Digital Wireless Communication Theoretical Laboratory Session

WIRELESS COMMUNICATIONS 371-1-1903

Part 1 – Theoretical Background

You are given the following reading material:

https://en.wikipedia.org/wiki/Data_transmission
https://en.wikipedia.org/wiki/Single-input_single-output_system
https://en.wikipedia.org/wiki/Modulation#Digital_modulation_methods
https://en.wikipedia.org/wiki/Bit_error_rate
https://en.wikipedia.org/wiki/Minimum_detectable_signal

These sources will help you to both answer the questions and understand the required theoretical background for this lab.

In the following practical session, we will initiate a communication channel of our own and try to transfer data, then we will affect the transmission channel in several ways and see what happens.

Theoretical Questions

1. Why is modulation required? Give several reasons for using modulation.
2. Name and explain the usage of several analog modulation schemes and several digital modulation schemes.
3. Assume a system using OOK modulation. I.e., $X \in \{0,\ A\}$. Each symbol is transmitted with the same probability. Assume an Additive White Gaussian Noise (with a *standard deviation* of $\frac{\sigma}{2}$) is added to the transmitted signal. The receiver obtains the noisy signal and compares it to $\frac{A}{2}$. If the amplitude is greater than $\frac{A}{2}$, the decoder declares $\hat{X} = A$. Otherwise, it declares $\hat{X} = 0$. What is the bit-error-rate (BER) of the system?
4. What is the most dominating factor affecting the BER of a channel? (noise, interference, distortion synchronization, attenuation, fading, or else). *Hint:* What is $\left(\frac{A}{\sigma}\right)^2$?
5. Given a large amount of money that you can only spend on your own system (hardware - receiver and transmitter only, not the channel medium), what would you do to improve your ability to transfer data?

Part 2 – Running GNU Radio

Helpful Links

https://www.gnuradio.org/
https://en.wikipedia.org/wiki/GNU_Radio
https://wiki.gnuradio.org/index.php/Main_Page

Perquisitions

Notice: Linux system is recommended, yet Windows will work as well (but can be buggier). It is your responsibility that you work on a system that is functioning properly and that you achieve the correct results!

To work at home, follow the instructions on how to install GNU Radio on your system, and choose one of the following options:

a. You can use the Virtual Machine uploaded to Moodle, it includes GNU Radio 3.7 with many added modules. Just download the image file and use your favorite VM player (recommendation: VMware workstation Player).

   In the command prompt:
   `pybombs run gnuradio-companion`

b. You can install it on your system by following "gnuradio installation commands" file (based on Pybombs).

c. You can try installing from scratch, which is highly unrecommended:

   https://wiki.gnuradio.org/index.php/InstallingGR

Upon completion, verify that everything is working correctly.

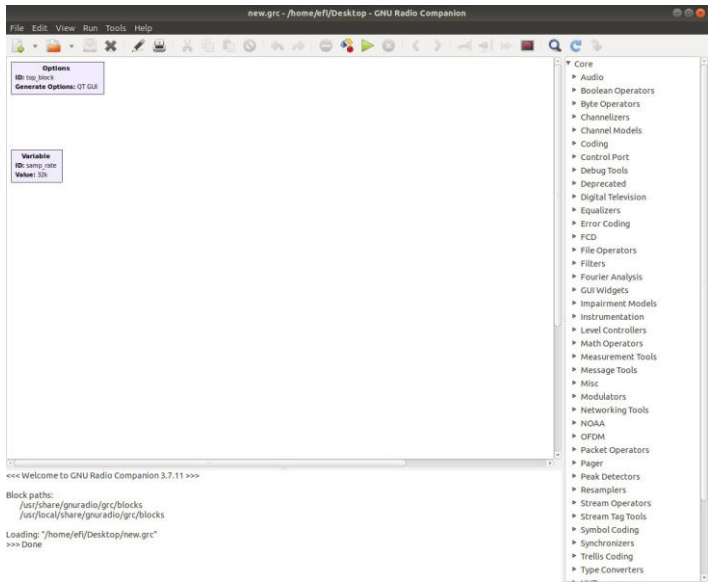
Feel free to read about and explore GNU radio and all of its components and abilities.

Tutorial

In this part we will construct several basic simulations to learn how to use GNU Radio.

