Digital Wireless Communication Theoretical Laboratory Session

WIRELESS COMMUNICATIONS 371-1-1903
SPRING 2020

Part 1 – General Theoretical Information

SISO Modulation

Data transmission (also data communication or digital communications) is the transfer of data (a digital bitstream or a digitized analog signal) over a point-to-point or point-to-multipoint communication channel.

Examples of such channels are copper wires, optical fibers, wireless communication channels, storage media and computer buses. The data is represented as an electromagnetic signal, such as an electrical voltage, radio wave, microwave, or infrared signal.

A **single-input and single-output** (**SISO**) system is a simple single variable control system with one input and one output. In radio it is the use of only one antenna both in the transmitter and receiver.

In digital modulation, an analog carrier signal is modulated by a discrete signal. Digital modulation methods can be considered as digital-to-analog conversion and the corresponding demodulation or detection as analog-to-digital conversion. The changes in the carrier signal are chosen from a finite number of M alternative symbols (the *modulation alphabet*).

A **modulator** is a device that performs modulation. A **demodulator** (sometimes detector or demod) is a device that performs demodulation, the inverse of modulation. A modem (from modulator–demodulator) can perform both operations.

The aim of analog modulation is to transfer an analog baseband (or lowpass) signal, for example an audio signal or TV signal, over an analog bandpass channel at a different frequency, for example over a limited radio frequency band or a cable TV network channel. The aim of digital modulation is to transfer a digital bit stream over an analog communication channel, for example over the public switched telephone network (where a bandpass filter limits the frequency range to 300–3400 Hz) or over a limited radio frequency band.

In a communication system, the receiver side bit error rate may be affected by transmission channel noise, interference, distortion, bit synchronization problems, attenuation, wireless multipath fading, etc.

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 is several ways and see what happens.

