NI USRP Lab: DQPSK Transceiver Design

1 Introduction

1.1 Aims

This Lab aims for you to:

- understand the USRP hardware and capabilities;
- build a DQPSK receiver using LabVIEW and the USRP.

By the end of this exercise, you should be able to:

- understand how to build a communications system using NI USRP;
- appreciate the challenges in designing a communications system;
- link the theory with practical implementation.

In this exercise you will use the NI USRP-2920 equipment and build the receiver using LabVIEW.

Have Fun,

Mohammed El-Hajjar
Rob Maunder
Michael Ng

1.2 Preparation

Before the laboratory session begins, complete the "Introduction to LabVIEW" examples.

Additionally, read through Section 1.3 to understand the hardware you will use in the experiment and Section 2.1 to understand the theory of the system you are building.

1.3 NI USRP

The Universal Software Radio Peripherals, or USRP for short, is a software reconfigurable RF hardware from National Instruments to build digital communication systems. In this Lab you will use NI USRP-2920 hardware shown in Figures 1 and 2. Figure 1 shows a USRP connected to a PC, which acts as a software-defined radio. The USRP forms the RF front-end of the transceiver while all signal processing is done in LabVIEW on the PC. The PC controls the USRP through the Gigabit Ethernet cable connecting the two together.

In the transmitter part, the LabVIEW processes the signal, such as OFDM modulation, and passes the I/Q signal to the USRP over the Gigabit Ethernet. The USRP upconverts the signal to a RF before amplifying and transmitting the signal over the air. The USRP-2920

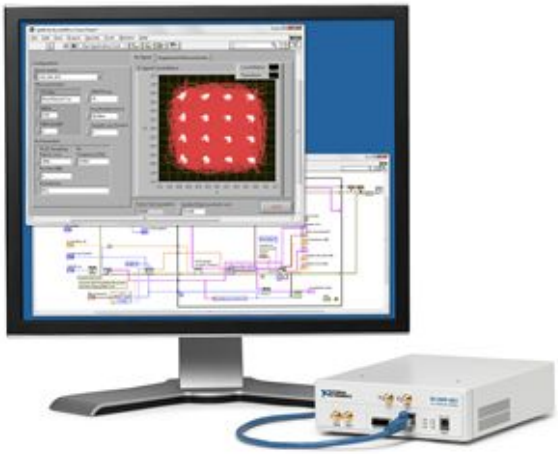


Figure 1 NI USRP setup.

used in this lab supports frequencies in the range 50 MHz to 2.2 GHz with up to a 25 MHz bandwidth, which covers the frequency range for applications including broadcast FM, public safety, land-mobile, low-power unlicensed devices, sensor networks, cellphones, amateur radio and GPS.

The NI USRP is also capable of receiving the signal, where the received signal is mixed down from RF using a direct-conversion receiver to baseband I/Q components. The digitized I/Q data follows parallel paths through a digital down-conversion process that mixes, filters and decimates the input signal to a user-specified rate. The down-converted samples are passed to the host computer over a standard Gigabit Ethernet connection. Figure 3 shows a block diagram of the USRP for the transmit and receive channels.

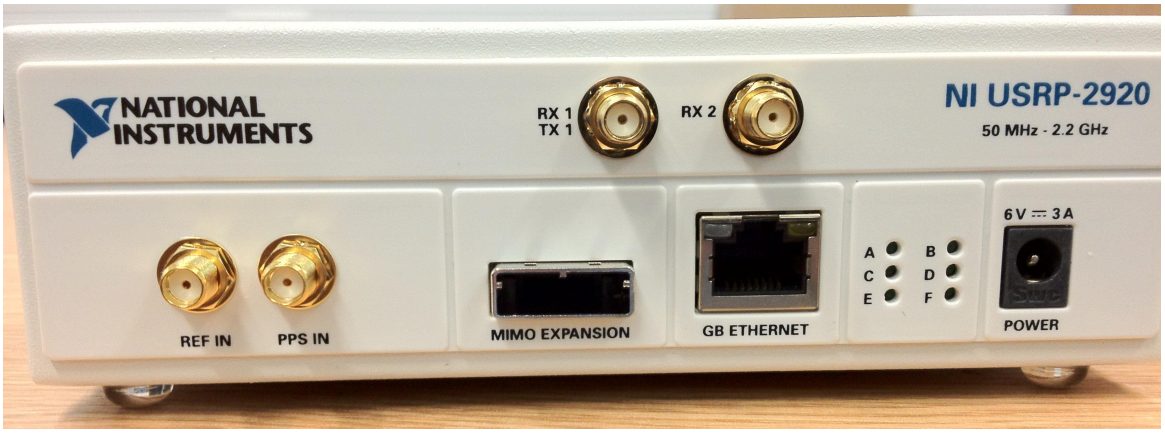


Figure 2 NI USRP front panel.

