Final Project: Transmitters and Receivers

Part B: Transmitter Side (60%)

Submission deadline. Friday April 30, 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 B of the project, you will implement the transmitter side. There is one Python script you need to download: **EncoderTemplate.py**, which will assist you in encoding the message.
In this part, there are *two milestones* to validate whether your flowgraph and python script work as intended. You will need to download the following files:

- **M3_EncoderScriptCheck.bin**
- **M4_TransmitterFlowgraphCheck.bin**

1.1 Deliverables

Your group must submit the following deliverables on Blackboard (**4 files in total**):

1. 2 grc files: one for your transmitter and one for your transceiver.
2. 1 Python Script for EncoderTemplate.py
3. A PDF file containing your solutions. Call this file `project_GNU_Part_B.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:

- 20 points for your GRC flowgraphs.
- 15 points for your Python Script.
- 25 points for your solutions to the Questions.

3 Procedure

The project is split into three main parts:

1. Building the Receiver (completed in Part A).
2. Building the Transmitter.
3. Building the Transceiver.

3.1 Building the Transmitter

You now need to create the transmitter part of the communication system. This consists of a script that takes a message and formats it into a binary packet, and a GRC flowgraph that will modulate the packet and transmit it. In this part of the final project, this "transmission" will be stored in an IQ file. For now, do not implement the transmitter in the same flowgraph as the receiver. **Create a separate flowgraph.**

3.1.1 Python Script

Using the **EncoderTemplate.py** template, create a python script that reads an input from the user and creates a packet with the following protocol:

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

There are several helper functions that will assist you in assembling your packet.

Note: The script takes the output file name as a command line argument: `$ python EncodeTemplate.py OUTPUT_FILE`

3.1.2 Transmitter Modification

Your transmitter script must also generate and include the CRC in the packet. Using the helper function, generate a CRC for the payload. Ensure that your input to the function is the ASCII message you are sending, not the hex encoded version. The CRC should be appended to the **end** of your packet.

3.1.3 Verifying your Python Encoder Script

You will verify **EncoderTemplate.py** by comparing the provided M3_EncoderScriptCheck.bin to the binary generated from your EncoderTemplate.py using a predetermined input. This should produce a new file, which you should name **CheckEncode.bin**. When you run EncoderTemplate.py and the terminal prompts you to enter some text, type in the following **exactly as shown**

Milestone 3: Checking Encoder!

To check if your produced binary is as expected, run the following command in the terminal line:

```
diff -s M3_EncoderScriptCheck.bin CheckEncode.bin
```

diff stands for difference. This command is used to display the differences in the files by comparing the files line by line. The "-s" parameter will tell you if the files are the same. You should see the following output after you run the **diff** command:

```
Files M3_EncoderScriptCheck.bin and CheckEncode.bin are identical
```

3.1.4 GRC Transmission Flowgraph

Create a flowgraph that reads from the binary file your script creates using a `File Source`. Use a `GFSK Mod` block to modulate the message into GFSK.

Ensure that all your parameters match your receiver. The baud rate, carrier frequency, etc must match for communication to occur successfully. Additionally, there are a few parameters in the GFSK Demod and Mod blocks that must be set appropriately. The relationship of those parameters is shown below:

GFSK Parameter	**Relationship**
Samples/Symbol	$GFSKDemod = GFSKMod$
Sensitivity	$GFSKDemod = 1/GFSKMod$

Hint: For the transmitter, modulate your baseband signal to produce the passband signal.

Question 2. What is the baud rate that your packet is being transmitted at?

Remember: the units for baud rate is symbols per second. All the values needed to calculate this can be found in your flowgraph.

3.1.5 Verifying your Transmitter Flowgraph

You will be using the last downloaded file, **M4_TransmitterFlowgraphCheck.bin**, to verify your transmitter flowgraph. Here are the following steps:

1. In your transmitter flowgraph, put **M4_TransmitterFlowgraphCheck.bin** in the *File* parameter in the `File Source` block. Your transmitter flowgraph should output an IQ file, which you can call **TransmitTest_pt1.iq**.

2. Pull up your receiver flowgraph. Put **TransmitTest_pt1.iq** in the *File* parameter in the `File Source` block. Your receiver flowgraph should output a binary file, which you can call **TransmitTest_pt2.bin**.

3. Run your **DecoderTemplate.py** script on **TransmitTest_pt2.bin**. You should see the following output:

   Milestone 4: Checking Transmitter flowgraph!

3.2 Building the Transceiver

In the final part of the final project, you will be combining both flowgraphs into one to mimic a live transceiver. To do this, copy each of your receiver and transmitter flowgraphs into a new flowgraph, deleting any duplicate variables. To connect the two, simply remove the `File Source` block from your receiver and remove the `File Sink` from your transmitter. Replace these with a `Channel Model` and connect the output of the transmitter to the input of the `Channel Model` and the output of the `Channel Model` to the input of the receiver. This way you can change the noise level and test communication with bit errors.

Note: When you copy and paste entire flowgraphs, pay careful attention to all blocks copied over to make sure there is no overlap and that the flowgraph can still be executed.

Note 2: Use the default values of the channel model first where *Noise Voltage* is set to 0.

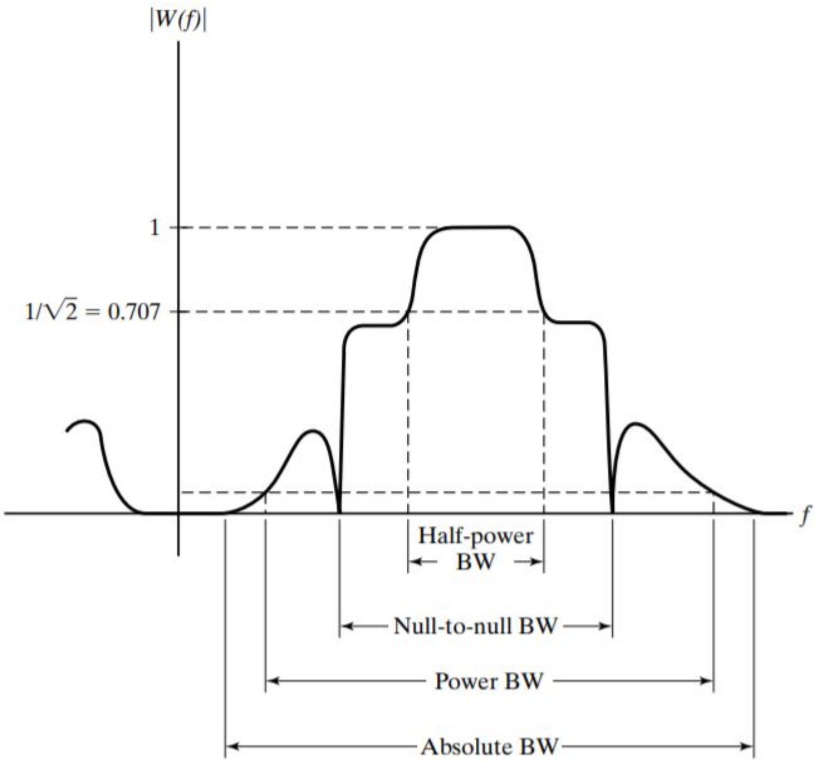

Figure 1: Types of Bandwidth Definitions

Question 3. What is the half-power bandwidth of your signal? You can estimate this through a `QT GUI Frequency Sink` connected to the output of the *Channel Model*.

Note: Refer to Figure 1 to see an illustration of half-power bandwidth.

Question 4. What is the minimum noise voltage you can set your `Channel Model` block to where you will not receive any correctly parsed messages?