday April 28, 11:59PM EST.

1 Introduction

For the final project, you will implement a functional receiver and transmitter in GNU Radio. The goal is to successfully transmit and receive a text file. You will use channel emulation to test the functionality of the full system. Note that the code that you will be developing can be transferred directly to SDRs with transmit/receive capabilities.
In Part A of the project, you will implement the receiver side. There is one Python script you need to download: **DecoderTemplate.py**, which will assist you in decoding the message.
In this part, there are *two milestones* to validate whether your flowgraph and python script works as intended. You will need to download the following files:

- **M1_DecoderScriptCheck.bin**
- **M2_ReceiverFlowgraphCheck.iq**

1.1 Deliverables

Your group must submit the following deliverables on Blackboard (**3 files in total**):

1. 1 grc file for your receiver.
2. 1 Python Script for DecoderTemplate.py
3. A PDF file containing your solutions. Call this file `project_GNU_Part_A.pdf`. You will also need to explain how your flow graph works in the PDF. Include screenshots of any relevant information that shows successful message reception. **Please briefly explain at the beginning of the PDF file how each student contributed.**

As usual, do not hesitate to contact the lab assistants (Josh: joshuaat@bu.edu, John: jkulskis@bu.edu) or go to office hours to ask for help.

2 Grade Breakdown

We will grade this part of the project with the following weights:

- 15 points for your GRC flowgraph.
- 15 points for your Python Script.
- 10 points for your solution to Question 1.

3 Procedure

The project is split into three main parts:

1. Building the Receiver.
2. Building the Transmitter (to be completed in Part B).
3. Building the Transceiver (to be completed in Part B).

3.1 Building a Receiver

The first component of the final project is very similar to Lab 5. You will be creating a receiver that takes in an IQ file and outputs a binary file. Once the binary file is produced, you will then run your python decoder file to check if the message parsed is the one we expect.

3.1.1 Modulation Scheme

For the receiver, you will be using **GFSK** as your modulation scheme. You may repurpose your receiver and Python script from Lab 5. The provided IQ test file is already modulated with GFSK so that you can verify that your receiver implementation works properly.

3.1.2 Creating your Flow Graph

To demodulate the signal, the steps are identical as in the GFSK flowgraph for Lab 5. You may repurpose your old flowgraph and make modifications to it as necessary or create a new one from scratch following those instructions. Below are the general variables you will be using. These variables are nearly identical to Lab 5. **The main difference here are: (i) the `filter_cutoff` variable, which you need to change to 10000; (ii) the `samples_per_symbol` which you need to change to 20.** Ensure you update your flowgraph appropriately.

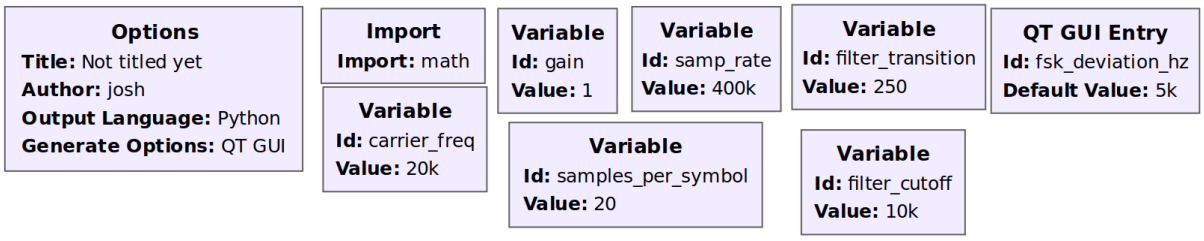

Figure 1: General Variables Used

3.1.3 Receiver Side: Python Script

You will implement a Python Script similar to the one you programmed in Lab 5. Build upon the downloaded **DecoderTemplate.py** to parse the binary output of the GRC flowgraph. You may transfer parts of your Python script from Lab 5 to this template. The script should:

1. Search for a preamble
2. Get the message length
3. Decode the message into ASCII
4. Print the message
5. Check the CRC
6. Repeat from step 1 until the entire file is processed.

3.2 Introducing Error Detection

One of the most common ways of detecting errors is using a CRC code. CRC stands for Cyclic Redundancy Check and is a effective way of checking for bit errors, while making it extremely unlikely that a false negative occurs (i.e. an error happened but was not detected).

Python provides a library to generate CRC codes for us, so we will add this to our packet to check if errors occurred during transmission.

3.2.1 Modifying the Receiver Python Script

You will first need to modify your receiver script to properly decode and check the CRC code. First, consume another 4 bytes after you get the message bytes. This is the CRC that was transmitted with the packet. Then, using the helper function 'check_crc', determine if the packet had any transmission errors. Note: the function takes an ASCII string, not a hex-encoded string.

	Preamble	**Payload Length**	**Payload**	**CRC**
Length	4 Bytes	1 Byte	Variable size	4 bytes
Value	0xAABBCCDD	0 to 255 (0x00 to 0xff)	Hex Encoded ASCII String	Raw Bytes

The test file **M1_DecoderScriptCheck.bin** already contains the CRC in the packets. Use these to test your implementation. You should see the function return True if the message has no errors and False if it does.

3.2.2 Verifying your Python Decoder Script

To check if you wrote **DecoderTemplate.py** correctly, run this python script on the provided file **M1_DecoderScriptCheck.bin**. You should see the following output:

Milestone 1: Checking if Receiver Script works!

3.2.3 Verifying your Receiver Flowgraph

To verify your receiver works as intended, you will be using files **DecoderTemplate.py** and **M2_ReceiverFlowgraphCheck.iq**.

1. Put M2_ReceiverFlowgraphCheck.iq as the *File* parameter in the `File Source` block
2. Execute your flowgraph.
3. Run your verified **DecoderTemplate.py** on the output binary file, in which you should name **decodertest.bin** You should see the following message:

   Milestone 2: Checking if Receiver Flowgraph works!

Note: Since the transmitter likely repeats the message, and if your receiver also repeats the message, you might get the same output several times, this is totally fine. It is also possible that some of these will be corrupted since there is a chance of corruption during file write.

Question 1. What is the value of the CRC in the packet found in **decodertest.bin**? Please answer in hex format.