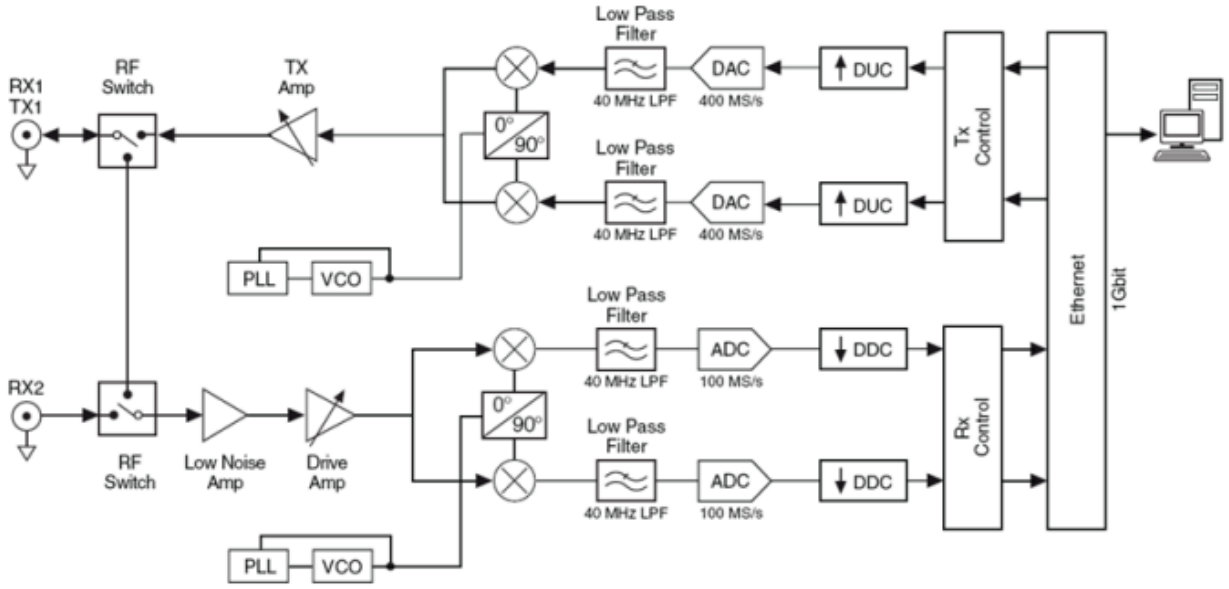


Figure 3 NI USRP-2920 system block diagram.

1.4 Front panel and Connections

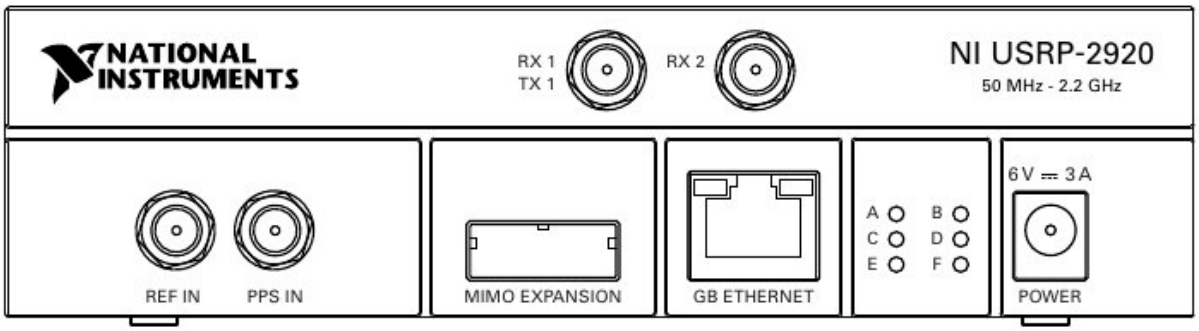


Figure 4 NI USRP-2920 front panel.

