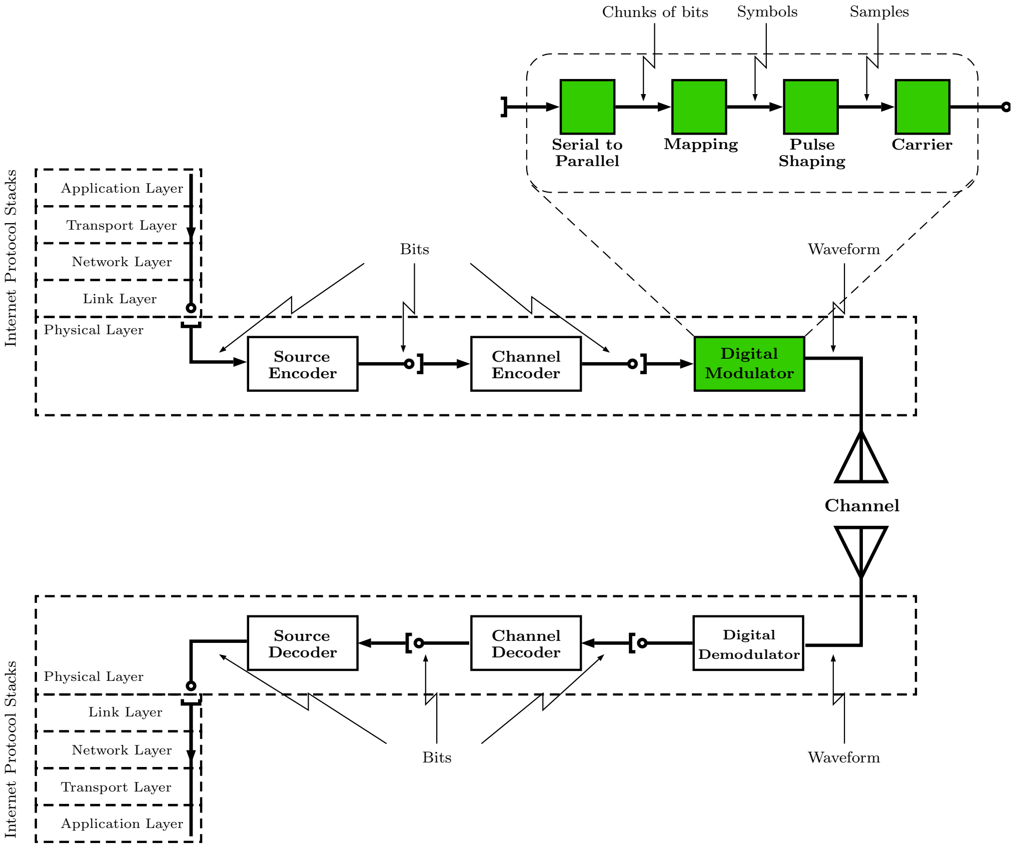
ELEN90051 Advanced Communication Systems
Digital Communication Systems in GNU Radio
Workshop week 5 (=*week of 26 March*): QPSK Modulation

Department of Electrical and Electronic Engineering
The University of Melbourne

Created by Xiangyue Meng and Margreta Kuijper

updated on March 20, 2018

This series of workshops will guide you to implement practical software-defined digital communication systems. The tools that you will learn are powerful enough to build radio systems for studies, research, commercial use, and hobbies. Figure 1 shows the road map towards the end of this semester. We highly encourage you to apply the theories that you learned from the lectures to explore different functionalities of the system.



The green blocks are going to be implemented in this workshop.

Figure 1: The road map of implementing a digital communication system.

I Objectives and Logistics

In this workshop, a practical QPSK modulator will be implemented in GNU Radio. Specifically, you will

- understand the principles of QPSK modulation;
- implement a QPSK modulator in GNU Radio;
- implement several other simple modulators in GNU Radio.

You are expected to **be prepared before attending the workshop session**. For this, read through the workshop manual. An **individual pre-workshop report**, answering Question 1 to Question 5, worth 10 marks, is to be submitted before the start of the workshop session.
There is an in-workshop checkoff point upon successfully finishing Section III, worth 10 marks.
You are also asked to write a group project report, following the instructions in Section IV. This report is worth 30 marks and should be submitted before the start of your next workshop in week 6 (=week of 9 April). Please read the document "Rules on workshops & report submission" for more information, see `LMS/Workshops/Rules on Workshops and Report Submission/RulesWorkshopsReportsELEN90051.pdf`.

II Background

1 Quadrature Phase-Shift Keying

1.1 Signal Space Representation of Bandpass Signals

Recall from the lecture that, in QPSK, information is embedded in the phase of the sinusoidal carrier. Specifically, the phase of the sinusoidal carrier takes on one of four equally spaced values, such as $\pi/4, 3\pi/4, 5\pi/4$, and $7\pi/4$. For this set of phases, the transmitted signals $s(t)$, for $t \in (0, T]$, are given by

$$\begin{aligned} s(t) &= g(t)\cos\left(\frac{(2m+1)\pi}{4} + 2\pi f_c t\right), \qquad m \in \{0,1,2,3\} \\ &= g(t)\cos\left(\frac{(2m+1)\pi}{4}\right)\cos 2\pi f_c t - g(t)\sin\left(\frac{(2m+1)\pi}{4}\right)\sin 2\pi f_c t \end{aligned} \tag{1}$$

where T is the signaling period, f_c is the carrier frequency (usually $f_c \gg \frac{1}{T}$), and $g(t)$ is the lowpass signal shaping pulse. The signal space's orthonormal basis is given by,

$$\begin{aligned} \phi_1(t) &= \sqrt{\frac{2}{\mathcal{E}_g}}g(t)\cos 2\pi f_c t, \\ \phi_1(t) &= -\sqrt{\frac{2}{\mathcal{E}_g}}g(t)\sin 2\pi f_c t, \end{aligned} \tag{2}$$

where $\mathcal{E}_g$ is the energy in $g(t)$ given by $\mathcal{E}_g = \int_{-\infty}^{\infty} g^2(t)\mathrm{d}t$. Now $s(t)$ can be written as

$$s(t) = \sqrt{\frac{\mathcal{E}_g}{2}}\cos\left(\frac{(2m+1)\pi}{4}\right)\phi_1(t) + \sqrt{\frac{\mathcal{E}_g}{2}}\sin\left(\frac{(2m+1)\pi}{4}\right)\phi_2(t). \tag{3}$$

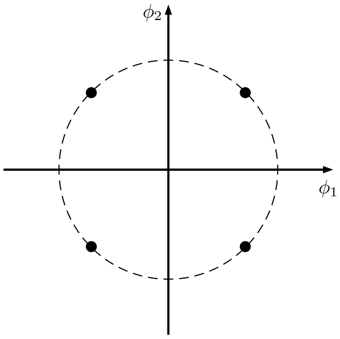


Figure 2: A sketch of a QPSK constellation diagram in the signal space.

Clearly the signal space dimensionality is $N = 2$ and the resulting vector representations of $s(t)$ is

$$\mathbf{s} = \left(\sqrt{\frac{\mathcal{E}_g}{2}} \cos\left(\frac{(2m+1)\pi}{4}\right), \sqrt{\frac{\mathcal{E}_g}{2}} \sin\left(\frac{(2m+1)\pi}{4}\right) \right), \qquad m \in \{0,1,2,3\}.$$

Figure 2 shows a sketch of a QPSK constellation diagram in the signal space.