Resources
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

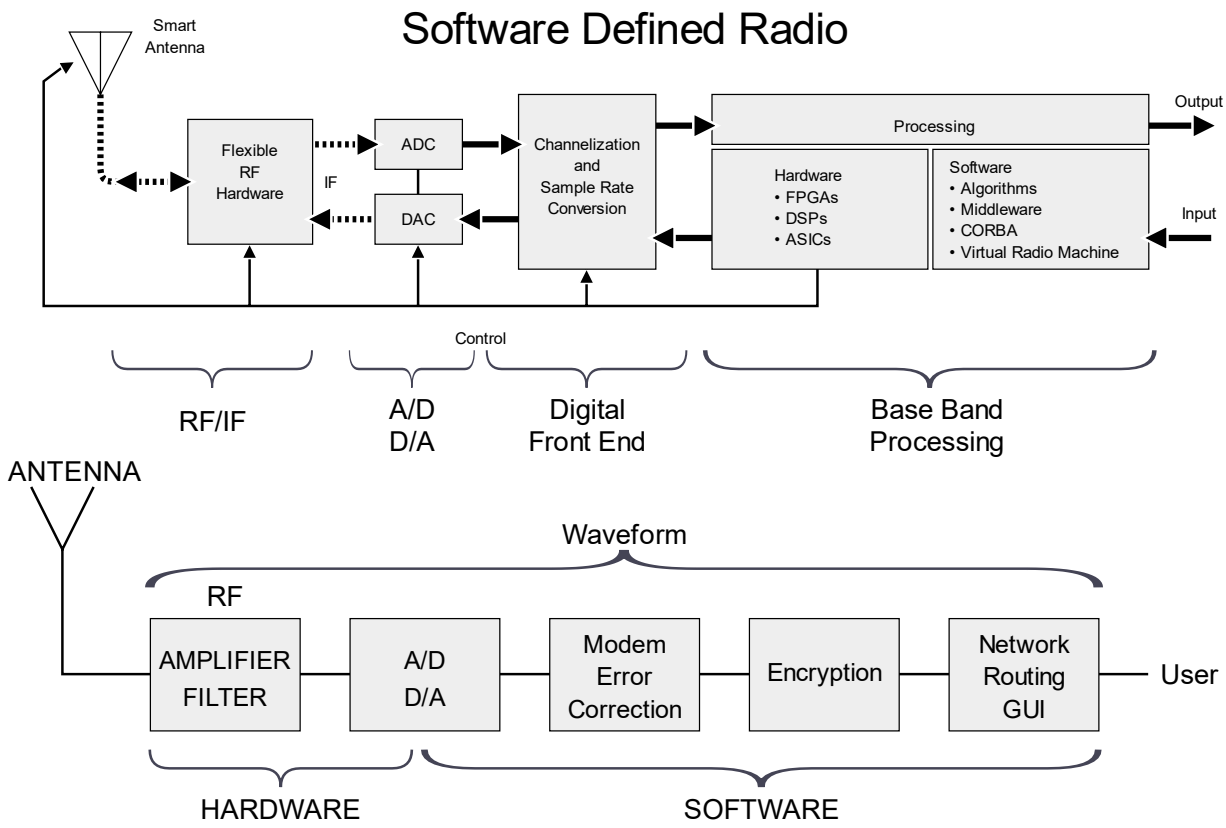
Theoretical Questions

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. Choose one of the modulation schemes from question 2 and explain how you would calculate an error probability for that kind of modulation. What is the minimal SNR and resulting BER required at the receiver end for a good output?
4. What is the most dominating factor effecting the BER of a channel? (noise, interference, distortion synchronization, attenuation, fading or else)
5. Given an endless amount of money that you can only spend on your own system (receiver and transmitter only, not the channel medium) what would you do to improve your transfer of data?

SDR - Software-defined radio

Software-defined radio (**SDR**) is a radio communication system where components that have been traditionally implemented in hardware (e.g. mixers, filters, amplifiers, modulators/demodulators, detectors, etc.) are instead implemented by means of software on a personal computer or embedded system.

A basic SDR system may consist of a personal computer equipped with a sound card, or other analog-to-digital converter, preceded by some form of RF front end. Significant amounts of signal processing are handed over to the general-purpose processor, rather than being done in special-purpose hardware (electronic circuits). Such a design produces a radio which can receive and transmit widely different radio protocols (sometimes referred to as waveforms) based solely on the software used.



The ideal receiver scheme would be to attach an analog-to-digital converter to an antenna. A digital signal processor would read the converter, and then its software would transform the stream of data from the converter to any other form the application requires.

An ideal transmitter would be similar. A digital signal processor would generate a stream of numbers. These would be sent to a digital-to-analog converter connected to a radio antenna.

The ideal scheme is not completely realizable due to the current limits of the technology. The main problem in both directions is the difficulty of conversion between the digital and the analog domains at a high enough rate and a high enough accuracy at the same time, and without relying upon physical processes like interference and electromagnetic resonance for assistance.

Receiver architecture

Most receivers use a variable-frequency oscillator, mixer, and filter to tune the desired signal to a common intermediate frequency or baseband, where it is then sampled by the analog-to-digital converter. However, in some applications it is not necessary to tune the signal to an intermediate frequency and the radio frequency signal is directly sampled by the analog-to-digital converter (after amplification).

Real analog-to-digital converters lack the dynamic range to pick up sub-microvolt, nanowatt-power radio signals. Therefore, a low-noise amplifier must precede the conversion step and this device introduces its own problems. For example, if spurious signals are present (which is typical), these compete with the desired signals within the amplifier's dynamic range. They may introduce distortion

in the desired signals or may block them completely. The standard solution is to put band-pass filters between the antenna and the amplifier, but these reduce the radio's flexibility. Real software radios often have two or three analog channel filters with different bandwidths that are switched in and out.

Resources

https://en.wikipedia.org/wiki/Software-defined_radio
https://www.allaboutcircuits.com/technical-articles/introduction-to-software-defined-radio/
https://ieeexplore.ieee.org/document/7727079

Theoretical Questions

6. Point out several advantages and disadvantages in using an SDR versus using a traditional hardware radio.
7. What are the performance limitations of an SDR?
8. What software could one use If he to implement a radio application using an SDR?

Part 2 – GNU Radio

GNU Radio is a free software development toolkit that provides signal processing blocks to implement software-defined radios and signal-processing systems. It can be used with external RF hardware to create software-defined radios, or without hardware in a simulation-like environment. It is widely used in hobbyist, academic, and commercial environments to support both wireless communications research and real-world radio systems.

The GNU Radio software provides the framework and tools to build and run software radio or just general signal-processing applications. The GNU Radio applications themselves are generally known as "flowgraphs", which are a series of signal processing blocks connected together, thus describing a data flow.