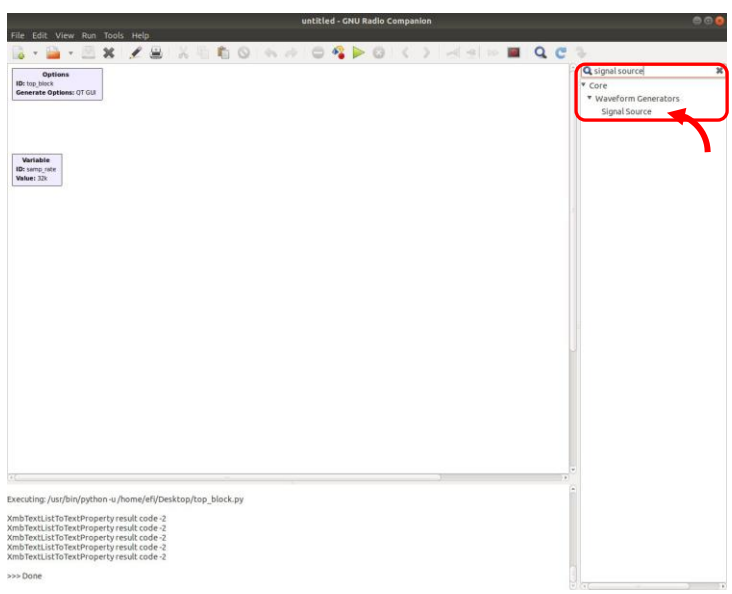
Instructions:

1. Open GNU Radio.



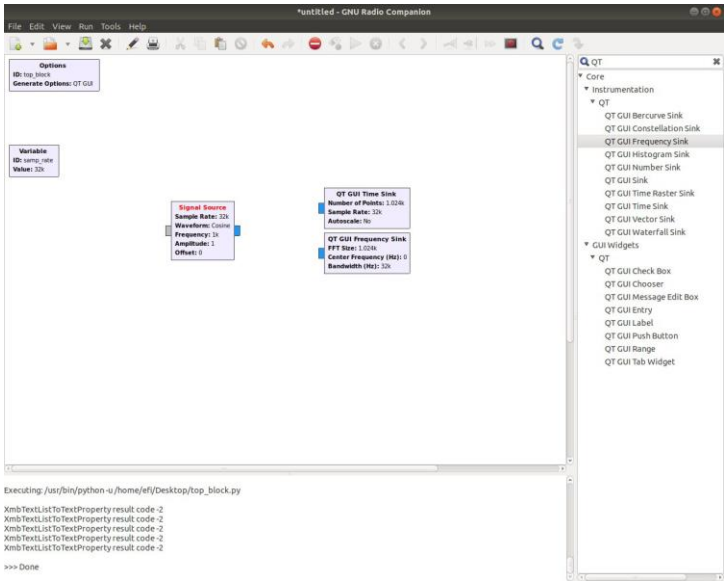
2. Open a new QT GUI file.

3. In the right side of the screen, you can find the components tab. Search for a component named “Signal Source” and place it in your workspace by dragging it.

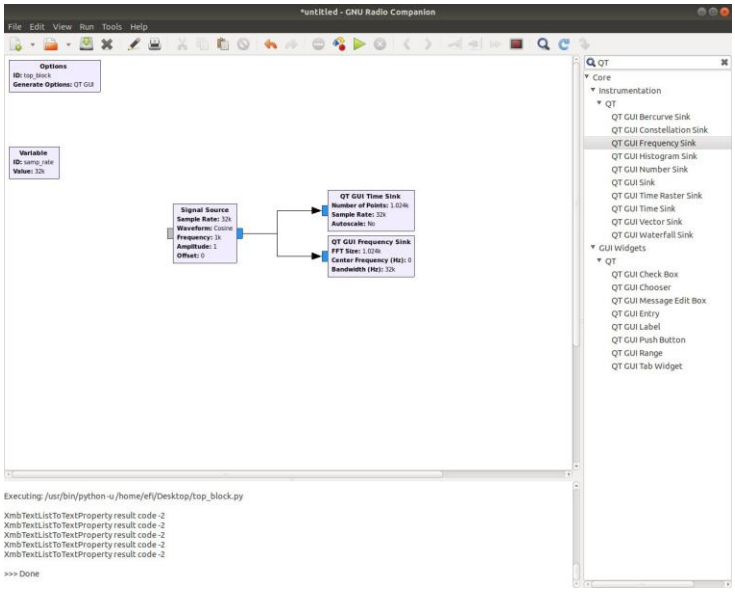


This block simulates a signal source, a sinusoidal function such as $x(t) = Asin(2\pi ft + \varphi)$, Cosine, Square, Triangle and more…