1.2 Signal Space Representation of Lowpass Signals

Recall from the prerequisite subject *ELEN90057 Communication Systems* that a bandpass signal $s(t)$ can be expressed in terms of its lowpass equivalent $s^{\text{lowpass}}(t)$ as

$$s(t) = \Re\{s^{\text{lowpass}}(t) e^{j2\pi f_c t}\}. \tag{4}$$

The real and imaginary parts of $s^{\text{lowpass}}(t)$ are called the *in-phase component* and the *quadrature component*, respectively, and are denoted by $I(t)$ and $Q(t)$. Both $I(t)$ and $Q(t)$ are real-valued lowpass signals, and we have,

$$s^{\text{lowpass}}(t) = I(t) + jQ(t). \tag{5}$$

From (1), for $t \in (0, T]$, the signal $s_m(t)$ can be rewritten as

$$s_m(t) = \Re\left\{ g(t) e^{j\frac{(2m+1)\pi}{4}} e^{j2\pi f_c t} \right\}, \qquad m \in \{0,1,2,3\}. \tag{6}$$

Therefore, its lowpass equivalent is

$$\begin{aligned} s_m^{\text{lowpass}}(t) &= g(t) e^{j\frac{(2m+1)\pi}{4}} \\ &= g(t)\cos\left(\frac{(2m+1)\pi}{4}\right) + jg(t)\sin\left(\frac{(2m+1)\pi}{4}\right), \qquad m \in \{0,1,2,3\}, \end{aligned} \tag{7}$$

and the in-phase and quadrature components are

$$\begin{aligned} I_m(t) &\triangleq g(t)\cos\left(\frac{(2m+1)\pi}{4}\right) \\ Q_m(t) &\triangleq g(t)\sin\left(\frac{(2m+1)\pi}{4}\right). \end{aligned} \tag{8}$$

With orthonormal basis

$$\begin{aligned}\phi_1^{\text{lowpass}}(t) &= \frac{g(t)}{\sqrt{\mathcal{E}_g}},\\ \phi_2^{\text{lowpass}}(t) &= \frac{jg(t)}{\sqrt{\mathcal{E}_g}},\end{aligned} \tag{9}$$

we can write

$$s^{\text{lowpass}}(t)_m = \sqrt{\mathcal{E}_g}\cos\left(\frac{(2m+1)\pi}{4}\right)\phi_1^{\text{lowpass}}(t) + \sqrt{\mathcal{E}_g}\sin\left(\frac{(2m+1)\pi}{4}\right)\phi_2^{\text{lowpass}}(t). \tag{10}$$

Again the signal space dimensionality is $N = 2$ and the resulting vector representation of $s^{\text{lowpass}}(t)$ is

$$\mathbf{s}_m^{\text{lowpass}} = \left(\sqrt{\mathcal{E}_g}\cos\left(\frac{(2m+1)\pi}{4}\right), \sqrt{\mathcal{E}_g}\sin\left(\frac{(2m+1)\pi}{4}\right)\right), \qquad m \in \{0,1,2,3\}. \tag{11}$$

If we define $a_1, a_2 \in \{-1, 1\}$, $\mathbf{s}_m^{\text{lowpass}}$ can be written as

$$\mathbf{s}_m^{\text{lowpass}} = \left(\sqrt{\frac{\mathcal{E}_g}{2}}a_1, \sqrt{\frac{\mathcal{E}_g}{2}}a_2\right). \tag{12}$$

Note that there is only a factor of $\sqrt{2}$ difference from the signal space representation of the bandpass signals, due to the exclusion of the carrier. The orthogonality of the signal space of bandpass signals comes from the waveforms $\cos(x)$ and $\sin(x)$, whereas the orthogonality of the signals space of lowpass signals comes from the real and imaginary unit bases.

In practice, a mapping between a dibit b_1b_2 and the coordinates on the complex plane (a_1, a_2) is needed, which is referred as the **look-up table (LUT)** in the textbook.

Dibit: A group of two bits.

At this point, the following question naturally arises: how to map the four dibits to the four constellation points?

2 Gray Code

A Gray code is a binary numeral system where two successive values differ in only one bit. Today, Gray codes are widely used in digital communication systems.

Gray code, after Frank Gray, is also known as reflected binary code (RBC).

ToDo: **Question 1 (*2 marks*).** Give a sequence of Gray codewords for 16-ary modulation starting with $\{\texttt{0000}, \texttt{0001}, \ldots\}$.

The next step is to map Gray codewords to the points of the constellation for a specific modulation scheme.

ToDo: **Question 2 (*4 marks*).** Consider a QPSK modulation scheme that uses Gray encoding, where the phase of the sinusoidal carrier takes on one of four equally spaced values, such as $\pi/4, 3\pi/4, 5\pi/4$, and $7\pi/4$.

(a) Give a sequence of Gray codewords for QPSK modulator starting with $\{\texttt{00}, \ldots\}$.

(b) Complete the corresponding LUT below.

Index	**Gray Codeword**	**Phase**	**Coordinate**	**Complex Number**
0	`00`	$5\pi/4$	$(-1,-1)$	$-1-1j$

(Here the index is the decimal representation of the Gray codeword.)

(c) Sketch the constellation diagram in the complex plane and label the signal points with the corresponding dibits.

In practice, we use the column **Complex Number**, ordered by the column **Index**, to construct the LUT, a `list` of complex numbers in `Python`.

`list`: A versatile datatype in `Python` that stores a sequence of comma-separated items between square brackets.

III In-Workshop Tasks (*10 marks*)

In this workshop, you are supposed to complete the following three tasks:

(1) Getting to know the GNU Radio Companion (GRC)

(2) Implementing the QPSK Modulator in GRC (In-Workshop)

(3) Implementing a 16-QAM Modulator in GRC (Report)

After this workshop, you will be able to implement simple modulators in GNU Radio.

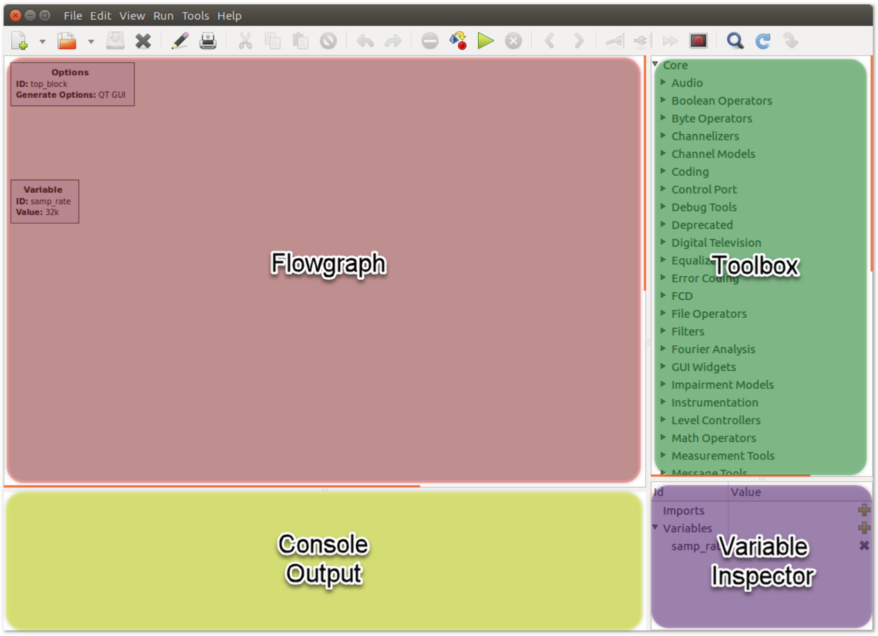
Task 1: Getting to know the GNU Radio Companion

GNU Radio is a free and open-source software development toolkit that provides signal processing blocks to implement software radios. It is implemented in `C++` and officially supported in `Linux` systems. GNU Radio Companion (GRC) is the graphic user interface for using the functions implemented in GNU Radio. It works in a similar way as Simulink, where functional blocks can be dragged and dropped into a flowgraph and connected by flow arrows. First, let us familiarize ourselves with the interface of the GRC. We assume that you have successfully installed GNU Radio on your computer. For technical supports, please contact the demonstrator.

Despite that great efforts have been made to provide installation instructions and solve build bugs for GNU Radio on Windows systems, as reported in the developer community, its compatibility with other operating systems such as Windows and MAC OS is still limited.

Step 1: Launch GRC by clicking the GRC icon on the launch bar of `Ubuntu`. Wait a few seconds until the following main window appears.

The GRC icon on the launch bar of `Ubuntu`.



Options
ID: top_block
Generate Options: QT GUI

Variable
ID: samp_rate
Value: 32k

Figure 3: The main window of the GRC in Ubuntu.