As with all software-defined radio systems, reconfigurability is a key feature. Instead of using different radios designed for specific but disparate purposes, a single, general-purpose, radio can be used as the radio front-end, and the signal-processing software (here, GNU Radio), handles the processing specific to the radio application.

These flowgraphs can be written in either C++ or the Python programming language. The GNU Radio infrastructure is written entirely in C++, and many of the user tools are written in Python.

Resources

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 correct results!

Follow the instructions on how to install GNU Radio on your system, choose one of the following options:

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

   Command prompt:
   `pybombs run gnuradio-companion`

b. You can install it on your system using this command instruction 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.

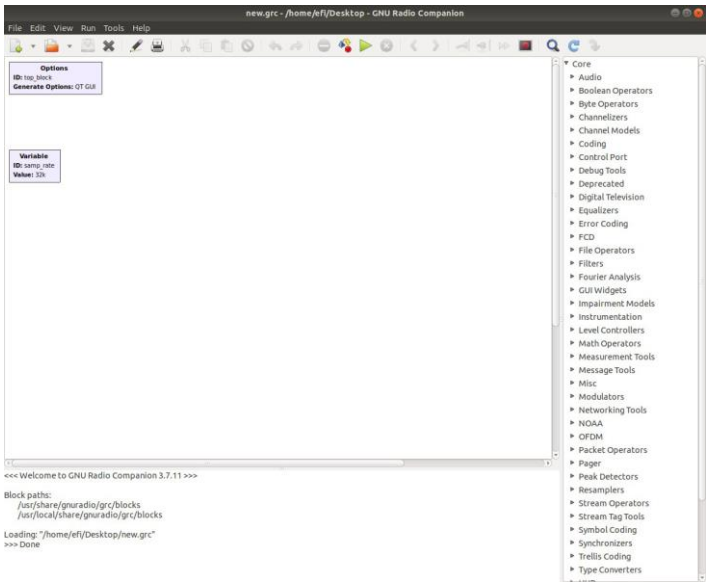
Feel free to read about and explorer the GNU radio and all of its components and abilities.

Tutorial

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

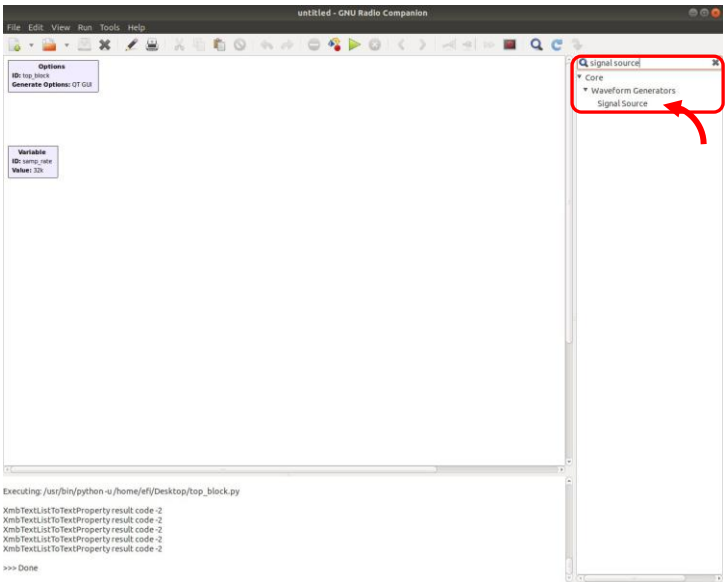
Instructions:

1. Open GNU Radio.



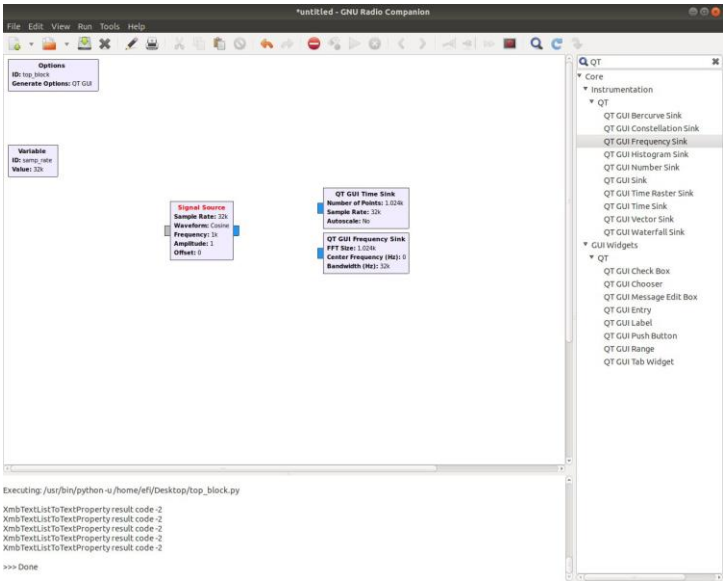
2. Open a new QT GUI file.

