Digital Wireless Communication Practical Laboratory Session

WIRELESS COMMUNICATIONS 371-1-1903

Part 1 – Send *"Alice,"* receive *"Bob"*

Description

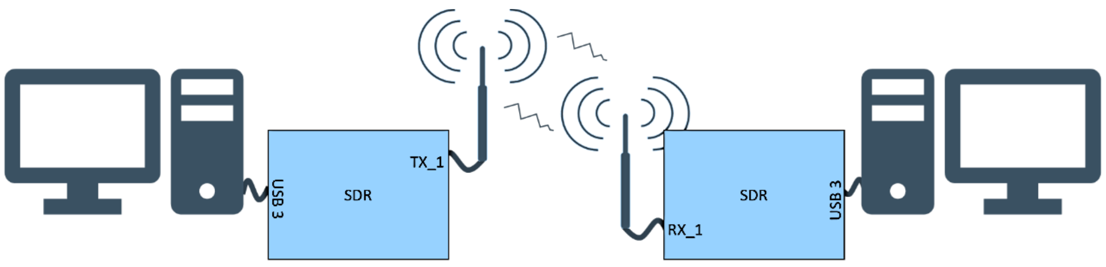
In this experiment, we will construct a communication system using the GNU Radio freeware to upgrade the ASK concept you created in the theoretical session. If you haven't performed the theoretical session, you will find it difficult to perform this practical session.

Equipment needed

- 2 Linux PC with GNU Radio installed.
- 2 LimeSDR/USRP Software defined radios.
- 2 SMA to SMA RF cables
- 2 ULF-SMA Adapters.
- Attenuators box.

If any of the above equipment is missing or defective, notify the lab instructor before starting this session, or otherwise, you may obtain false results (and major frustration).

Equipment Setup



Recording

This is a practical session; thus, you most likely do not have this equipment at your disposal at home. Therefore, documenting and recording your results during this session is highly recommended. This will assist you in completing your report at home. You may only partially complete what is required of you during the time you have; take this into account.

Instructions

1. Create 2 new .grc files, each on its own PC. One will be the transmitter, and the other will be the receiver.

2. Download Alice.txt (Alice In Wonderland textbook) from Moodle.

3. Build the transmitter as follows:

   a. Set the "samp_rate" to $192KHz$.

   b. Add a "File Source" and set the file path to where Alice.txt is. Set Repeat to "Yes".

   c. Add a "Varicode Encoder" and connect its input port to the output port of the "File Source".

      - Varicode is a self-synchronizing code used for text compression. It converts Byte stream to Binary stream and supports all ASCII characters, but the characters used most frequently in English have shorter codes. The space between characters is indicated by a 00 sequence, an implementation of Fibonacci coding. Originally created for speeding up real-time keyboard-to-keyboard exchanges over low bandwidth links.

   d. Add a "Chunks to Symbols", and set the symbol table: "$0{,}1$" with dimensions: "1". Connect its input port to the output port of the "Varicode Encoder".

   e. Add the "Import" component and enter "$import\ numpy$" in the Import field.

      - This allows us to use Python libraries in our code.

   f. We need a convolution FIR filter. Add "Interpolating FIR filter" and set the Interpolation to: 20. To set a filter characteristic, insert the following in the Taps field:

      ```
      numpy.convolve(numpy.array(filter.firdes.gaussian(1, 20, 1.0, 4*20)),numpy.array((1,) * 20))
      ```

      Connect its input port to the output port of the "Chunks to Symbols" component.

   g. Add "Multiply const" with a constant of 0.9. Connect its input port to the output port of the "Interpolating FIR filter" component.

   h. Add a "Rational Resampler" for changing the sample rate by 10. Enter an Interpolation value of 10. Connect its input port to the output port of the "Multiply const" component.