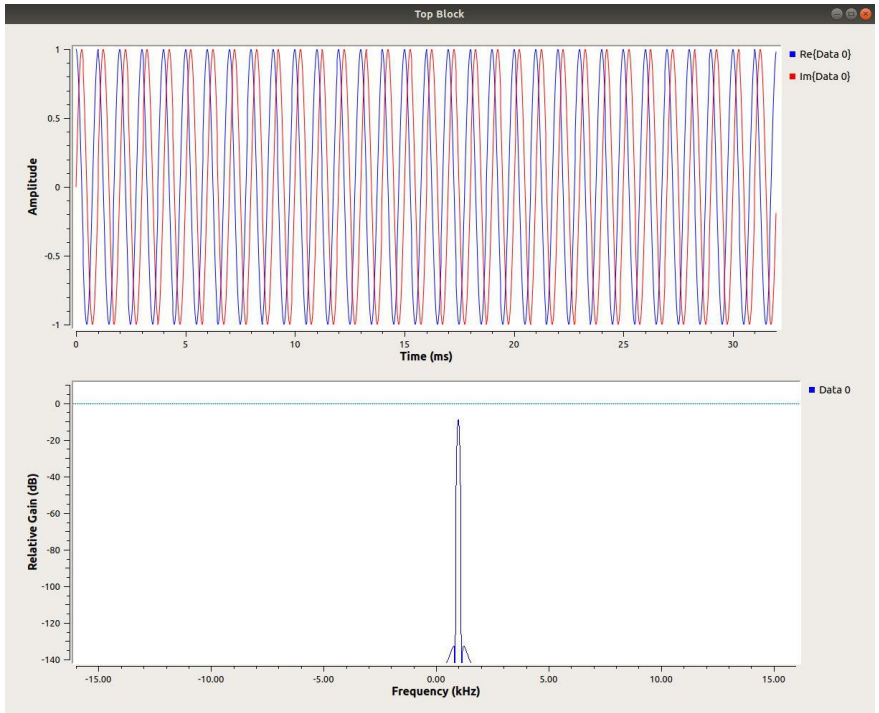
4. Now search and place “QT GUI Time Sink” and “QT GUI Frequency Sink”. These are graphical user interface that allow us to see the signal in the time domain and its spectrum in the frequency domain.



5. Now, let’s connect the signal source and the sinks. Click on the small blue part (out port) of the block that you wish to connect, and then click on the other block’s blue input port. An arrow should appear, this indicates that the components are now connected.
   Do the same for the remaining sink.
   Blue port indicates a complex number input\output, orange indicates a float, you can see all types of color mapping in “help”. Notice, you can only connect ports of the same types.

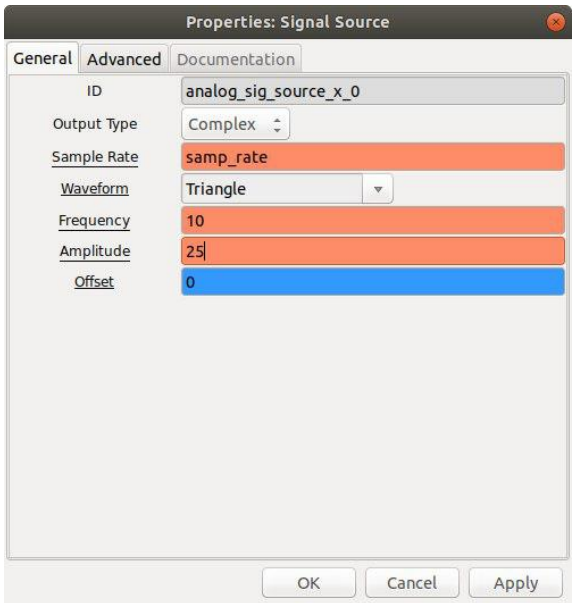


6. Now, go ahead and run the code (remember to save it first).

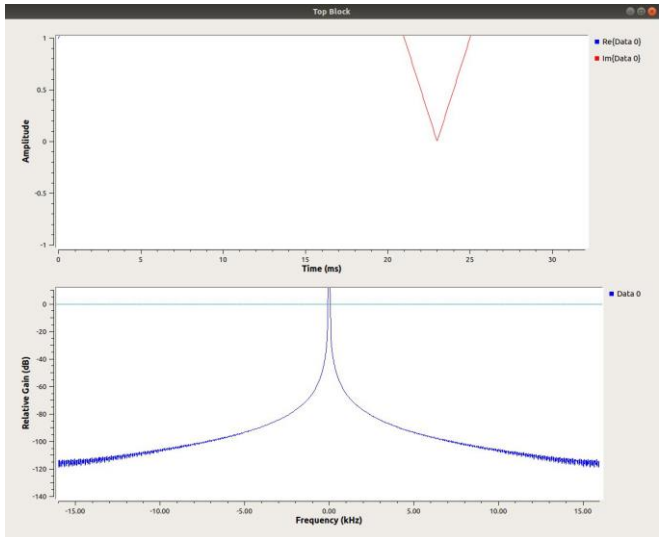


A new window appears which shows the signal amplitude vs time and frequency. Notice that the signal has default values since we did not change any of the component's properties.

7. Stop the running of the code and double click the signal source block. This opens the properties of the component. Choose a Triangle waveform and, change the frequency to 10Hz and the amplitude to 25V.



8. Go ahead and run it again.



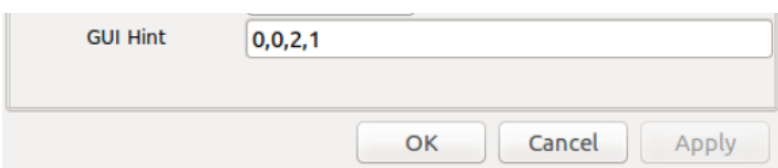
This time, the signal is not displayed properly. Change the properties of the sinks to show the signal.