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.

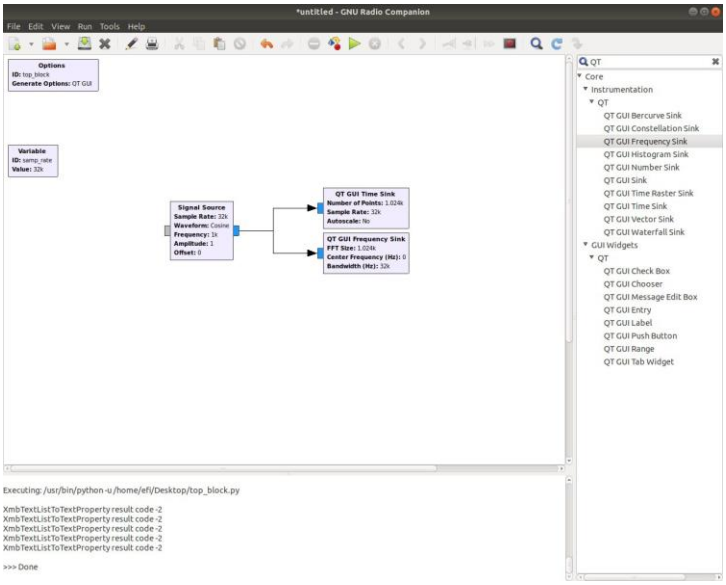


This block simulates a signal source, a sinusoidal function such as $x(t) = Asin(2\pi ft + \varphi)$, Cosine, Square, Triangle and more…

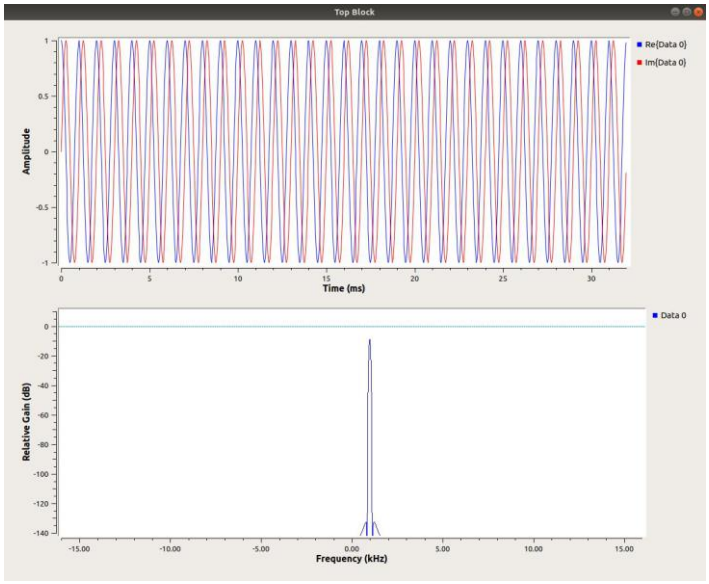
4. Now search and place “QT GUI Time Sink” and “QT GUI Frequency Sink”. These are graphical user interface that allow us to see the signal in the time domain and its spectrum in the frequency domain.



5. Now, lets 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 sell all types color mapping in “help”. Notice, you can only connect ports of the same types.