Figure 4 shows the front panel for the NI USRP-2920 device. In this lab experiment you will use one USRP device to detect signals transmitted from another USRP device. The main USRP device has already been setup by the lab demonstrator as the transmitter. You need to setup the USRP device, given to you, as the receiver by following the following steps:

1. Connect the power cable to the USRP as shown in Figure 5.
2. Connect the Ethernet cable and the antenna to the USRP as shown in Figure 6.
3. When you have connected the other end of the Ethernet cable to the PC and the electricity for the USRP device is turned on, navigate to **Start** >> **programs** >> **National Instruments** >> **NI-USRP** >> **NI-USRP Configuration Utility**.
4. When the USRP configuration utility starts, you should get a device connected with a certain IP address, i.e. 10.0.0.3, for the receiver as shown in Figure 7.

Figure 5 NI USRP-2920 front panel connections.

Figure 6 NI USRP-2920 front panel connections.

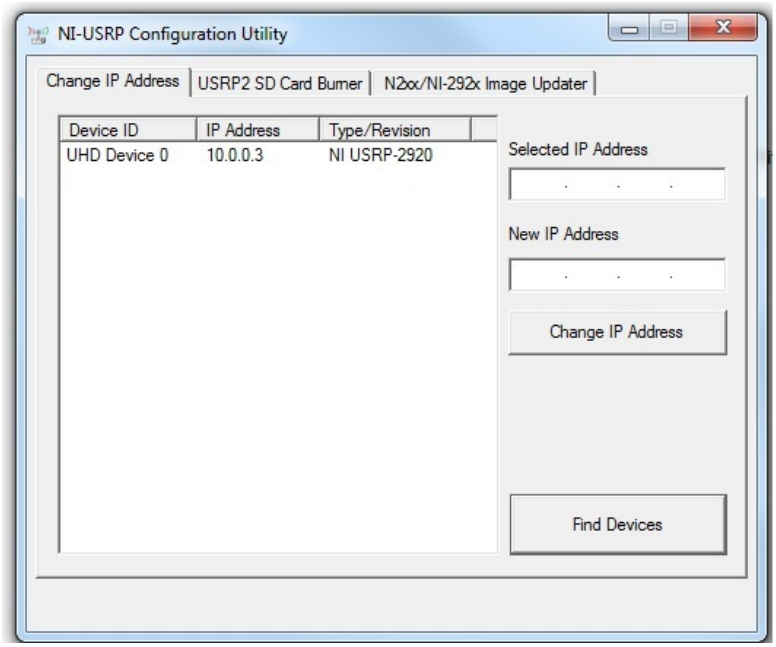


Figure 7 NI USRP configuration utility.

2 DQPSK Receiver Implementation

2.1 Theory

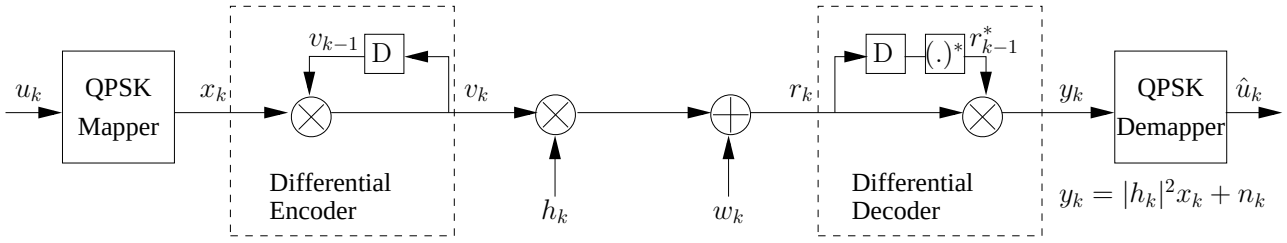


Figure 8 Baseband system block diagram of DQPSK transmission and detection.

The main purpose of differential modulation is to avoid the need for channel estimation during the demodulation of the received signal. The baseband system block diagram of DQPSK transmission and detection is shown in Figure 8. More specifically, the QPSK mapper maps a two-bit integer u_k to a complex-valued symbol x_k. Then the differential encoder will encode the QPSK symbol to a new complex-valued symbol as:

$$v_k = v_{k-1}x_k \ , \tag{1}$$

for transmission over the channel. The received signal is given by:

$$r_k = h_k v_k + w_k \ , \tag{2}$$

where $h_k = |h_k|e^{\angle h_k}$ and $w_k = |w_k|e^{\angle n_k}$ are the complex-valued channel fading and noise in the baseband. The variance of noise w_k is given by N_0. Note that, the channel fading rate has to be relatively slow in order for the differential modulation to work, where we assume that:

$$h_k = h_{k-1} \ . \tag{3}$$

The differential decoded signal is given by:

$$y_k = r_k r^*_{k-1} \ , \tag{4}$$

where r^*_{k-1} is the complex-conjugate of the $(k-1)$th received signal. Based on Equations (1), (2) and (3), we can rewrite Equation (4) as:

$$\begin{aligned} y_k &= (h_k(v_{k-1}x_k) + w_k)\ \left(h^*_k v^*_{k-1} + w^*_{k-1}\right) &(5)\\ &= |h_k|^2|v_{k-1}|^2 x_k + w_k w^*_{k-1} + w_k h^*_k v^*_{k-1} + h_k(v_{k-1}x_k)w^*_{k-1} &(6)\\ &= |h_k|^2 x_k + n_k \ , &(7)\end{aligned}$$

where we have $|v_{k-1}|^2 = 1$ and $n_k = w_k w^*_{k-1} + w_k h^*_k v^*_{k-1} + h_k(v_{k-1}x_k)w^*_{k-1}$. The variance of the effective noise is given by $2|h_k|^2 N_0$, where the expectation of the channel magnitude is unity, i.e. $E\{|h_k|^2\} = 1$. Hence, we have removed the dependency of the channel phase at the cost of doubling the noise power. The magnitude of the kth channel can be further estimated as:

$$\begin{aligned} |\hat{h}_k|^2 &= r_k r^*_k &(8)\\ &= |h_k|^2 + \epsilon_k \ , &(9)\end{aligned}$$

where $\epsilon = h_k v_k w^*_k + w_k h^*_k v^*_k + |w_k|^2$. However, since the term $|h_k|^2$ in Equation (7) is always positive, you can make a hard-detection of x_k by simply checking the phase of y_k, without the need for the estimation in Equation (8).

2.2 Implementation in LabVIEW

In this lab you will build the DQPSK demodulation using LabVIEW. An incomplete DQPSK demodulation sub VI has been given, namely **DemodulateSubVI.vi**. Your task is to complete this sub VI and insert it into **DPSK_rx.vi** for detecting DQPSK signals transmitted from the USRP transmitter in the lab. You should first test your **DemodulateSubVI.vi** without the USRP hardware using **DPSK_tx_0.vi**, which is a modified VI file for the transmitter.

Please open the **DPSK_tx_0.vi** file. When it is loaded, please press "ctrl-T" to open up both the Front Panel and the Block Diagram. You will find from the Block Diagram of **DPSK_tx_0.vi**, that **DemodulateSubVI.vi** has been connected inside the **Modulate** state. The icon of **DemodulateSubVI.vi** has a green top with letters "RX" and a white bottom with letters "DQPSK". There are five inputs (controls) and four outputs (indicators) in **DemodulateSubVI.vi**.

Now you may open **DemodulateSubVI.vi** by double-clicking on its icon. Again, you may open both its Front Panel and Block Diagram by pressing "ctrl-T". Please follow the following steps to complete your demodulator in **DemodulateSubVI.vi**.

2.2.1 Root Raised Cosine Filter

Apply the root raised cosine shaping filter to the received and down-converted signal.

- ❏ Select **View** >> **Functions Palette** from the LabVIEW menu.
- ❏ Expand the Programming palette by selecting  next to **Programming** on the Functions palette.
- ❏ Select the **Signal Processing** >> **Filters** >> **Advanced FIR Filtering** palette.
- ❏ Select the **FIR Filter** Express VI and add the **FIR Filter** Express VI to the Block Diagram of DemodulateSubVI.vi.
- ❏ If the **context help** window is not open, show the **context help** window and hover over the **FIR Filter** Express VI to understand what it does and its connections.
- ❏ Right-click the **FIR Filter** VI and select **Visible Items** >> **Label** to show the label above the express VI. Then, right-click the **FIR Filter** VI again and select **Properties**. Write **Root Raised Cosine Filter** in the label. Note here that this is a label and hence you **do not** need to include .vi in the label.
- ❏ Connect the **X** input of the **FIR Filter** VI[1] to the **Input Signal** array.
- ❏ Connect the **FIR Coefficients** input of the **FIR Filter** VI to the **Filter Coefficients** array.
- ❏ save the VI.