Tip: Customize the GUI!

The GUI acts like a *matrix* whose entries are (the numbers are arbitrary and can be larger than 3…):

(0,0)	(0,1)	(0,2)	(0,3)
(1,0)	(1,1)	(1,2)	(1,3)
(2,0)	(2,1)	(2,2)	(2,3)

All the QT GUI widgets and plots have a parameter called “GUI Hint”. This is used to arrange GUIs in the window, as well as assign them to tabs in a QT GUI Tab Widget.



The format is: (row, column, row span, column span). For example,

Waveform Selector (0,0,2,1)	Offset Slider (0,1,1,1)
	Frequency Slider (1,1,1,1)
Time Display (2,0,1,1)	Frequency Display (2,1,1,1)

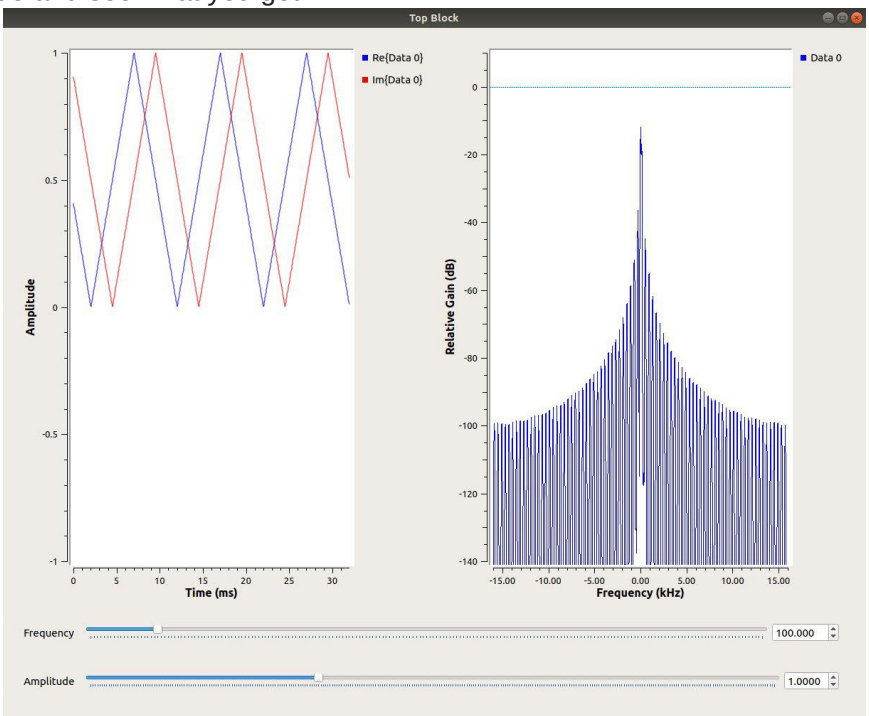
9. Now that you have everything under control, let's take it up a notch.

   Add a "QT Range" block to your code. Edit its properties as such:
   - Change its ID to "Frequency".
   - Change its default value to 100.
   - Change its start to 0.
   - Change its stop to 1e3.
   - Change its step to 0.1.

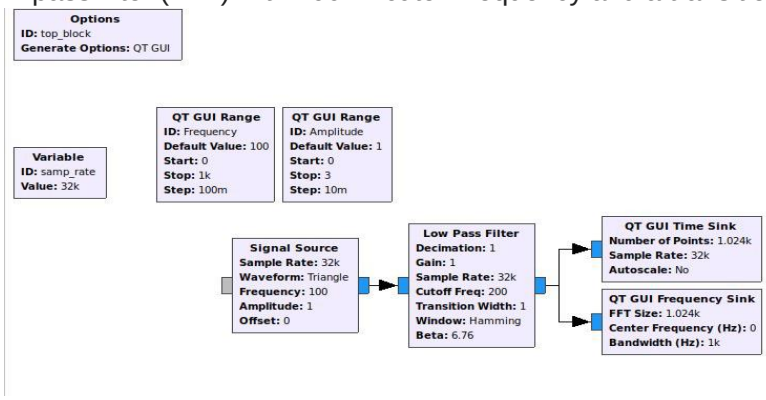
   Now go to the signal source and enter the "Frequency" (name that you gave to the QT Range) in Frequency.

   Create one more range for the amplitude and enter it in the signal properties.

   Run the code and see what you get.



10. Now, add a low pass filter (LPF) with 200Hz cutoff frequency and a transition width of 1Hz.



Run the code and slide to 199Hz. Slowly move to 201Hz. What do you see happening to the signal?