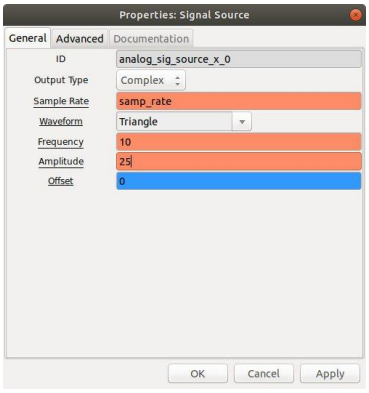


6. Now, go ahead and run the code (remember to save it first).

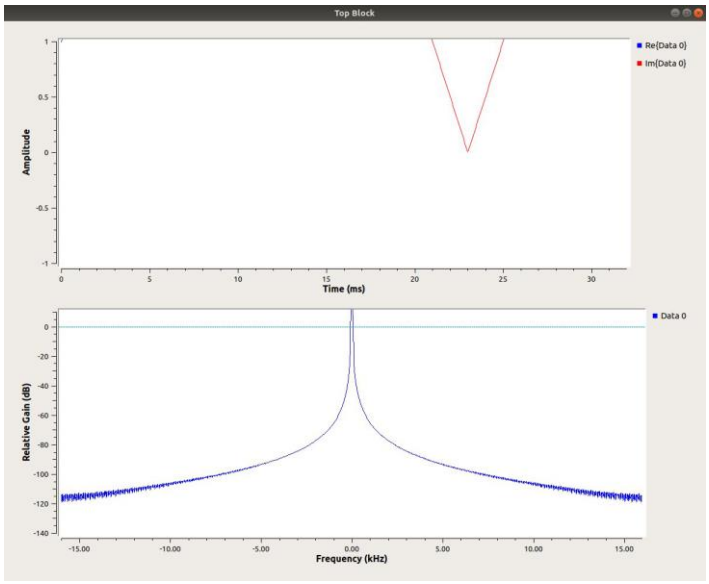


A new window appears which shows the signal amplitude vs time and frequency. Notice that the signal has default values since we did not change any of the component's properties.

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.

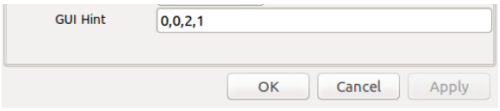


8. Go ahead and run it again.



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

All of 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,

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

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)

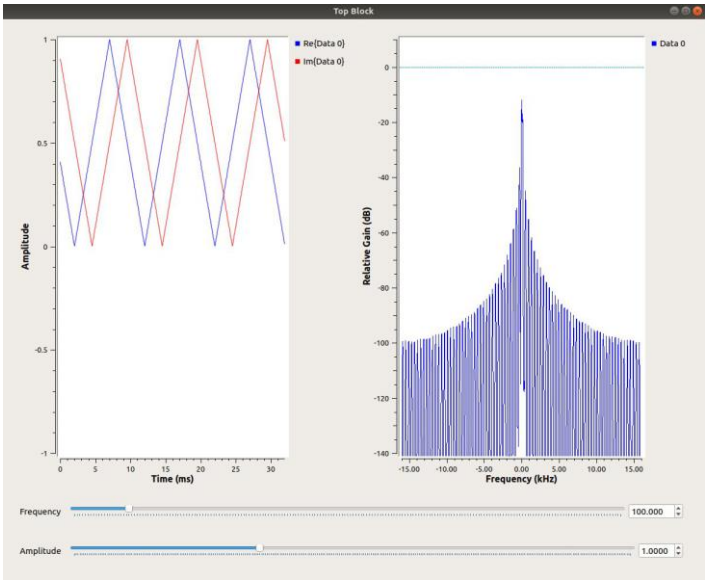
9. Now that you have everything under control, let's take it up a notch.

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

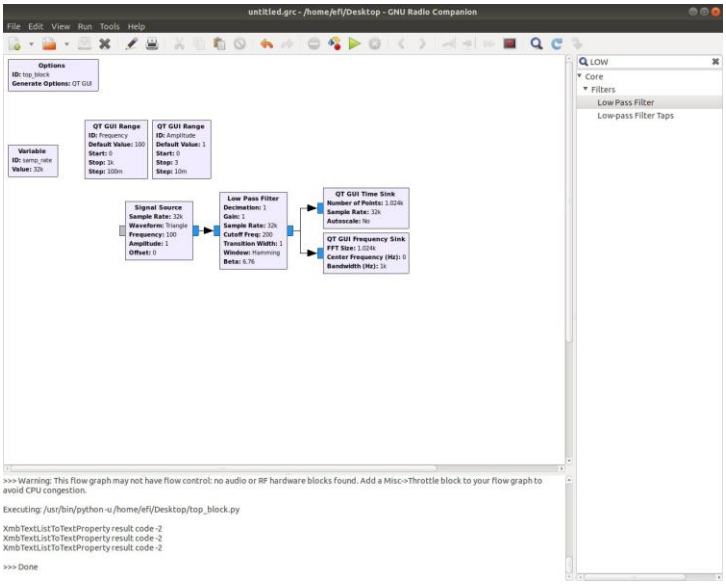
Now go to the signal source and enter the “Frequency” (name that you gave to the QT Range) in Frequency.