The main window of the GRC consists of five areas. The menu and toolbar are on the top of the window, where the main functions of the GRC are located, such as `New`, `Open`, `Save`, `Run`, `Stop`, and `Search`. The workspace is further divided into four parts:

1 **Flowgraph**: A work area where blocks are created and connected in the flowgraph. There will be tabs on the top of the area if multiple flowgraphs are opened.

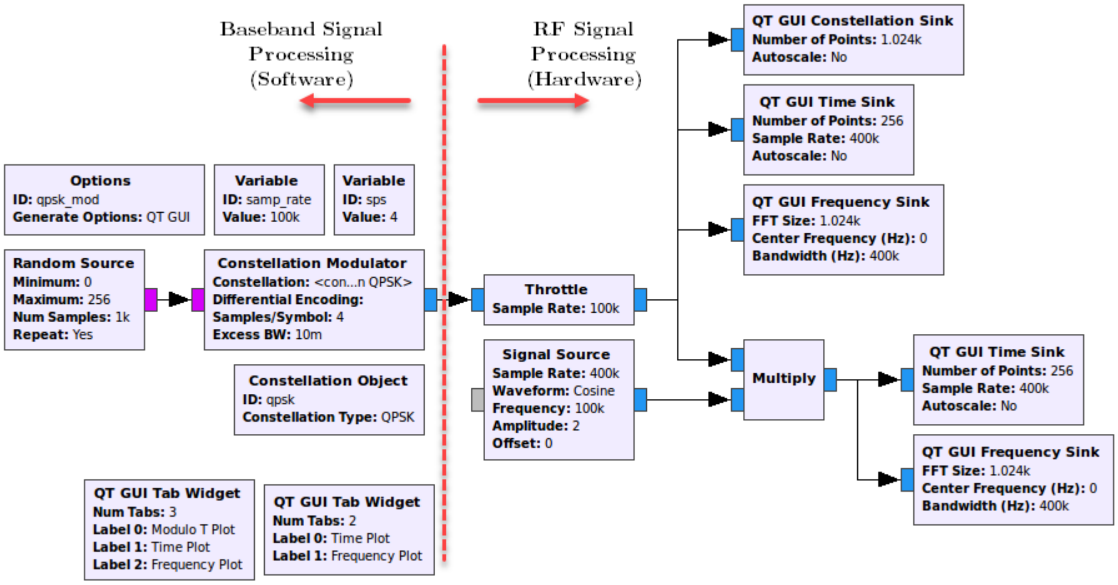
2 **Toolbox**: A tree list that shows all available blocks installed in GNU Radio.

3 **Console Output**: An output console that shows debug and error messages.

4 **Variable Inspector**: A tree list that shows all variables and their values as used in the flowgraph.

Step 2: In the virtual machine, download `qpsk_mod_demo.grc` from LMS and open it in the GRC. You will see the high-level flowgraph of a modulator, as shown in figure 4.

`Open` button is the second icon on the toolbar of the GRC.


Baseband Signal Processing (Software)
RF Signal Processing (Hardware)

Options
ID: qpsk_mod
Generate Options: QT GUI

Variable
ID: samp_rate
Value: 100k

Variable
ID: sps
Value: 4

Random Source
Minimum: 0
Maximum: 256
Num Samples: 1k
Repeat: Yes

Constellation Modulator
Constellation: <con...n QPSK>
Differential Encoding:
Samples/Symbol: 4
Excess BW: 10m

Constellation Object
ID: qpsk
Constellation Type: QPSK

QT GUI Tab Widget
Num Tabs: 3
Label 0: Modulo T Plot
Label 1: Time Plot
Label 2: Frequency Plot

QT GUI Tab Widget
Num Tabs: 2
Label 0: Time Plot
Label 1: Frequency Plot

Throttle
Sample Rate: 100k

Signal Source
Sample Rate: 400k
Waveform: Cosine
Frequency: 100k
Amplitude: 2
Offset: 0

QT GUI Constellation Sink
Number of Points: 1.024k
Autoscale: No

QT GUI Time Sink
Number of Points: 256
Sample Rate: 400k
Autoscale: No

QT GUI Frequency Sink
FFT Size: 1.024k
Center Frequency (Hz): 0
Bandwidth (Hz): 400k

Multiply

QT GUI Time Sink
Number of Points: 256
Sample Rate: 400k
Autoscale: No

QT GUI Frequency Sink
FFT Size: 1.024k
Center Frequency (Hz): 0
Bandwidth (Hz): 400k

Figure 4: A high-level flowgraph of the QPSK modulator in GRC.

Match the types of variables in the flowgraph is very important. Later you will learn that the types of variables are color-coded. In Figure 4, `byte` is displayed in purple and `complex` is displayed in blue.

At this point, try to understand the workings of GNU Radio by observing the flowgraph carefully. You can double click on each block to see the parameters and try to understand their meaning. In Figure 4, the RF carrier is represented by the `Signal Source` block, which multiplies the baseband signals by the `Multiply` block. Therefore, the most important task is to design baseband signal processing blocks. In real-world implementations, the RF part of the system is usually implemented in hardware.

`ToDo:` **Question 3 (*2 marks*).** Match up the blocks in Figure 4 with as many blocks in Figure 5 as you can. Appreciate the fact that complex numbers can efficiently manage I and Q channels simultaneously.

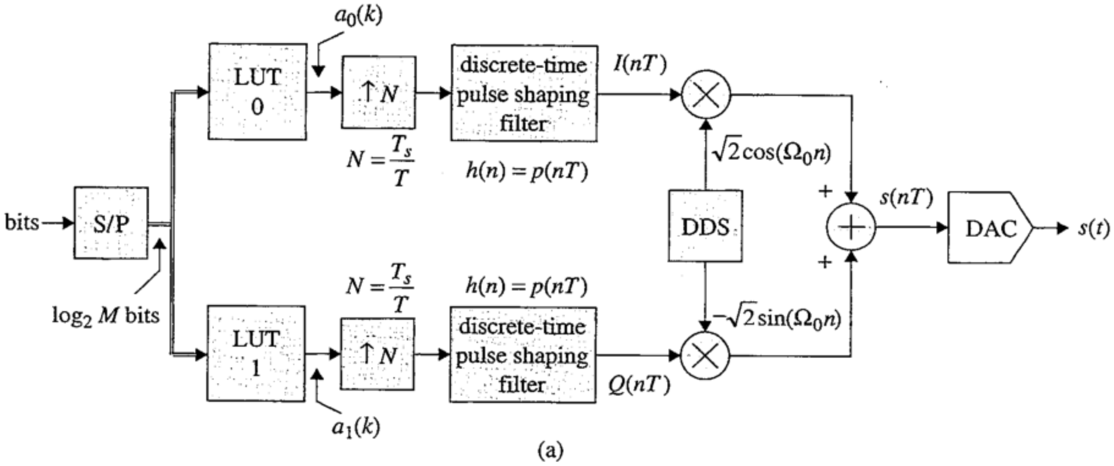


Figure 5: Discrete-time processing for QAM: (a) A discrete-time QAM modulator. (Source: Rice, Figure 5.3.16)

Step 3: Without changing any parameter, click the green `Execution` button to run the flowgraph. You should see similar plots as shown in Figure 6.

The green **`Execute`** button is on the toolbar of the GRC window.

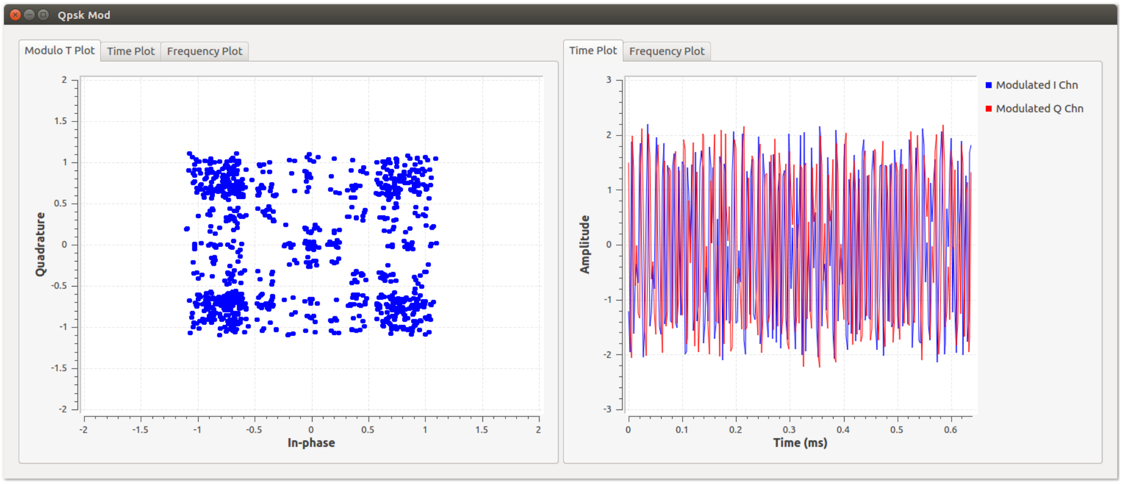


Figure 6: A display after executing the flowgraph `qpsk_mod_skeleton.grc`.