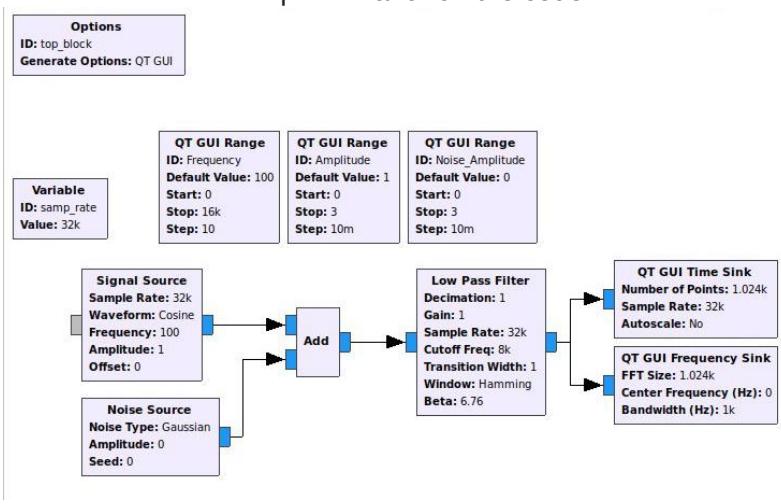
When the LPF's cutoff frequency is lower than the triangle signal's bandwidth, you will see the triangle is losing its "shape" (in other words, distortions). The distortions are due to the filtering of the higher frequency components of the triangle signal.
As you play with the cutoff frequency, you're supposed to see that, at some point, no signal passes the filter.

Between 199.5Hz and 200.5Hz, we have a transition from passing all the signals (multiply by 1) and rejecting all the signals (multiply by 0).

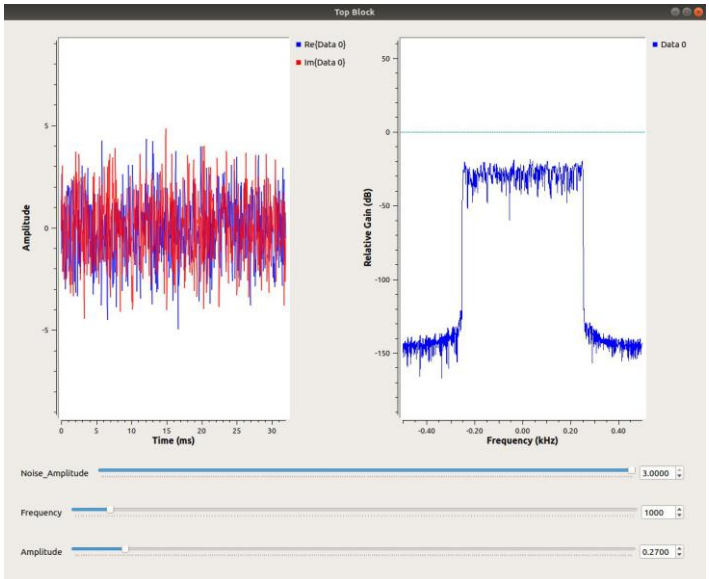
11. Now let's go higher (in frequency). Change back to a Cosine waveform and change the "Frequency" range to stop at 16KHz (16e3).

Change the cutoff of the filter to 8KHz.

Add a Noise Source and a corresponding QT Range for the Noise Amplitude from 0 to 3. Sum both sources with an "Add" component and run the code.



12. Play around with the sliders. Can you see the shape of the filter?



Save your GNU Radio file (code) and add it to your submission!

Theoretical Questions

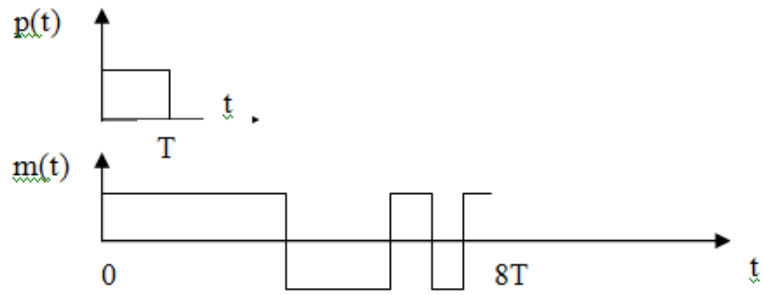
1. How do we know there is an error in our code? How do we figure out what the error is?
2. Say that we have two signals in our flowgraph that we wish to multiply together. How would we find a block that multiplies signals?
3. If you saw a block that had an unused, light gray input port on it, what kind of port would that be?
4. What is the meaning of a yellow port? Given a sample rate of 10 samples per second, how many bits pass this yellow port in 1 second?
5. Say we want to process speech audio data, and we have a microphone that won't let any frequencies in higher than 8 kHz. What is the minimum sampling rate we must use?
6. Now, we want to digitize a radio signal that goes from 99.9 MHz to 100.1 MHz. How large is the minimum applicable sampling rate?

Part 3 – The real deal, modulation!

Digital messages

$m(t)$ is a digital message.