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 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?

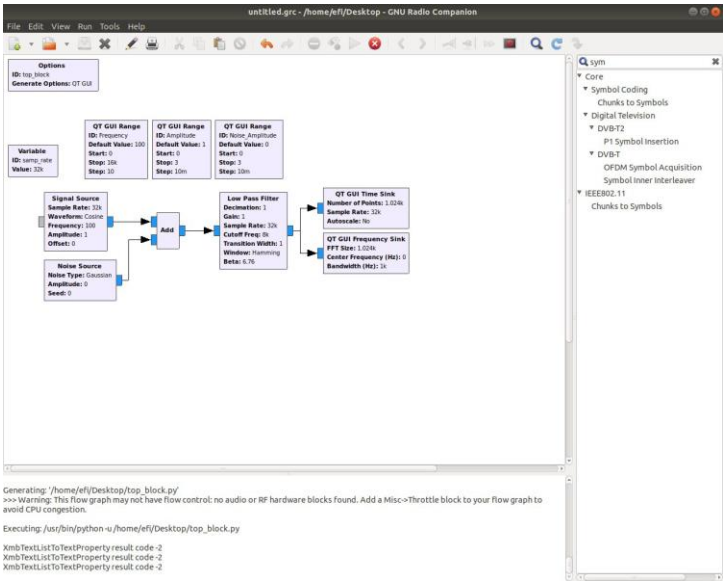
When you low pass limit the triangle signal's bandwidth you can see that it loses its higher frequencies, thus all fast changing components of the signal (like a quick drop or rise) are now missing and we are left with a smaller number of sinusoidal components (low). When we filter out all but the main sinusoidal component, we get a sinus waveform. When we pass the cutoff frequency, we are left with nothing, thus, no signal passes the filter.

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

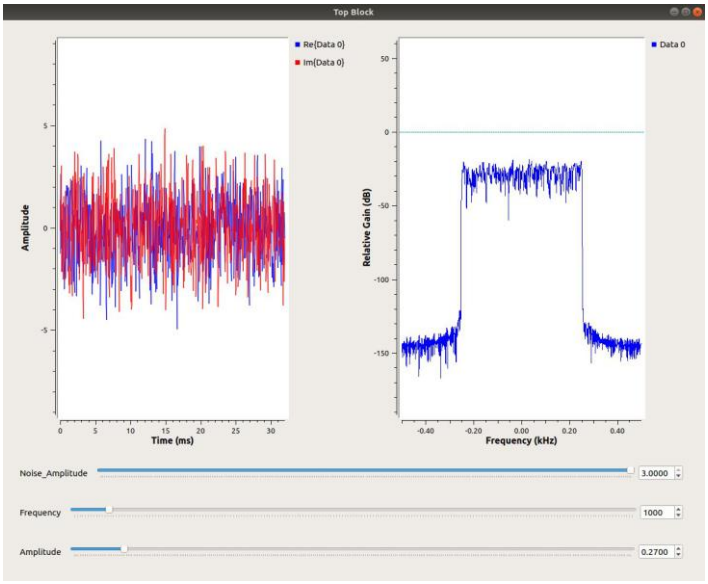
11. Now let's go higher (in frequency). Change back to a Cosine waveform and change the "Frequency" range to stop at 16KHz (16e3).

    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

13. How do we know there is an error in our code? How do we figure out what the error is?

14. Say that we have two signals in our flowgraph that we wish to multiply together. How would we find a block that multiplies signals?

15. If you saw a block had an unused, light gray input port on it, what kind of port would that be?

16. 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?

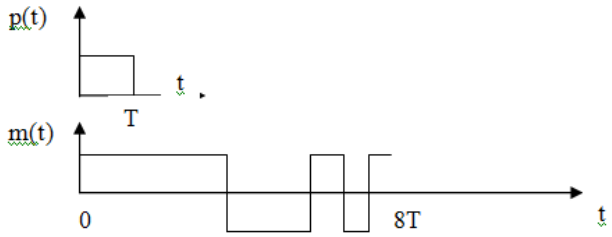
17. Now we want to digitize a radio signal that goes from 99.9 MHz to 100.1 MHz. How large is the minimum applicable sampling rate?