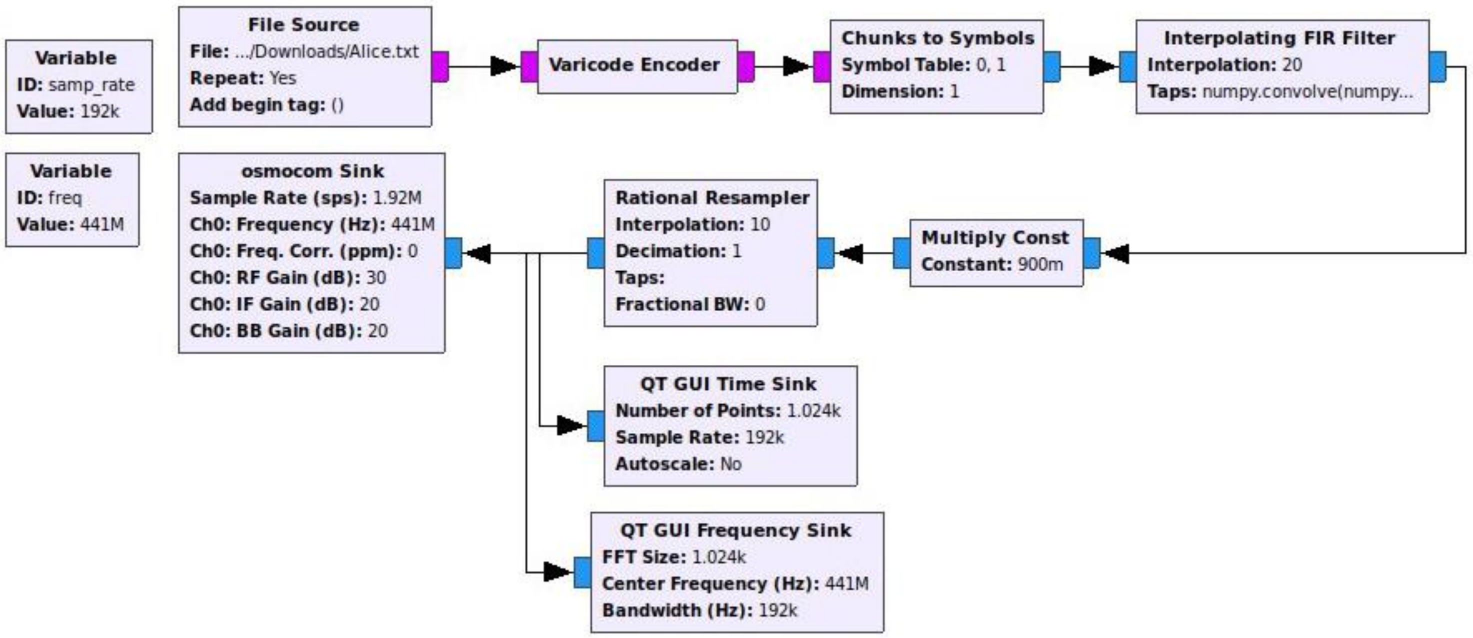
   i. To the "Rational Resampler" connect a "QT GUI Time Sink" and a "QT GUI Frequency Sink" (both with the control panel set to "Yes")

j. Create a Variable for the carrier frequency. Set it to:

400MHz+"your pair number"*20MHz

k. Add an "osmocom Sink" and set its frequency as your defined variable. Set the sample rate to 10*samp_rate. Set the gain to 30dB. Connect it to the output port of the "Rational Resampler" component.

If you performed all the stages correctly, your system's code should look like this:



Variable
ID: samp_rate
Value: 192k

Variable
ID: freq
Value: 441M

File Source
File: .../Downloads/Alice.txt
Repeat: Yes
Add begin tag: ()

Varicode Encoder

Chunks to Symbols
Symbol Table: 0, 1
Dimension: 1

Interpolating FIR Filter
Interpolation: 20
Taps: numpy.convolve(numpy...