2.2.2 Downsampling

The Downsampling function has been implemented in the subVI called **Downsample(SubVI).vi**. Insert and connect **Downsample(SubVI).vi** into the block diagram of **DemodulateSubVI.vi** as follows:

[1]Note that when you are searching for the VI, look for the VI called **FIR Filter** and not **FIR Filter VI**.

- ❏ From the **Function palette**[2], select **Select a VI**, then navigate to and select **Downsample(SubVi).vi** from the window.
- ❏ Connect the output of the **FIR Filter** to the **samples** input of the **Downsample(SubVi)**.
- ❏ Connect the **samples per symbol** control inside the block diagram to the **samples per symbol** input of the **Downsample(SubVi)**.
- ❏ Connect the **symbols per frame** control inside the block diagram to the **symbols per frame** input of the **Downsample(SubVi)**.
- ❏ Add a **Free Label**[3] above the **Downsample** sub VI and type "Downsample" inside the **Free Label**.
- ❏ save the VI.

Testing
It is important to test your code while building it. Hence, at this stage it is essential that you compare the output of your **Downsampling** with the input of the **Upsampling** in the transmitter. The following steps are a first order approach in the debugging process, i.e. this is normally the first step you do in the debugging process and if you find that the following steps do not satisfy you that your code is functioning as expected, then further debugging should be implemented, such as using graphs to plot the difference between the input and output. However, this is outside the scope of this experiment.

Please implement the following steps in order to compare the output of the **Downsampling** with the input of the **Upsampling** in the transmitter:

- ❏ In the **DemodulateSubVi** sub VI, there are some indicators included in order to save the output of different receiver modules for debugging purposes.
- ❏ Connect the output of the **Downsample** sub VI to the **Downsample Output** indicator.
- ❏ Open the **DPSK_tx_0** VI. Observe that the **Downsample Output** of the **demodulateSubVi** sub VI is connected to an array indicator. Also note that the input to the **Upsample** sub VI in the transmitter is connected to an array indicator.
- ❏ Save the VI.
- ❏ Check to see if the "white arrow" of the run button at the top left corner is broken. If it is broken, fix the problem. When the "white arrow" is not broken, save your implementation in **DemodulateSubVI.vi**.
- ❏ At this stage you are ready to run the code and compare the output of the **Downsample** with the input of the **Upsample**. The output of the **Downsample** in the receiver should match the input to the **Upsample** in the transmitter. However, note that there is delay in the filters and hence the first output in the receiver **Downsample** does not correspond to the first input to the transmitter **Upsample**. Try to find the point where the input and output match by looking through the values in the indicators.

[2]To remember what the **Function palette** is and how to open it, refer to the Lab preparation or to the beginning of this section.

[3]Search for Free Label Express VI and then add it.

2.2.3 Differential Demodulation

Now you need to implement the differential demodulation function in order to process the downsampled signals, as follows[4]

- ❏ From the **Function Palette**, select **Programming** >> **Structures** >> **For Loop**. On the block diagram, click and drag the mouse in a rectangular shape to specify the size of the **For Loop**. The input to the loop is an array of symbols and the for loop is used to operate on one symbol at a time.
- ❏ The **For Loop** has a **Loop Count** control and **Loop Index** indicator. In LabVIEW you can run a loop without specifying the loop counter if the input to the loop is an array. In this case, the loop runs for the length of the array.
- ❏ From the **Function Palette**, select **Programming** >> **Structures** >> **Feedback Node** and drop it inside the **For Loop**.
- ❏ Right-click the **Feedback Node** and select properties. Then click the option **Arrow points right**.
- ❏ From the **Function Palette**, select **Mathematics** >> **Numeric** >> **Complex** >> **Complex Conjugate** and drop it inside the **For Loop** at the right of the **Feedback Node**. Note that you can change the size of the **For Loop** by clicking on any corner or side of the loop and dragging.
- ❏ From the **Function Palette**, select **Mathematics** >> **Numeric** >> **Multiply** and drop it inside the **For Loop** at the right of the **Complex Conjugate**.
- ❏ Connect the output "symbols" of **Downsample(SubVi)** to the input(left) of **Feedback Node** through the **For Loop**.
- ❏ Connect the output of **Feedback Node** to the input of **Complex Conjugate**.
- ❏ Connect one input of the **Multiply** VI to the output of the **Downsample(SubVI)** through the **For Loop** and then connect the other input of the **Multiply** VI to the output of the **Complex Conjugate**.
- ❏ Hover your mouse to the star (which is the initializer Terminal) at the bottom of **Feedback Node**, then right-click >> **Create** >> **Constant**.
- ❏ Drag the **Constant** to the outside of the **For Loop** and write "1+0i" inside it. You may need to reconnect the **Constant** to the **Feedback Node**, if the connection is broken. This specifies the initial value of the feedback register; if this is not initialised or is initialised to '0', then the output of the differential demodulator will always be '0'.
- ❏ Label the **For Loop** as "Differential Demodulation".
- ❏ save the VI.