You can

- Click on the legend icon to switch off/on the trace.
- Click on the tabs to switch among different plots.
- Select a part of the plot to zoom in (right-click to reset the view).
- Middle-click on the plot to access more features.

`ToDo:`

1. Explore the functions of the plot sink and understand every feature provided by the graphical user interface (GUI), such as `Stop`, `Grid`, `Line Style`, `Save`, and `Control Panel`.
2. In the time plots, turn off the Q channel. Compare the waveforms of the I channel.
3. In the frequency plots, identify the key features of the spectrum. Are these observations consistent with your expectation?
4. There is no channel simulator in the flowgraph. Can you figure out a reason why the Modulo T plot seems to be 'noisy'? What happens inside the block `Constellation Modulator`?
5. In the block `Constellation Modulator`, try different values for `Excess BW` between 0.1 to 1 and compare the corresponding Modular T plot. What value for `Excess BW` is better and why?[1]

This exercise is one of the keys to understand the connections between digital signals and analog waveforms.

Task 2: Implement QPSK Modulator in GNU Radio

As you have seen in the previous task, the `Constellation Modulator` works as a black box. And you only have limited control over a few parameters. In this task, we will have a closer look at this block and implement its functions.

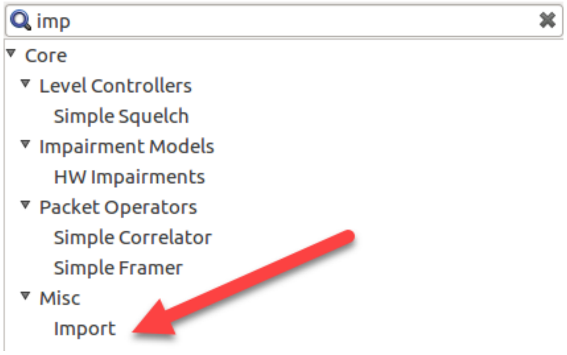
[1] You will find out later in the subject why in many practical systems, raised cosine pulse shaping filters are often preferred over rectangular pulse shaping filters.

Step 1: Preparation

Create a new flowgraph by clicking the `New` button on the toolbar. There are two default blocks in the flowgraph, `Options` and `Variable`. The `Options` block defines some basic parameters for the flowgraph and the `Variable` block defines the sampling rate variable, which can be used globally in the flowgraph.

In the GRC, we can conveniently import `Python` packages and declare global variables for the flowgraph. In the search box, type in keywords to look for the `Import` block in the toolbox, as shown in Figure 7.

The `New` button is on the top-left corner of the GRC toolbar.



imp

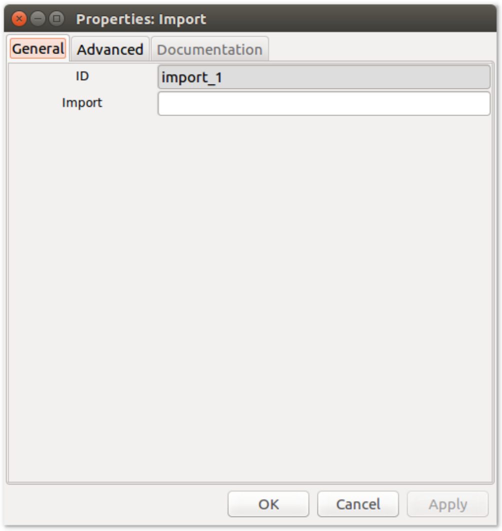
- Core
  - Level Controllers
    - Simple Squelch
  - Impairment Models
    - HW Impairments
  - Packet Operators
    - Simple Correlator
    - Simple Framer
  - Misc
    - Import

Figure 7: Using keyword `imp` to look for the `Import` block.

Click the `Search` button to look for blocks in the toolbox.

`ToDo:` Select the `Import` block in the toolbox on the right hand side and drag it into the flowgraph. Double click on the block and set the parameters as hinted in Figure 8.

Import
Import:



Properties: Import

General | Advanced | Documentation

ID: import_1

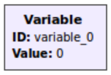
Import:

Name	Value
Import	`import numpy as np`

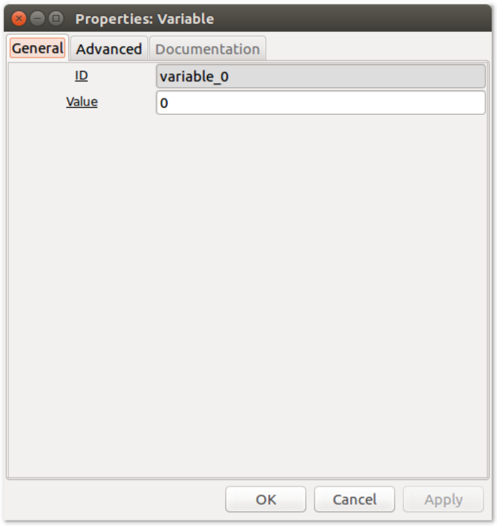
Figure 8: Setting parameters for the `Import` block in GRC.

`NumPy` is the fundamental package for scientific computing with `Python`.

`ToDo:` Drag and drop a `Variable` block into the flowgraph. Double click on the block and set the parameters as hinted in Figure 9. This variable is used to define the `sample-per-symbol` rate.



Variable
ID: variable_0
Value: 0



Properties: Variable

ID: variable_0

Value: 0

Name	Value
ID	sps
Value	8

sample-per-symbol is used later in the pulse-shaping block.

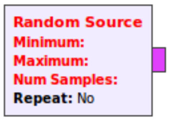
Figure 9: Setting parameters for the Variable block in GRC.

Step 2: Implement Source Block

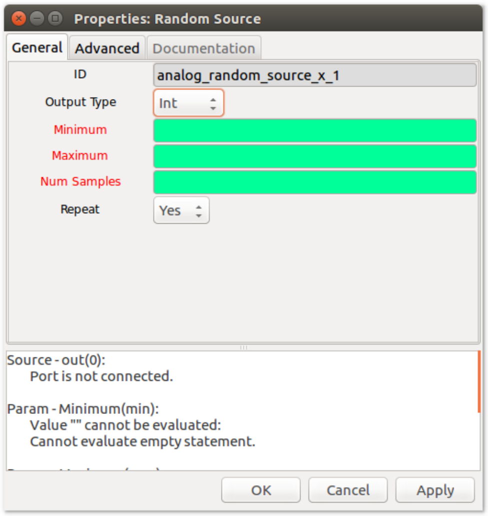
In this step, we repeatedly generate 1000-byte random sequences as the input to the system.

Question 4 (*1 mark*). What is the decimal range that a byte can represent?

ToDo: Drag and drop a Random Source block into the flowgraph. Double click on the block and set the parameters as hinted in Figure 10. Pay attention to any warning that appears in the bottom area of the Properties dialog. This block generates Num Samples number of random numbers between Minimum (inclusive) and Maximum (exclusive).



Random Source
Minimum:
Maximum:
Num Samples:
Repeat: No



Properties: Random Source

ID: analog_random_source_x_1

Output Type: Int

Minimum:

Maximum:

Num Samples:

Repeat: Yes

Source - out(0):
Port is not connected.