Multiply Const
Constant: 900m

Rational Resampler
Interpolation: 10
Decimation: 1
Taps:
Fractional BW: 0

osmocom Sink
Sample Rate (sps): 1.92M
Ch0: Frequency (Hz): 441M
Ch0: Freq. Corr. (ppm): 0
Ch0: RF Gain (dB): 30
Ch0: IF Gain (dB): 20
Ch0: BB Gain (dB): 20

QT GUI Time Sink
Number of Points: 1.024k
Sample Rate: 192k
Autoscale: No

QT GUI Frequency Sink
FFT Size: 1.024k
Center Frequency (Hz): 441M
Bandwidth (Hz): 192k

The modulation itself takes place in the SDR (by mixing the signal generated in GNU Radio with the carrier signal – field "CH0 Frequency" in OsmocomSink)

4. Now, Build the receiver as follows:

   a. Set the “samp_rate” to $1.2MHz$.

   b. Create a Variable for the carrier frequency. Set it like your transmitter:

      $$400\text{MHz}+\text{"your pair number"}*20\text{MHz}$$

   c. Add a “QT GUI Range” for the gain, from 0dB to 70dB in steps of 1dB and with a default value of $50dB$.

   d. Add an “osmocom Source” and set its frequency and RF gain as your defined variable and range accordingly.

   e. Add a “Low Pass Filter” and set its properties as follows:

      - Decimation: 5
      - Gain: 1
      - Cutoff Frequency: 50KHz
      - Transition Width: 25KHz

      Connect its input port to the “osmocom Source” output port.

   f. Add a “Complex to Mag” component and connect its input port to the output port of the “Low Pass Filter” component.

   g. Add a float “DC Blocker” and set the Length to 320.

      - This will ensure that our data is centered at 0V DC so that we can decode it.

      Connect its input port to the output port of the “Complex to Mag” component.

   h. To the “DC Blocker” connect a float “QT GUI Time Sink” and a float “QT GUI Frequency Sink” (both with the control panel set to “Yes”)

   i. Place a float “Clock Recovery MM” - this block is needed to recover a symbol from the input signal (a clock allows us to know what is the symbol duration):

      - in the field “Omega” type $(\text{samp\_rate}/5/9600)$

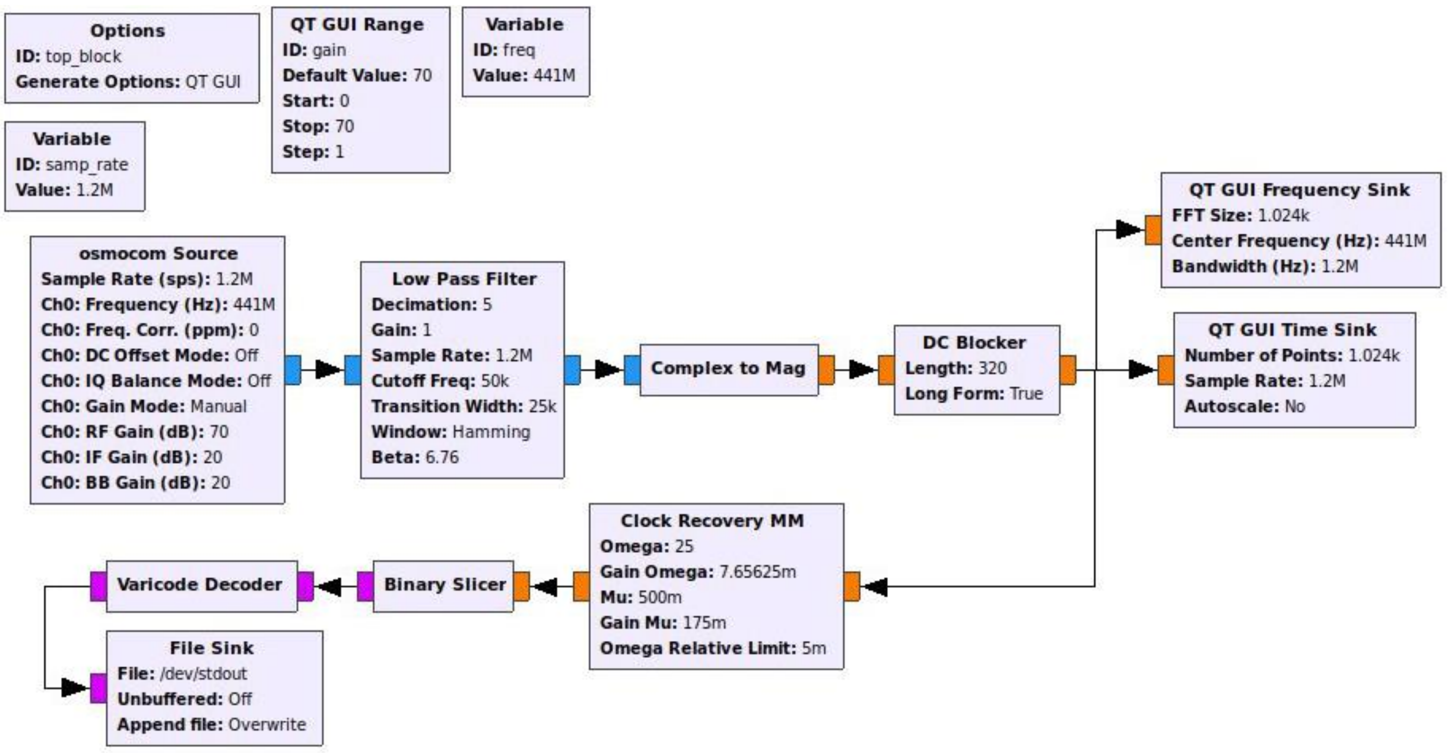
      Connect its input port to the output port of the “DC Blocker” component.