$p(t)$ is chosen to be a so-called square pulse, where we can write $p(t) = \begin{cases} 1\,, & 0 < t < T \\ 0\,, & otherwise \end{cases}$



We send one such $p(t)$ pulse for each data bit, a positive pulse $p(t)$ for logic 1, and a negative pulse for logic 0. In the figure, the message is shown in the interval $0 < t < 8T$ and the data bits are 11100101.

For the period $0 < t < T$, $m(t) = A_0 p(t) = +1$ and the data is logic 1. For the next period $T < t < 2T$, $m(t) = A_1 p(t - T) = +1$, and the data is logic 1. In the example figure above, the message is shown in the interval $0 < t < 8T$ and the information bits are 1110010.

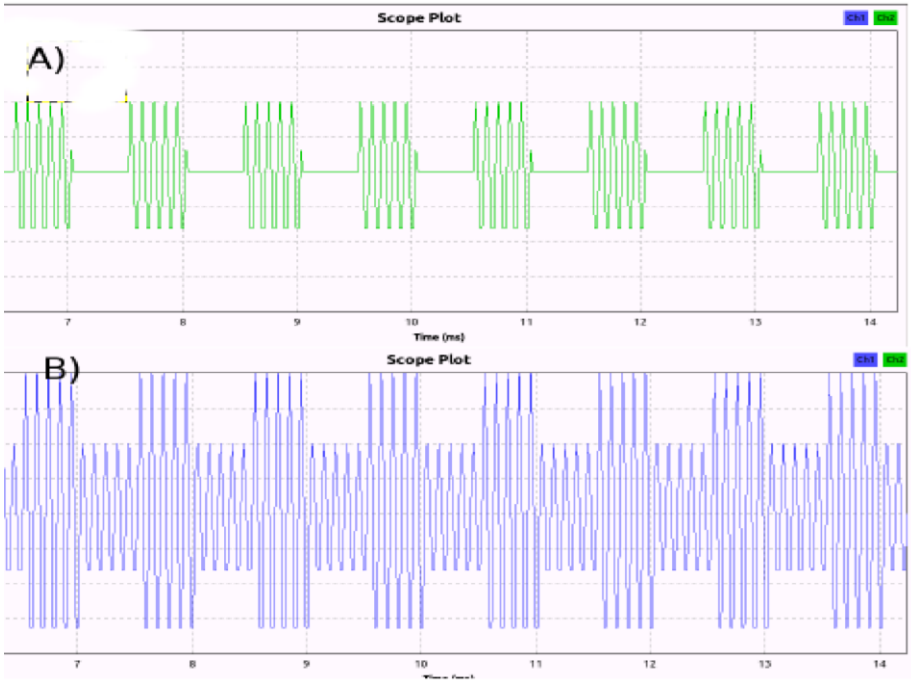
We can write $m(t) = A_0 p(t) + A_1 p(t - T) + A_2 p(t - 2T) = \sum_k A_k p(t - kT)$ where the real constants $A_k = \pm 1$ depending on whether a logic 1 or a logic 0 was sent.

In the figure above, $A_0 = +1$, $A_1 = +1$, $A_2 = +1$, $A_3 = -1$, $A_4 = -1$, $A_5 = +1$, $A_6 = -1$, $A_7 = +1$.

ASK Modulation

Amplitude-shift keying (ASK) is a form of amplitude modulation that represents digital data as variations in the amplitude of a carrier wave. In an ASK system, the binary symbol 1 is represented by transmitting a fixed-amplitude carrier wave and fixed frequency for a bit duration of T seconds. If the signal value is 1 then the carrier signal will be transmitted; otherwise, a signal value of 0 will be transmitted.

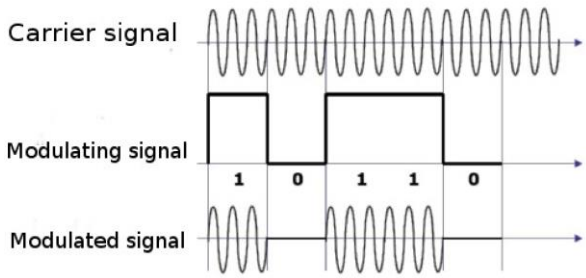
There are two types of ASK modulation: On Off Keying (OOK) and Amplitude Shift Keying



Examples of digitally modulated signals

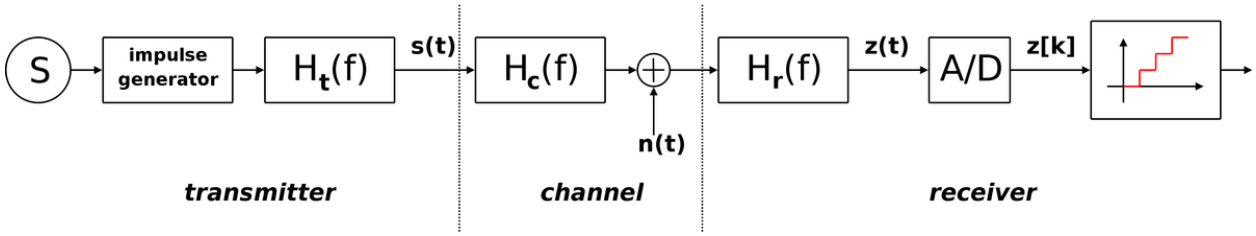
(A) modulation OOK (On-Off Keying), (B) Modulation ASK (Amplitude Shift Keying)

The main difference between the two methods of modulation signal is that, in the case of the ASK modulated signal is emitted during each symbol (even for 'Zero'). For OOK modulation the carrier signal is not transmitted during for 'Zero' symbol.