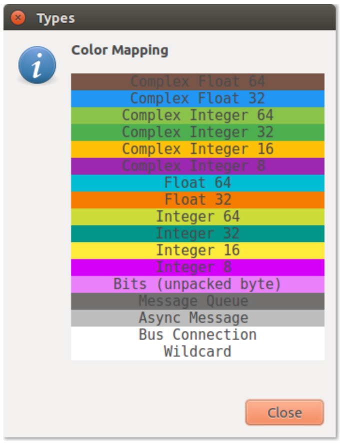
Param - Minimum(min):
Value "" cannot be evaluated:
Cannot evaluate empty statement.

Name	Value
Output Type	*A proper selection*
Minimum	*A proper value*
Maximum	*A proper value*
Num Samples	1000
Repeat	Yes

In GNU Radio/Python, the actual stop number is one less than the nominal number Maximum.

Figure 10: Setting parameters for the Random Source block in GRC.

The output type of each block is color-coded, as shown in Figure 11. It is important to choose the proper type for each block as to be able to connect them together.



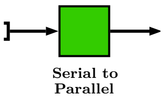
Types

Color Mapping

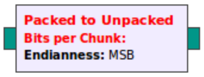
Complex Float 64
Complex Float 32
Complex Integer 64
Complex Integer 32
Complex Integer 16
Complex Integer 8
Float 64
Float 32
Integer 64
Integer 32
Integer 16
Integer 8
Bits (unpacked byte)
Message Queue
Async Message
Bus Connection
Wildcard

Figure 11: Color-coded type information can be obtained in `Menu\Help\Types`.

Step 3: Implement Serial-to-Parallel Block



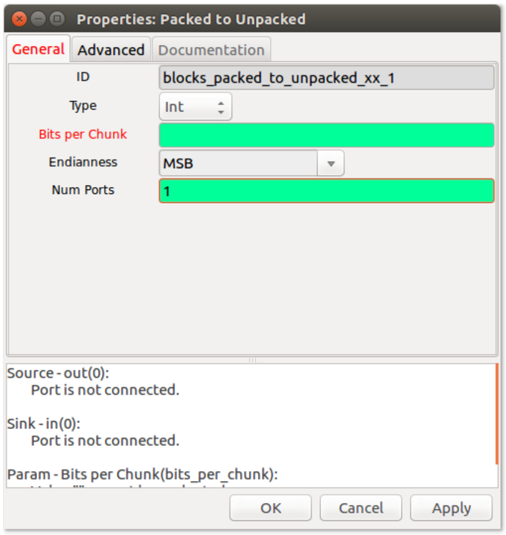
In practice, binary information is packed into bytes. Depending on the modulation scheme, we have to extract a portion of bits out of each byte to map to the constellation points. This can be done in parallel.



`ToDo:` Drag and drop a `Packed to Unpacked` block into the flowgraph to unpack bytes into bits.

Question 5 (*1 mark*). Remembering that we are implementing a QPSK modulator, how many bits per symbol should we have?

Double click on the block and set the parameters as hinted in Figure 12.



Properties: Packed to Unpacked

ID	blocks_packed_to_unpacked_xx_1
Type	Int
Bits per Chunk	
Endianness	MSB
Num Ports	1

Source - out(0):
Port is not connected.

Sink - in(0):
Port is not connected.

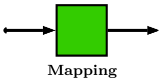
Param - Bits per Chunk(bits_per_chunk):

Name	Value
`Type`	*A proper selection*
`Bits per Chunk`	*A proper value*
`Endianness`	`MSB`

Endianness refers to the sequential order of bits transmission over a digital link. Words may be represented in big-endian or little-endian format, depending on whether bits are numbered from the big end (most significant bit (MSB)) or the little end (least significant bit (LSB)).

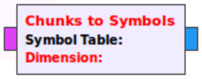
Figure 12: Setting parameters for `Packed to Unpacked` block in GRC.

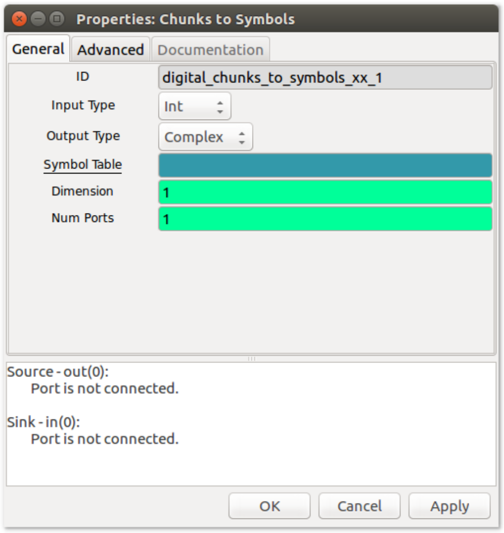
Step 4: Implement Mapping Block



Mapping is one of the key components in a modulator, which maps bits to the points on the constellation diagram. Use your QPSK mapping table in the pre-workshop question. You will need the column **Index** and **Complex Number**.

`ToDo:` Drag and drop a `Chunks to Symbols` block into the flowgraph. Double click on the block and set the parameters as hinted in Figure 13.


Chunks to Symbols
Symbol Table:
Dimension:


Properties: Chunks to Symbols

ID: digital_chunks_to_symbols_xx_1
Input Type: Int
Output Type: Complex
Symbol Table:
Dimension: 1
Num Ports: 1

Source - out(0):
Port is not connected.

Sink - in(0):
Port is not connected.

Name	Value
`Input Type`	*A proper selection*
`Output Type`	*A proper selection*
`Symbol Table`	*A proper list of comma separated complex numbers*

Figure 13: Setting parameters for `Chunks to Symbols` block in GRC.

Step 5: For technical reasons, a `Throttle` block is needed in the flowgraph where no real hardware is connected. This prevents GNU RADIO from consuming all CPU resources when the flowgraph is not being regulated by external hardware (i.e.: audio source/sink or USRP source/sink). You can think of the throttle block as the speed limiter, without which the CPU of the computer will be overloaded.

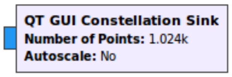
Throttle
Sample Rate:

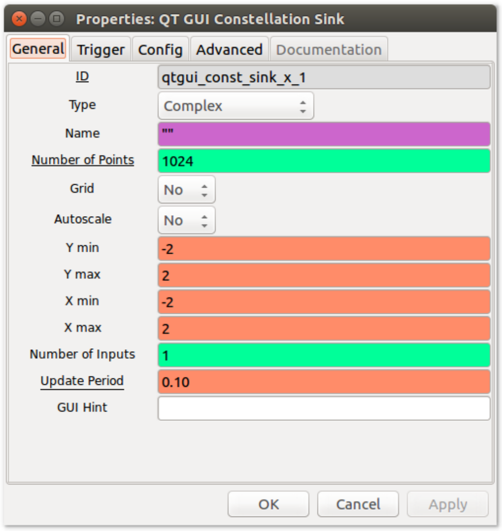
You only need one `Throttle` block in a flowgraph.

`ToDo:` Drag and drop a `Throttle` block into the flowgraph and and leave the parameter as default.

Step 6: At this point, we want to observe and verify the mapping you just implemented against the constellation diagram. To this end, GNU RADIO provides many graphical instruments that can be used, such as time sink, frequency sink, constellation sink, waterfall sink, etc.

`ToDo:` Drag and drop a `QT GUI Constellation Sink` block into the flowgraph. Double click on the block and set the parameters as hinted in Figure 14.


QT GUI Constellation Sink
Number of Points: 1.024k
Autoscale: No



Properties: QT GUI Constellation Sink

ID	qtgui_const_sink_x_1
Type	Complex
Name	""
Number of Points	1024
Grid	No
Autoscale	No
Y min	-2
Y max	2
X min	-2
X max	2
Number of Inputs	1
Update Period	0.10
GUI Hint	

Name	Value
Type	*A proper selection*
Name	Constellation
Grid	Yes
GUI Hint	0,0,2,1

`GUI Hint` defines the position and size of the sink displayed in the output window as {`row`,`column`, `row-span`,`column-span`}.

Figure 14: Setting parameters for `QT GUI Constellation Sink` block in GRC.

Step 7: Now it is time to connect all blocks together. In GNU Radio, most of the blocks are implemented in `C++` and connected by `Python` scripts. GRC is just a GUI that generates `Python` scripts that connect these `C++`-compiled blocks.