   j. Add a “Binary Slicer” to slice the float signal to a binary stream. Connect its input port to the output port of the “Clock Recovery MM” component.

k. Add a “Varicode Decoder” and connect its input port to the output port of the “Binary Slicer”.

l. Add a “File Sink” and in the file path, write: `/dev/stdout`
This will print the decoded data to the console.

If you performed all the stages correctly, your system’s code should look like this:



Options
ID: top_block
Generate Options: QT GUI

QT GUI Range
ID: gain
Default Value: 70
Start: 0
Stop: 70
Step: 1

Variable
ID: freq
Value: 441M

Variable
ID: samp_rate
Value: 1.2M

osmocom Source
Sample Rate (sps): 1.2M
Ch0: Frequency (Hz): 441M
Ch0: Freq. Corr. (ppm): 0
Ch0: DC Offset Mode: Off
Ch0: IQ Balance Mode: Off
Ch0: Gain Mode: Manual
Ch0: RF Gain (dB): 70
Ch0: IF Gain (dB): 20
Ch0: BB Gain (dB): 20

Low Pass Filter
Decimation: 5
Gain: 1
Sample Rate: 1.2M
Cutoff Freq: 50k
Transition Width: 25k
Window: Hamming
Beta: 6.76

Complex to Mag

DC Blocker
Length: 320
Long Form: True

QT GUI Frequency Sink
FFT Size: 1.024k
Center Frequency (Hz): 441M
Bandwidth (Hz): 1.2M

QT GUI Time Sink
Number of Points: 1.024k
Sample Rate: 1.2M
Autoscale: No

Clock Recovery MM
Omega: 25
Gain Omega: 7.65625m
Mu: 500m
Gain Mu: 175m
Omega Relative Limit: 5m

Binary Slicer

Varicode Decoder

File Sink
File: /dev/stdout
Unbuffered: Off
Append file: Overwrite

Save your code and add it to your submission!

5. Go ahead and run your code. You should see the text being transferred between the computers.

Report

Make sure your system is working correctly before filling this report

Include snapshots and explanations of your results in your submission.

1. **Power vs. distance graph**
   a. Place both antennas closely together. The printed text should be errorless. If it does not, try to calculate the dynamic range, and make sure you are working within these bounds by adjusting the Tx and Rx RF gain.
   b. Measure the distance and the voltage output of the data on the receiver side.
   c. Slowly move the antennas apart and away from each other, measure, and document the voltage of the data at the receiver side.
   d. Repeat several times (at least 5 different distances across a few meters) until you can see some errors in the printed text. If no errors appear, try to reduce some gain.

   Add to your submission a plot of "power vs. distance." Include a short explanation of this phenomenon and your measured results. What is the relation you found? If you haven't found one, try to explain why.

2. **Thresholds**
   a. Find and record the threshold at which errors begin to accrue.
   b. Find and record the threshold in which **NO** data is decoded at all. You can use the gain in your SDRs to achieve sufficient attenuation.
   c. What are the causes of these thresholds (or bounds)? Explain how we can improve these thresholds.

Submit your answers, plots, and screenshots, and explain your results.