Testing
Test your differential demodulation by comparing the output of the **Multiply** in the **Differential Demodulation** to the input of the **Differential Modulation**. Connect the output of the **Differential Demodulation** to the **Differential Demodulation Output** indicator and run the code.

Compare the Differential Demodulation output and differential modulation input by looking at the indicators in the **DPSK_tx_0** VI front panel. Remember that there is delay due to filtering.

[4]Note you are still building the **DemodulateSubVI.vi** sub VI.

2.2.4 Convert Constellation Points to Symbols

This block implements the detection of the QPSK symbols from the Differential Demodulator, then convert the QPSK Constellation Points to integers. It can be done in various ways but a default subVI called **PSK demod(SubVI)** has been implemented.

- ❏ From the **Function palette** choose **Select a VI**, then navigate to and select **PSK demod(SubVI).vi** from the window.
- ❏ Connect the output of the **Multiply** inside your differential demodulator **Foor Loop** to the **received signals** input of the **PSK demod(SubVi)**, through the **Foor Loop**.
- ❏ Connect the **bits per symbol** input of **PSK demod(SubVi)** to the **bits per symbol** control.
- ❏ Label the **PSK demod(SubVi)** as "Convert Constellation Points to Symbols". This can be done by adding a **Free Label** and placing it above the sub VI.
- ❏ save the VI.

Testing
Test your **Convert Constellation Points to Symbols** module by comparing the output of the **Convert Constellation Points to Symbols** to the input of the **Convert Symbols to Constellation Points** in the transmitter. Connect the output of the **Convert Constellation Points to Symbols** to the **Gray Code Symbols** indicator and run the code.

Figure 9 shows the front panel of the **DPSK_tx_0** VI after running the code.