Example of modulating a binary signal (values 0 and 1) by OOK modulation.

ASK system can be divided into three blocks. The first one represents the transmitter, the second one is a linear model of the effects of the channel, and the third one shows the structure of the receiver.



The following notation is used:

- $\boldsymbol{H_t(f)}$ *is the carrier signal for the transmission*
- $\boldsymbol{H_c(f)}$ *is the impulse response of the channel*
- $\boldsymbol{n(t)}$ *is the noise introduced by the channel*
- $\boldsymbol{H_r(f)}$ *is the filter at the receiver*
- $\boldsymbol{L}$ *is the number of levels that are used for transmission*
- $\boldsymbol{T_s}$ *is the time between the generation of two symbols*

1. ASK/OOK modulation in GNU Radio

1. • Strat in QT GUI mode. All components should be float.

   • Add two signal sources: carrier signal (wave type: **sine**, frequency = **10KHz**), and modulating signal (wave type: **square**, frequency = **100 Hz**). To adjust the frequency, use the component "QT Range" (add two "QT Range" components. One for carrier frequency (10KHz - 100KHz), and second for modulating frequency (10-1000Hz).

   • Multiply these two signals by themselves.

   • Show results of signal modulation (QT GUI Time Sink). Change properties of the QT GUI Time Sink to show the GUI control panel.

   - **This is OOK modulation.**

Place a snapshot of your results in your submission.

2. • For change to ASK modulation, you must change amplitude levels of modulated signal (from 0 and 1, to e.g. 1 and 2). It can be done by add constant value to modulated signal (Component "add const").

   • Show results of signal modulation (QT GUI Time Sink). **This is ASK modulation.**

Place a snapshot of your results in your submission.

Channel Model

The transmission channel model is used to reproduce the behavior of a signal during transmission, in the form of an electromagnetic wave. Additive White Gaussian Noise (AWGN) is used to generate interference and noise. In this case, the noise in our model has a constant spectral power density (watts / Hz) and its amplitude has a Gaussian distribution. This model is represented by the component "Channel Model". This parameter allows you to adjust the intensity of noise is called "Noise voltage". Should be regulated by the GUI (QT Range) in the range of 0 ... 1V (preferably at one mV), the default value should be 0V.

- Place a “Channel Model” block between your encoder and decoder and use a QT Range to control the noise level.

2. OOK / ASK decoding

This time the aim is to recover (to decode) the symbols 0 and 1. The output signal should be exactly the same form as the input signal (although it may shift in time).

• In order to filter out all other signals use the filter *frequency Xlating FIR Filter*. It requires an application filter parameter as a string parameter (variable filter_taps). Create a variable called filter_taps, whose value will be as follows:

firdes.low_pass(1, samp_rate, fc, 25000, firdes.WIN_HAMMING, 6.76)

Place it in the “Taps” parameter of the *frequency Xlating FIR Filter*

• (Compare the parameter names of the typical characteristics of the filter with parameters from 'filter_taps' variable, fc is a carrier frequency signal so change it according to the name you gave it).

• Next, we compute the squared module of the vector formed by the real part and the imaginary signal component via the "Complex to Mag ^ 2" component.

- The next step is decoding the symbol. Each symbol has a fixed duration, amounting to T_k. In our case it will be equal to twice the frequency of the modulating signal.

  To decode the symbols, use the component "Clock Recovery MM". The "Omega", which is a factor number of samples per symbol, is set to 1 (bit rate equal to the velocity of symbols).

• Show results of signal modulation (QT GUI Time Sink). Compare with original modulated.