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 11100101

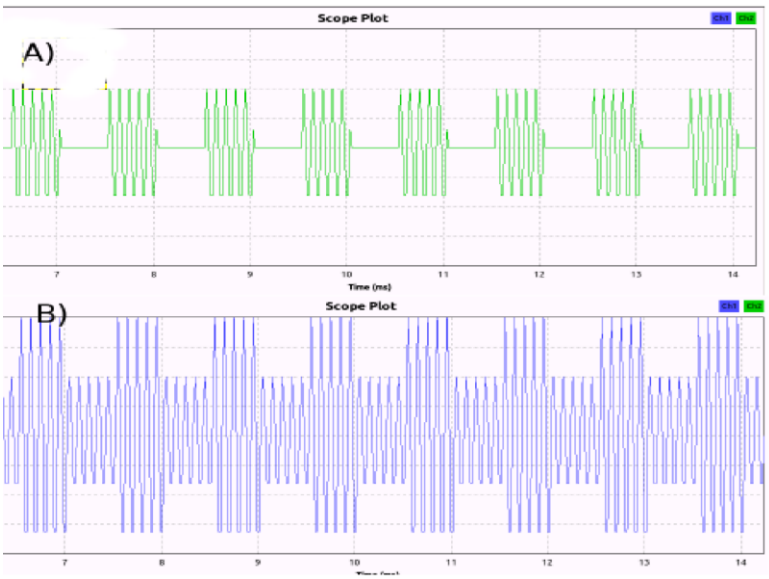
We can write $m(t) = A_0 p(t) + A_1 p(t - T) + A_2 p(t - 2T) = \sum_k A_k p(t - kT)$ where the real constants $A_k = \pm 1$ depending on whether a logic 1 or a logic 0 was sent.

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.

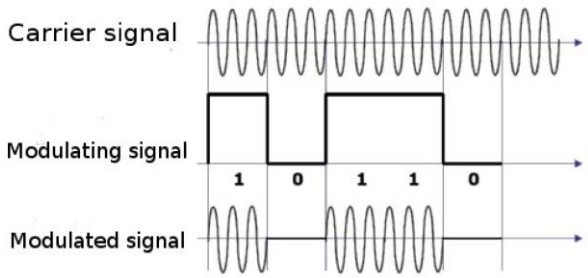
There are two types of ASK modulation: On Off Keying (OOK) and Amplitude Shift Keying



Examples of digitally modulated signals

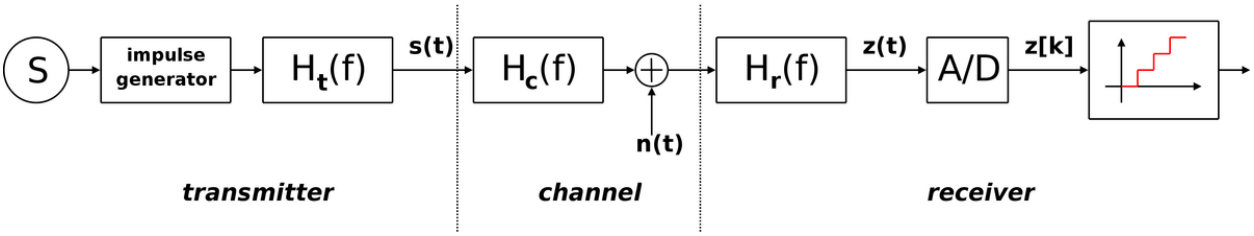
(A) modulation OOK (On-Off Keying), (B) Modulation ASK (Amplitude Shift Keying)

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, the third one shows the structure of the receiver.



The following notation is used:

- $h_t(f)$ *is the carrier signal for the transmission*
- $h_c(f)$ *is the impulse response of the channel*
- ***n(t)*** *is the noise introduced by the channel*
- $h_r(f)$ *is the filter at the receiver*
- ***L*** *is the number of levels that are used for transmission*
- T_s *is the time between the generation of two symbols*

1. ASK/OOK modulation in GNURadio

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

   • Add a 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 best reproduce the behavior of the signal during transmission, in the form of electromagnetic wave. To generate interference and noise AWGN (Additive White Gaussian Noise) is used. It is a model of the channel in which the noise and spectral power density (watts / Hz) is constant and there is a Gaussian distribution of amplitude. 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 be shift at time).

• In order to filter out all other signals use the filter *frequecny 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 *frequecny 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 "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 (in one period of symbol are contain two symbols 0 and 1).