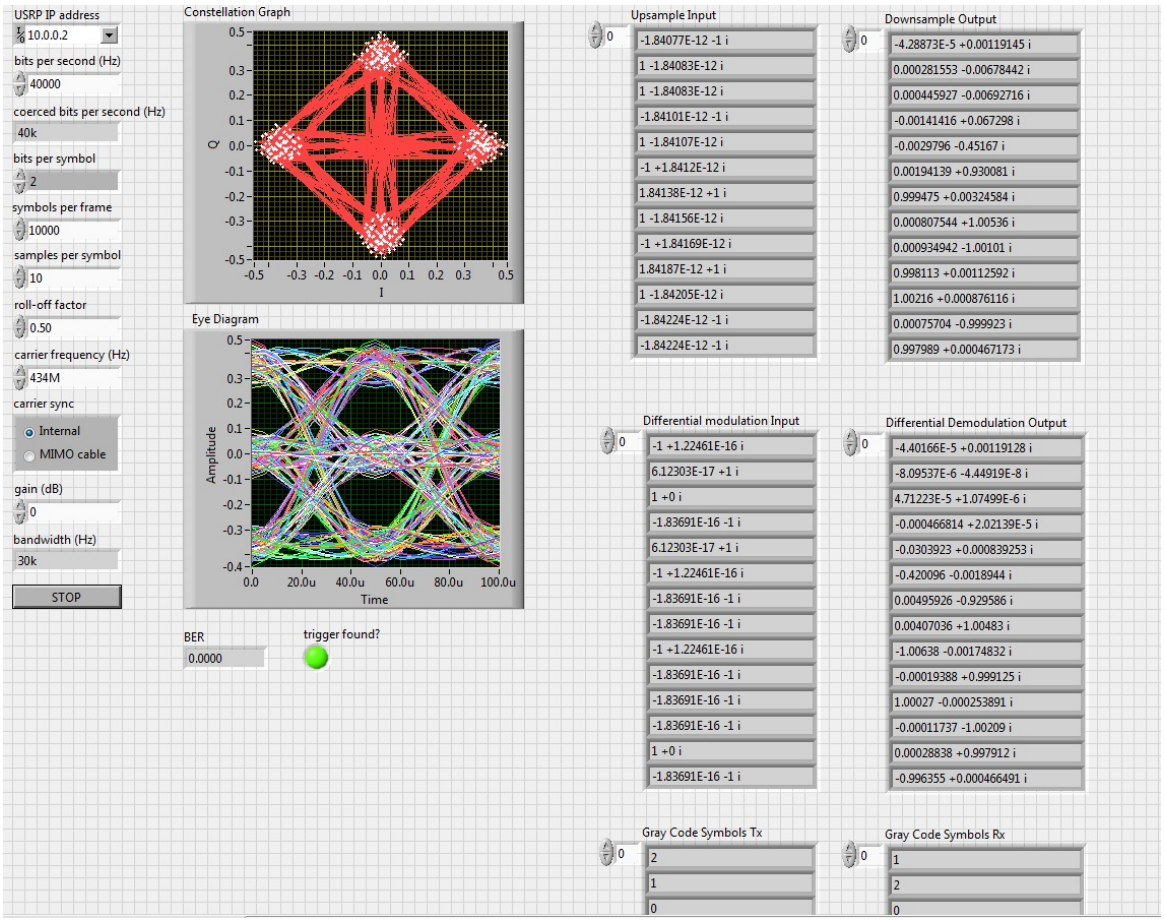


Figure 9 **DPSK_tx_0** VI front panel.

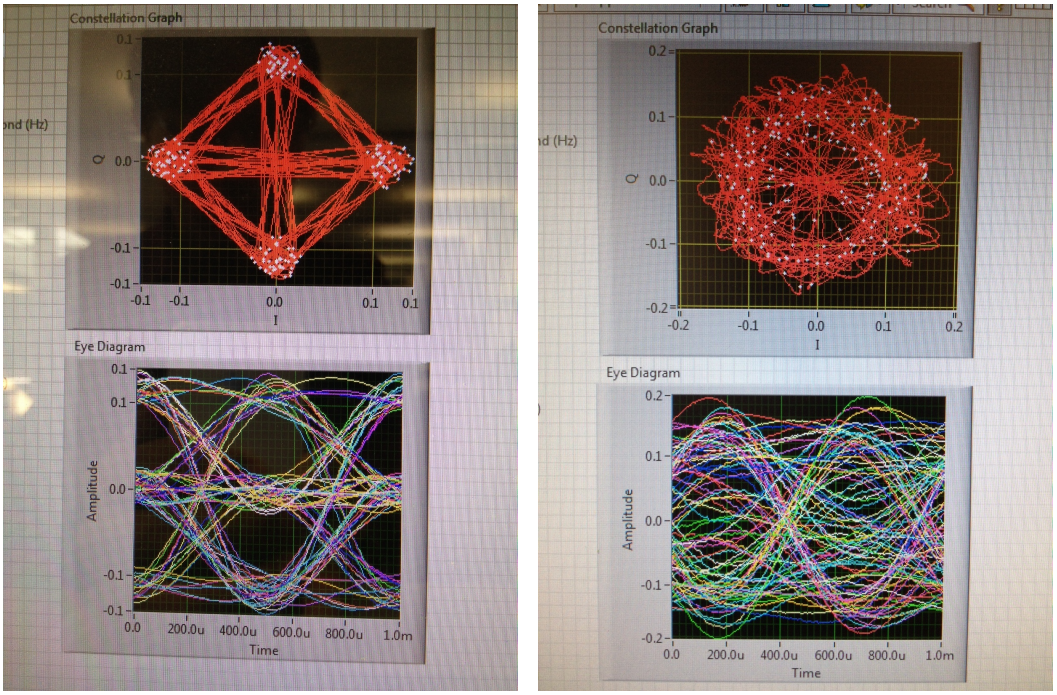
2.2.5 Gray Decode the Symbols

The output of the **Convert Constellation Points to Symbols** subVI is an array, which includes the indices of the points in the constellation. In the following, you will map the constellation point index to its corresponding Gray bit code.