`ToDo:` Connect the blocks in sequence by clicking the output of the previous block and the input of the next block, as shown below.

Any RED color along the flowgraph indicates that there are errors either in the connections or in the parameter settings.

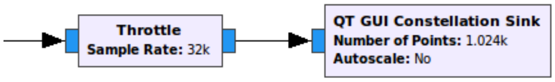


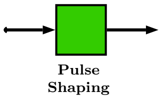
Figure 15: Connecting blocks in GRC.

Save and execute the flowgraph. You should see the same constellation diagram as you have drawn in the pre-workshop question.

Keep the `.grc` file with you. We will build the whole system upon this file later.

Step 8: Implement Pulse Shaping Block

You might wonder how to transmit complex numbers over the air. This is where pulse shaping comes into play. At the moment, we use a simple pulse, namely the rectangular pulse. As shown in Figure 16, a symbol, either the in-phase component or the qudrature component of a complex number, is firstly upsampled by a factor of `sample-per-symbol`, padding seven zeros in this case. Then they pass through a pulse shaping filter, which is a rectangular filter in this example. After digital-to-analog (D/A) conversion, a rectangular waveform is transmitted.



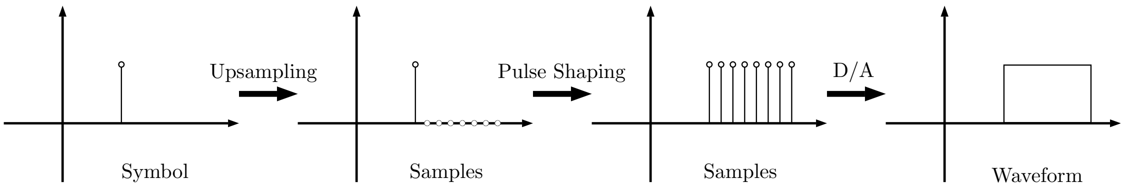
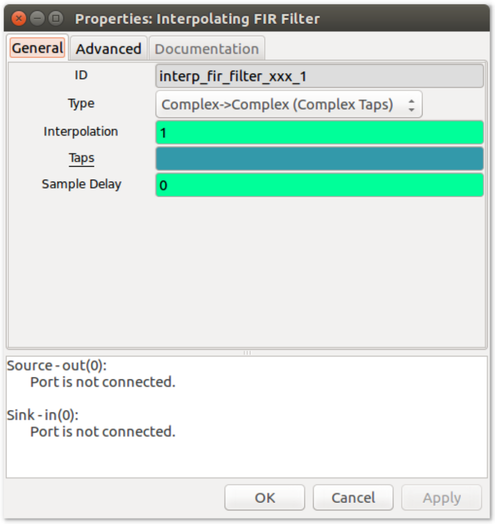


Figure 16: Pulse shaping process.

In the digital domain, a pulse shaping filter is essentially a FIR digital filter.

ToDo: Drag and drop an `Interpolating FIR Filter` block into the flowgraph. Double click on the block and set the parameters as hinted in Figure 17. Connect its input to the output of the block `Throttle`.

Interpolating FIR Filter
Interpolation:
Taps:


Properties: Interpolating FIR Filter

ID interp_fir_filter_xxx_1
Type Complex->Complex (Complex Taps)
Interpolation 1
Taps
Sample Delay 0

Source - out(0):
Port is not connected.

Sink - in(0):
Port is not connected.

Name	Value
Type	*A Proper selection*
Interpolation	`sps`
Taps	`np.ones(sps)`

Figure 17: Setting parameters for `Interpolating FIR Filter` block in GRC.

Without loss of generality, let us examine the real component of the transmit signal for brevity. You can construct another branch for the imaginary component later.

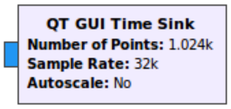
ToDo: Drag and drop a `Complex To Real` block into the flowgraph and connect its input to the output of the block `Interpolating FIR Filter`.

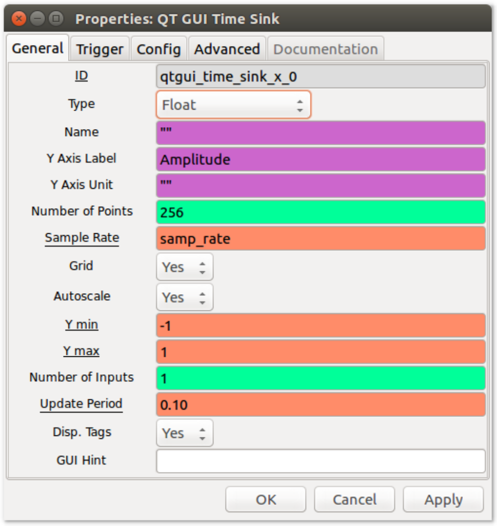
Complex To Real

ToDo: In order to observe the effect of pulse shaping, create another branch of `Interpolating FIR Filter + Complex to Real` and connect the input to the output of the block `Throttle` as well, but set the field `Taps` in the `Interpolating FIR Filter` to `1`.

Let us now observe the effect of the pulse shaping filter.

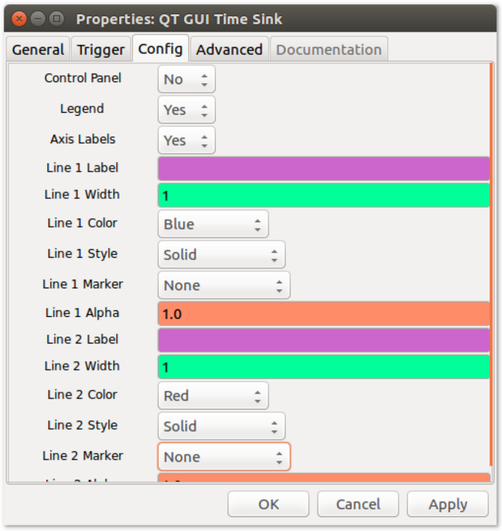
ToDo: Drag and drop a `QT GUI Time Sink` block into the flowgraph. Double click on the block and set the parameters as hinted in Figure 18. Connect its inputs to the two output interpolating branches respectively.


QT GUI Time Sink
Number of Points: 1.024k
Sample Rate: 32k
Autoscale: No



ID: qtgui_time_sink_x_0
Type: Float
Name: ""
Y Axis Label: Amplitude
Y Axis Unit: ""
Number of Points: 256
Sample Rate: samp_rate
Grid: Yes
Autoscale: Yes
Y min: -1
Y max: 1
Number of Inputs: 1
Update Period: 0.10
Disp. Tags: Yes
GUI Hint:

Name	Value
Type	*A proper selection*
Name	`Signals in Time Domain`
Number of Points	`256`
Sample Rate	`sample_rate * sps`
Grid	`Yes`
Autoscale	`Yes`
Number of Inputs	`2`
GUI Hint	`2,0,1,2`



Control Panel: No
Legend: Yes
Axis Labels: Yes
Line 1 Label:
Line 1 Width: 1
Line 1 Color: Blue
Line 1 Style: Solid
Line 1 Marker: None
Line 1 Alpha: 1.0
Line 2 Label:
Line 2 Width: 1
Line 2 Color: Red
Line 2 Style: Solid
Line 2 Marker: None

Name	Value
`Control Panel`	`Yes`
`Line 1 Marker`	`Circle`
`Line 2 Marker`	`Cross`

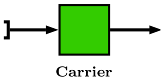
Figure 18: Setting parameters for `QT GUI Time Sink` block in GRC.

Save and execute the flowgraph.

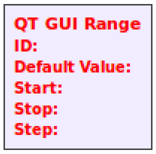
`ToDo:` Pause the time sink and zoom in to a proper level by left-clicking and dragging the mouse over the plot. Observe the effect of rectangular pulse shaping. You can restore the plot by right-clicking on the plot. You can also turn on/off a trace by clicking on its legend.