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

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

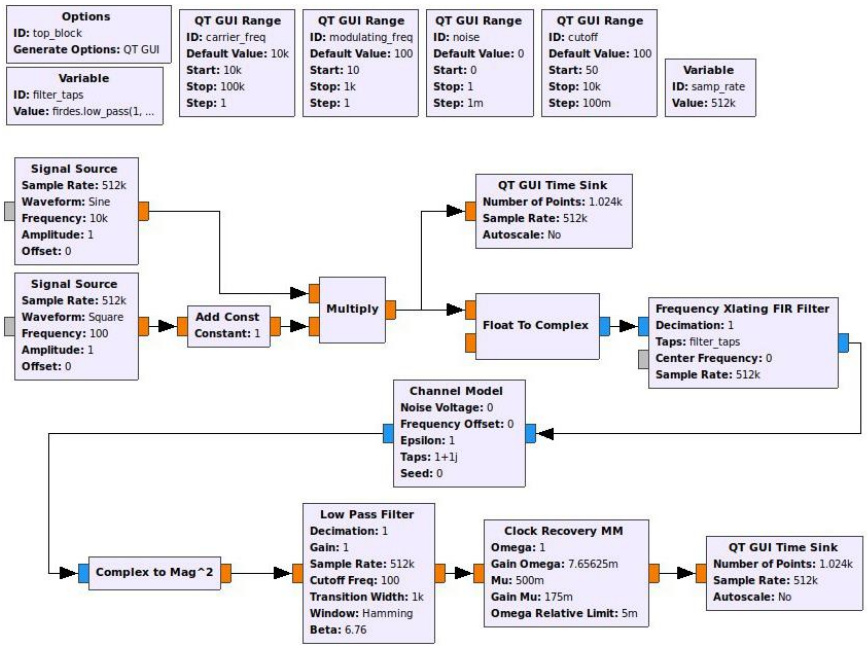
• Check how behaves when the input signal is increasing the noise level.

• 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).

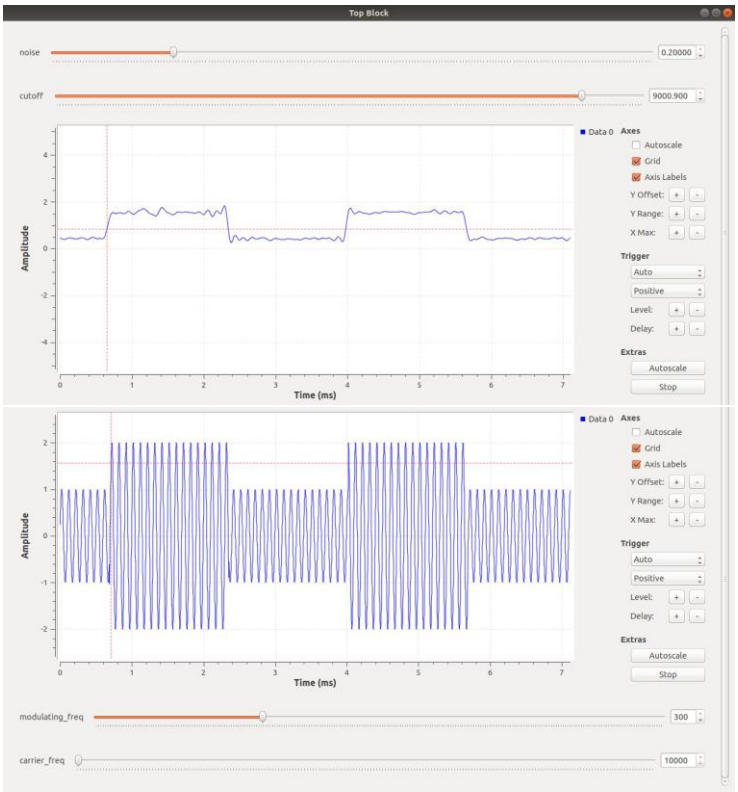
Place a snapshot of your results in your submission.

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 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

18. How did the low pass filter you added effect the recovered data? What was the effect of the cutoff frequency?

19. What would happen to the data if we didn't know the carrier frequency at the receiver side? Can you give a solution for this problem?

20. 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?

21. How could we upgrade the OOK/ASK in order to increase the amount of data transferred?

22. Can you upgrade your 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