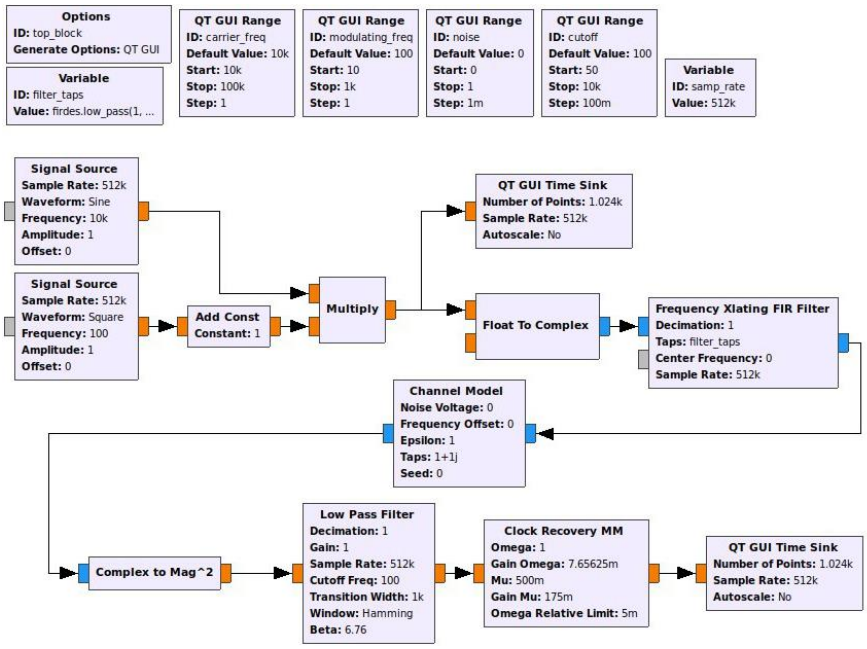
• Check how it behaves when the input signal is increases and is greater than the noise level.

• In order to improve the decoding can use a low pass filter to filter the noise. Place it between the "Complex Mag ^ 2" and "Clock Recovery MM". Adjust filter parameters to get the best results (to adjust the filter cutoff frequency, use the component “QT Range” from 50Hz to 10KHz, Transition With set to 100 Hz).

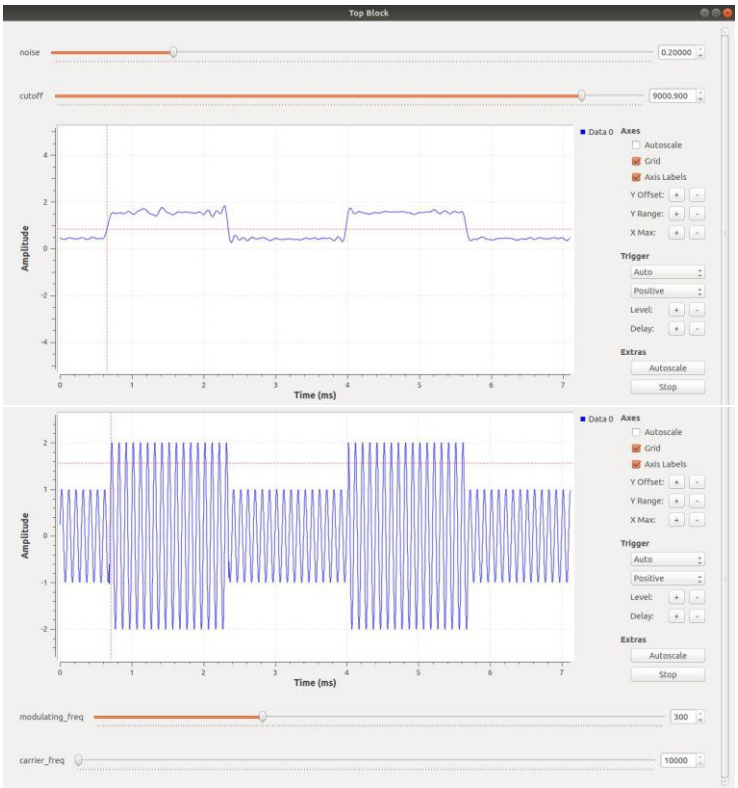
Place a snapshot of your results in your submission.

Save your code and add it to your submission!

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



This are the signals that you should see in your oscilloscopes:



Lower signal is the modulated ASK signal.

Upper signal is the decoded data with noise and low pass filtered.

In the practical session to come, we will construct a more complex wireless system and test it durability and preferences by affecting the wireless channel medium. Make sure you are familiar with the basic digital communication and wireless topics discussed and that you've installed the requested software – you are going to use it! If your code looks different, fix it or align it to the required standard… you are going to use it as well…

Theoretical Questions

1. How did the low pass filter you added affect the recovered data? What was the effect of the cutoff frequency? Why?
2. What would happen to the data if we didn't know the carrier frequency at the receiver side? If we could not reach the exact required frequency, can you give a solution for this problem?
3. ASK modulation is very susceptible to noise interference. This is due to the fact that noise affects the amplitude. What alternative can we use to transmit data that will have lesser noise susceptibility?
4. How could we upgrade the OOK/ASK in order to increase the amount of data transferred?
5. Amplitude changes can be considered as a *dimension of data*. Is it possible to upgrade ASK by adding another dimension of data? If so, what would be the advantages and disadvantages of this dimension addition?

Good luck!

References

http://www.eletel.p.lodz.pl/tele/en/index.php?option=com_jotloader&section=files&task=download&cid=250_5f7e7b91d8a50da010b55ee4ea27f6a2&Itemid=239