Step 9: Implement Carrier

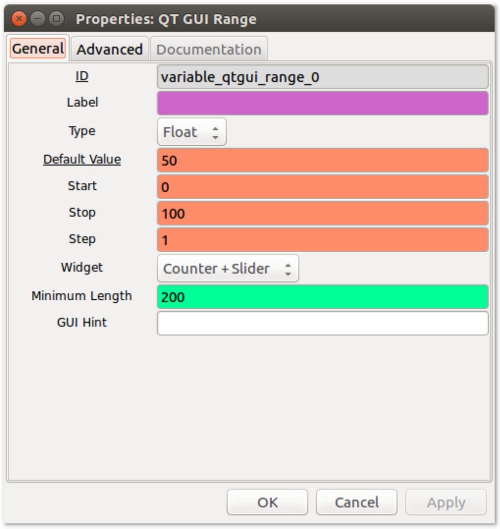
The last step of the modulator is to modulate the signals onto the carrier frequency f_c. Let us first create a variable to represent f_c.


Carrier

ToDo: Drag and drop a `QT GUI Range` block into the flowgraph. Double click on the block and set the parameters as hinted in Figure 19.



Those GUI components allow you to interact with the flowgraph and change parameters in real time.


Properties: QT GUI Range

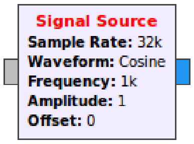
General | Advanced | Documentation

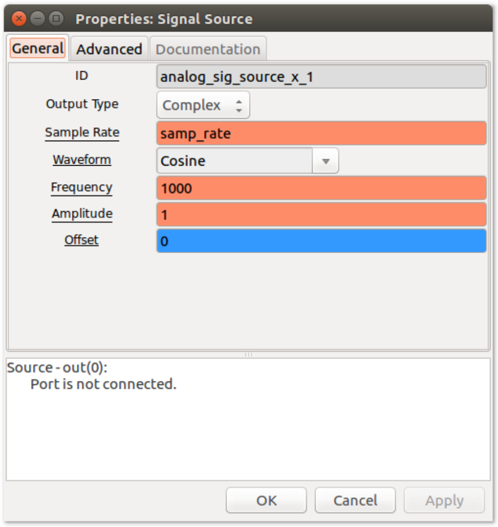
ID	variable_qtgui_range_0
Label	
Type	Float
Default Value	50
Start	0
Stop	100
Step	1
Widget	Counter + Slider
Minimum Length	200
GUI Hint	

Name	Value
ID	fc
Label	Carrier Frequency
Type	*A proper selection*
Default Value	50000
Start	1000
Stop	100000
Step	100
GUI Hint	1,1,1,1

Figure 19: Setting parameters for `QT GUI Range` block in GRC.

ToDo: Drag and drop a `Signal Source` block into the flowgraph. Double click on the block and set the parameters as hinted in Figure 20.




Properties: Signal Source

General | Advanced | Documentation

ID	analog_sig_source_x_1
Output Type	Complex
Sample Rate	samp_rate
Waveform	Cosine
Frequency	1000
Amplitude	1
Offset	0

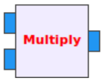
Source - out(0):
Port is not connected.

Name	Value
Output Type	*A proper selection*
Sample Rate	samp_rate * sps
Frequency	fc
Amplitude	np.sqrt(2)

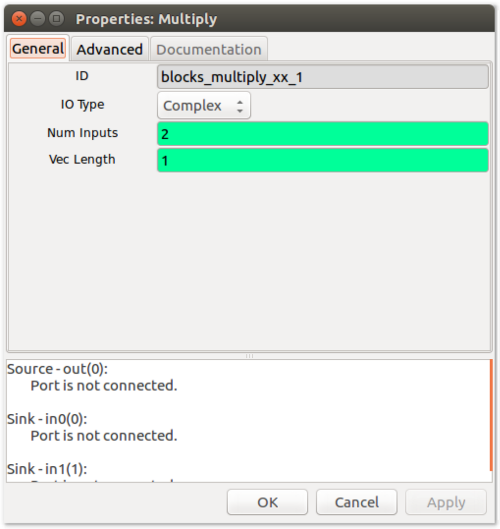
Figure 20: Setting parameters for `Signal Source` block in GRC.

Next, we multiply the signals with the carrier.

ToDo: Drag and drop a `Multiply` block into the flowgraph. Double click on the block and set the parameters as hinted in Figure 21. Connect its inputs



to the outputs of the `Signal Source` and the pulse shaping branch respectively.



ID	blocks_multiply_xx_1
IO Type	Complex
Num Inputs	2
Vec Length	1

Source - out(0):
  Port is not connected.

Sink - in0(0):
  Port is not connected.

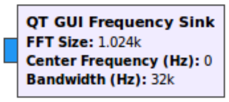
Sink - in1(1):

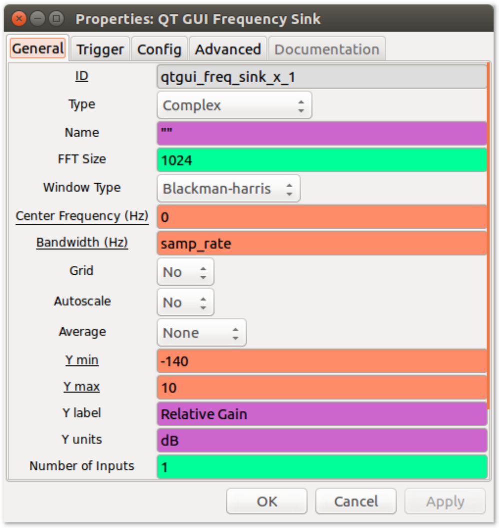
Name	Value
Type	*A proper selection*

Figure 21: Setting parameters for `Multiply` block in GRC.

Step 10: Lastly, let us observe the carrier-modulated signals in time and frequency domains.

`ToDo:` Drag and drop a `QT GUI Frequency Sink` block into the flowgraph. Double click on the block and set the parameters as hinted in Figure 22. Connect its input the output of the `Multiply` block.


QT GUI Frequency Sink
FFT Size: 1.024k
Center Frequency (Hz): 0
Bandwidth (Hz): 32k



ID	qtgui_freq_sink_x_1
Type	Complex
Name	""
FFT Size	1024
Window Type	Blackman-harris
Center Frequency (Hz)	0
Bandwidth (Hz)	samp_rate
Grid	No
Autoscale	No
Average	None
Y min	-140
Y max	10
Y label	Relative Gain
Y units	dB
Number of Inputs	1

Name	Value
Type	*A proper selection*
Name	`Spectrum`
`Window Type`	`Hann`
Bandwidth	`samp_rate * sps`
Grid	`Yes`
Average	`Low`
`GUI Hint`	`0,1,1,1`

Figure 22: Setting parameters for `QT GUI Frequency Sink` block in GRC.

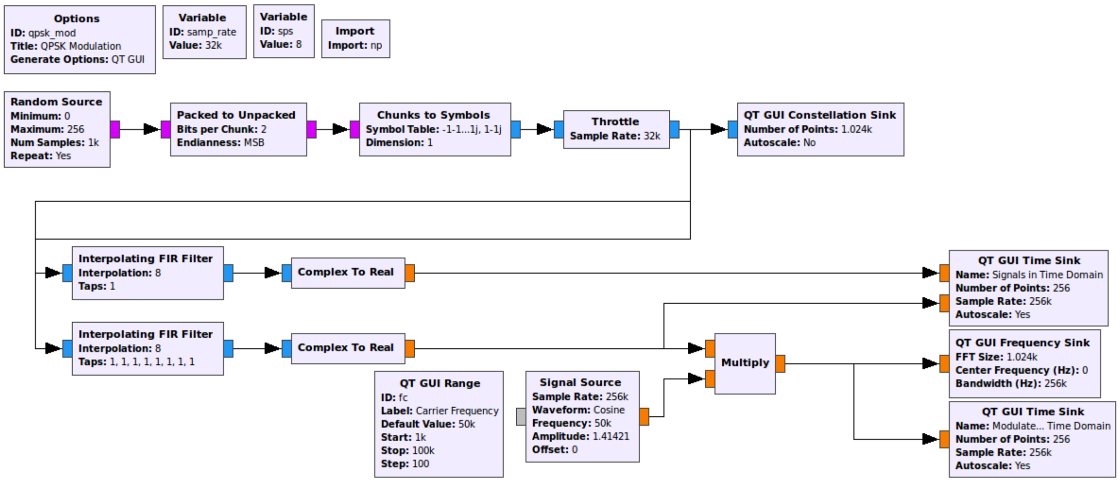
Drag and drop a `QT GUI Time Sink` block into the flowgraph. Set its parameters as hinted in Table 1 and connect its input the output of the `Multiply` block as well.