- ❏ You should be still in **DemodulateSubVI.vi** block diagram.
- ❏ From the **Function Palette**, select **Programming** >> **Structures** >> **For Loop**. On the block diagram, click and drag the mouse in a rectangular shape to specify the size of the **For Loop**. The input to the loop is an array of symbols and the for loop is used to operate on one symbol at a time.
- ❏ Wire the output of the **Convert Constellation Points to Symbols** sub VI to the **For Loop**.
- ❏ Inside the **For Loop** insert a **MathScript Node**. The MathScript is an add-on module which allows you to add your Matlab code in LabVIEW.
  1. From the **Functions** palette, select **Programming** >> **Structures** >> **MathScript Node**.
  2. Click and drag the mouse in a rectangular shape to place the **MathScript Node** inside the **For Loop**.
- ❏ Right-click the **MathScript Node** and select Import from the shortcut menu to import the provided **decode.m** file.
- ❏ Right-click the top side of **MathScript Node** frame and select **Add Input** from the shortcut menu.
- ❏ Type **input_symbol** in the input terminal to add an input for the input_symbol variable in the Matlab script. This creates an input to the MathScript Node that you need to connect to the **tunnel** in the **For Loop** where the output of the **Convert Constellation points to Symbols** sub VI is connected.
- ❏ Add another input to the MathScript Node and call it **bits_per_symbol**. Connect this input to the **bits per symbol** control.
- ❏ Add an output to the MathScript Node on the right side and select **output_symbol** as the required output.
- ❏ Connect the output of the **MathScript Node** to the "output Symbol" indicator.
- ❏ Connect the output of the **MathScript Node** to the **input integers** input of the **Convert symbols to bits** sub VI, which has been pre-inserted into the block diagram.
- ❏ Label the **For Loop** as "Gray Decode the Symbols".
- ❏ Save the VI.

Testing of the module
Now you should be ready to test your implementation in **DemodulateSubVI.vi**.

- ❏ Check to see if the "white arrow" of the run button at the top left corner is broken. If it is broken, fix the problem. When the "white arrow" is not broken, save your implementation in **DemodulateSubVI.vi**.



(a) At the transmitter site.

(b) At the receiver site.

Figure 10 Front panels of the DQPSK system.

- ❏ Open the **DPSK_tx_0.vi** VI and click the run button of the Front Panel.
- ❏ You will find that the "Constellation Graph" and "Eye Diagram" of the transmitted signals has been shown at the Front Panel of **DPSK_tx_0.vi**.
- ❏ If your implementation of **DemodulateSubVI.vi** is correct, then the "trigger found?" button should be in bright green colour and the BER indicator should show "0.0000".

2.2.6 Testing with the USRP Hardware

Congratulations if you have achieved the above results. Once this is successful, you are ready to insert **DemodulateSubVI.vi** into the **DPSK_rx.vi** for detecting DQPSK signals using the USRP hardware.

- ❏ Open **DPSK_rx.vi** and show both its Front Panel and Block Diagram by pressing "ctrl-T".
- ❏ Now insert **DemodulateSubVI.vi** into the **Demodulate** state at the Block Diagram of **DPSK_rx.vi**.
- ❏ Connect the orange-colour input-wire marked by "Detected Signal from USRP" to the "Input Signal" input of **DemodulateSubVI.vi**.
- ❏ Connect the other four inputs of **DemodulateSubVI.vi** from the corresponding outputs of the **Unbundle By Name** Express VI.
- ❏ Connect the "Output Symbol" output of **DemodulateSubVI.vi** to the "Input Integer" input of the **Convert symbols to bits** subVI.
- ❏ The white arrow at the run button of the Front Panel of **DPSK_rx.vi** should not be broken now. If it is still broken, check your connection again.

- ❏ Click the run button of the Front Panel and your should find that the "trigger found?" button is in bright green colour and the BER indicator should show "0.0000" or a very low value.

- ❏ The Front Panel at your PC should look like Figure 10(b), where the Constellation Graph is a rotating one due to the phase error. However, the DQPSK demodulator can still detect the signal in the presence of phase errors as explained in Section 2.1.

- ❏ The Front Panel at the demonstrator's PC is illustrated in Figure 10(a), which shows the original QPSK Constellation Graph and its Eye Diagram.

You have built a receiver using the USRP and LabVIEW, where you are receiving data from the **Transmitter USRP** via a wireless interface. You transceiver uses DQPSK that dispenses with synchronisation and channel estimation, which makes it a relatively simple receiver to design and build. In the following, try to appreciate the importance of the system parameters when designing real communications systems.

Play with the parameters at the left hand side of the Front Panel and record your findings in terms of the Constellation Graph, Eye Diagram and BER value:

1. What happens when you change the default carrier frequency?

2. What are the effects of increasing and decreasing the roll-off factor from the default 0.5 value?

3. Change the other parameters and record the observations.