Name	Value
`Type`	*A proper selection*
`Name`	`Modulated Signals in Time Domain`
`Number of Points`	`256`
`Sample Rate`	`samp_rate * sps`
`Grid`	`Yes`
`Autoscale`	`Yes`
`GUI Hint`	`3,0,1,2`

Table 1: Setting parameters for `QT GUI Time Sink` block in GRC

Step 11: Verification (10 marks)

Show your working flowgraph to the demonstrator to check off the in-workshop assessment. You might be asked some questions relating to your flowgraph. For your reference, we provide a sample flowgraph and output for debugging, as shown in Figure 23 and Figure 24.



Options
ID: qpsk_mod
Title: QPSK Modulation
Generate Options: QT GUI

Variable
ID: samp_rate
Value: 32k

Variable
ID: sps
Value: 8

Import
Import: np

Random Source
Minimum: 0
Maximum: 256
Num Samples: 1k
Repeat: Yes

Packed to Unpacked
Bits per Chunk: 2
Endianness: MSB

Chunks to Symbols
Symbol Table: -1-1...1j, 1-1j
Dimension: 1

Throttle
Sample Rate: 32k

QT GUI Constellation Sink
Number of Points: 1.024k
Autoscale: No

Interpolating FIR Filter
Interpolation: 8
Taps: 1

Complex To Real

QT GUI Time Sink
Name: Signals in Time Domain
Number of Points: 256
Sample Rate: 256k
Autoscale: Yes

Interpolating FIR Filter
Interpolation: 8
Taps: 1, 1, 1, 1, 1, 1, 1, 1

Complex To Real

Multiply

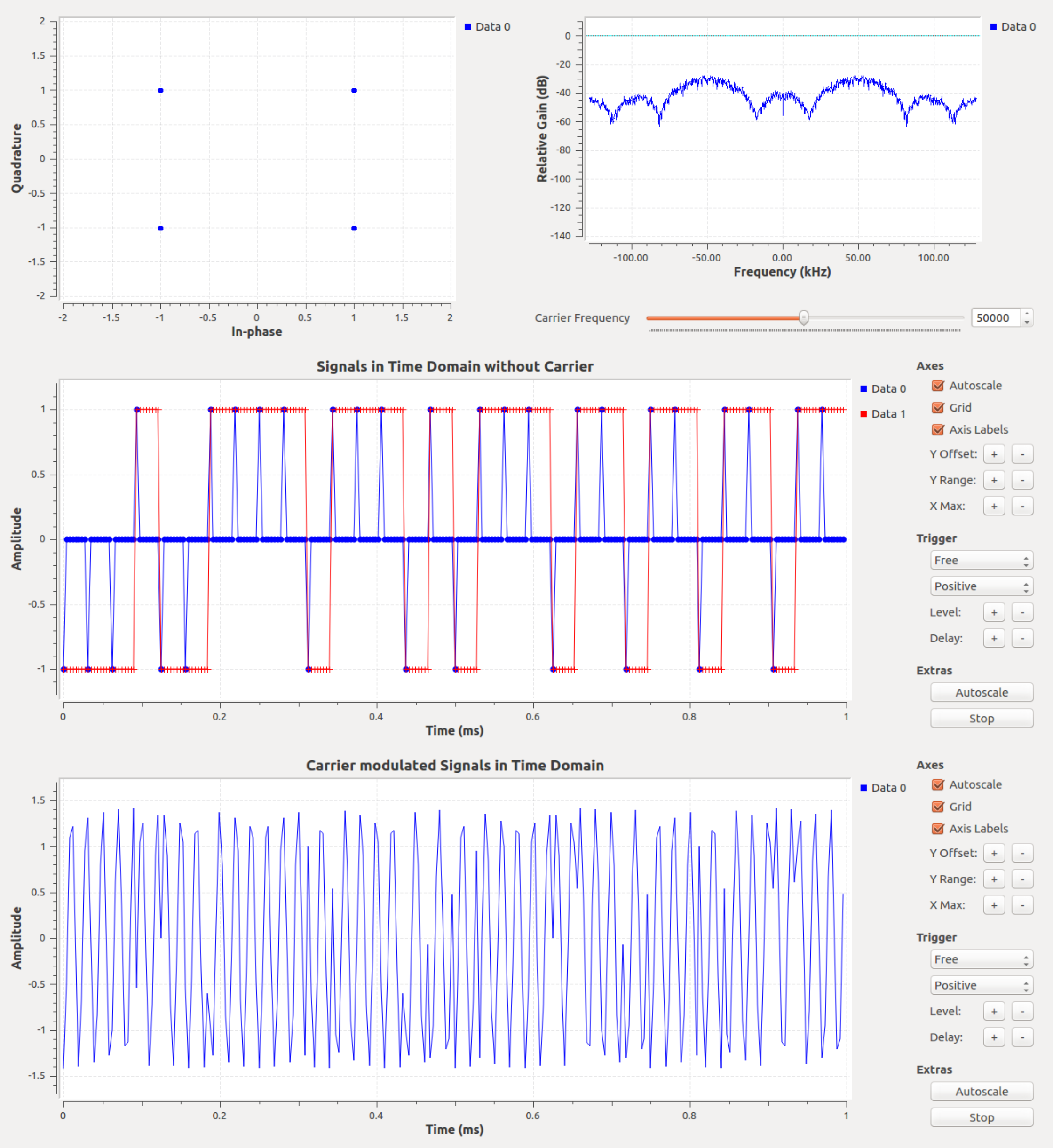
QT GUI Frequency Sink
FFT Size: 1.024k
Center Frequency (Hz): 0
Bandwidth (Hz): 256k

QT GUI Range
ID: fc
Label: Carrier Frequency
Default Value: 50k
Start: 1k
Stop: 100k
Step: 100

Signal Source
Sample Rate: 256k
Waveform: Cosine
Frequency: 50k
Amplitude: 1.41421
Offset: 0

QT GUI Time Sink
Name: Modulate... Time Domain
Number of Points: 256
Sample Rate: 256k
Autoscale: Yes

Figure 23: Sample flowgraph and output.



Carrier Frequency 50000

Axes: ☑ Autoscale ☑ Grid ☑ Axis Labels

Trigger: Free, Positive

Figure 24: Sample flowgraph and output.

IV Report Tasks (*30 marks*)

Task 1: Implement a 16-QAM Modulator in GRC (*10 marks*)

`ToDo:` Implement a 16-QAM modulator in GNU RADIO by completing the following tasks:

(1) Complete the constellation mapping table for 16-QAM, similar to what you have done in the pre-workshop questions;

(2) Draw the constellation diagram with all values annotated;

(3) Implement a 16-QAM modulator in GRC.

(4) You should construct a complete 16-QAM modulator, including in-phase and quadrature components.

Task 2: Discussion and Reflection (*20 marks*)

You should structure your report as follows:

1. Introduction: Introduce your report, e.g., make some general comparisons between analog and digital communication systems, what are the motivations of using digital modulation, what different types of digital modulation schemes exist.

2. Background: Give a brief summary of modulation theory, e.g., complex number, signal space, Gray coding, etc. What is the connection between digital signals and analog waveforms? What are the trade-offs of using high-order modulations schemes?

3. Implementation: Describe the procedure of your implementation of the 16-QAM modulator **in detail**, including the flowgraph and outputs. Use screen shots and plots to show that your implementation works.

4. Conclusion: Draw conclusions based on your observations and analysis. Discuss the trade-offs of the different modulation schemes. Can you think of some potential issues that the receiver might encounter? Think about how the receiver would retrieve the sequence of message symbols from the received waveform when there is channel noise and/or channel distortion?

5. Reference: Use IEEE reference style. See `LMS/Workshops/Workshop week 3: source coding 2018/ieeecitationref.pdf` for details.

End of Workshop