# ELEC50002 Communications - Laboratory

**Laboratory Handout 4 (Lab Session 7 + Lab Session 8)**

# Lab 4: Binary Phase Shift Keying (BPSK) Via USRP

In phase-shift keying (PSK) modulation, information is encoded on the phase of the transmitted carrier, rather than on its amplitude (ASK), or frequency (FSK). In binary phase-shift keying (BPSK) there are two phase values, $0^0$ or $180^0$, which means that an unmodified carrier is transmitted to represent one binary data value, while an inverted carrier is transmitted to represent the other. BPSK is optimum among binary modulation schemes in achieving the lowest average power for a given target bit error ratio (BER)[1].

## Exercise 1: BPSK Transmitter

The BPSK signal is given by:

$$Ag_{TX}(t)\cos(2\pi f_c t + \theta(t)).$$

In this equation, $A$ is a constant corresponding to the transmitted power level, $g_{TX}(t)$ is a fixed pulse shape, $f_c$ is the carrier frequency, and $\theta(t)$ takes the value of either $0^0$ or $180^0$ to carry the desired information. Note that, we can also write the equation as:

$$\pm Ag_{TX}(t)\cos(2\pi f_c t),$$

where the plus sign corresponds to $\theta = 0^0$, and the minus sign to $\theta = 180^0$. We will assume that a new pulse is transmitted every $T$ seconds, so that the symbol rate (symbols per second) is $1/T$. For a binary scheme such as BPSK, the bit rate is the same as the symbol rate. Since the pulse $g_{TX}[n]$ does not carry information, its shape can be chosen to satisfy other criteria. We desire a pulse shape that provides a rapid spectral roll-off to minimize inter-symbol interference, e.g. *'root-raised-cosine'* filter.

The steps needed to form a BPSK signal are:

1. Symbol Mapping: The input data arrives as a stream of bits. The **MT Generate Bits** VI produces an array of bits. For BPSK we will represent the bits in the following way:

| Bit Value | Phase ( $\theta$) | Symbol |
|---|---|---|
| 0 | 0 | -1 |
| 1 | 180 | +1 |

*Remark: The USRP requires a complex valued input. Make sure you convert the symbols to complex numbers* $(\pm 1 + j0)$ *before sending them as input to the USRP.*

2. Upsampling: As a first step towards pulse shaping, interpolation is required, therefore, we will place $L-1$ zeros after each symbol. This produces a sample interval of

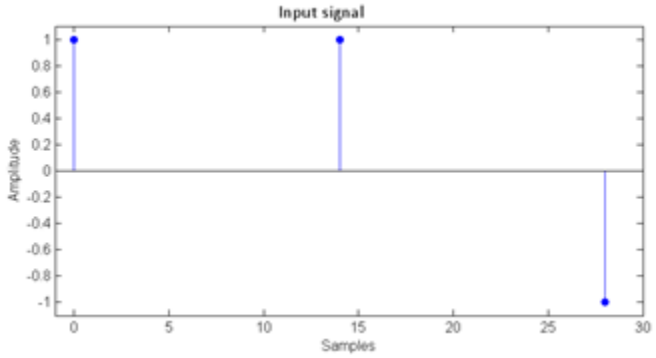
$$T_x = \frac{T}{L},$$

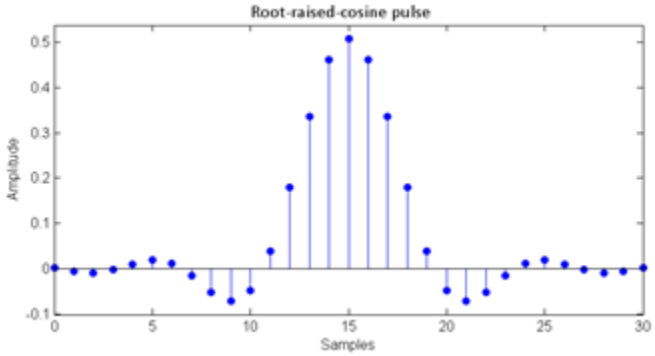
or, a sample rate of:

[1] BER represents the ratio of transmitted bits that have been incorrectly decoded at the receiver due to noise or interference. The goal is to achieve a BER as low as possible. There is a similar, but distinct quantity by the same acronym known as the Bit Error Rate, which measures the number of errors per unit time, rather than as a ratio to the total transmitted bits.

$$\frac{1}{T_x} = L\frac{1}{T}$$

3. Pulse Shaping: If the upsampled signal is applied to a filter whose impulse response $g_{TX}[n]$ is a root-raised-cosine pulse, then each symbol at the filter output will be represented by a root-raised-cosine pulse. This pulse shape has a very rapid spectral roll-off, so that consecutive transmitted symbols do not interfere with each other, i.e., avoid inter-symbol interference.




4. Modulation: The signal, of the form $(\pm 1 + j0)\ g_{TX}[n]$, can be sent directly to the USRP. The USRP transmitter will perform the DAC and the multiplication by the carrier $\cos(2\pi f_c t)$.

**Following the steps below, construct a BPSK transmitter:**
***Note*:** *Templates have been provided in the file **BPSK_Tx_Rx_Template.gvi**. Note that the file contains both transmitter (left) template and receiver (right) template.*

1. Add an ***MT Generate* Bits (PN Order - Fibonacci)** from the function palette to your template**.** This will create a pseudorandom sequence of bits for your BPSK system. You can create a control to specify the total number of bits. Note that by default, **MT Generate Bits** will produce the same sequence of bits every time you run the code. This is useful for debugging, but if you would like to generate a different sequence of bits every time, connect a **random number** to the '*seed in*' input terminal.

***Aligning the Received Bits***
*In this experiment, we will compare the received bits with the transmitted bits and see how many bits have been correctly received. For this comparison, the bit sequence at the output of the receiver should align with the transmitted bit sequence. Therefore, the receiver must recognize the beginning of the transmitted sequence. The **AddFrameHeader** sub-VI inserts a specific 26-bit sequence at the start of the transmission. At the receiver, the **FrameSync** sub-VI looks for this specific sequence and cuts off all the bits received before this frame header, as well as the frame header itself.*

2. Do the symbol mapping, which will convert the integers 0 and 1 from **MT Generate Bits** to doubles of -1 and +1, respectively. Then add the **AddFrameHeader** function provided.
3. Upsample the array of symbols using the **Upsample** function from the palette. Set the symbol rate $(1/T)$ to 10,000 symbols/s and the IQ rate $(1/T_x)$ to 200,000 Samples/s. Use these two inputs to calculate the upsampling factor $L$.
4. Use **MT Generate Filter Coefficients** function from the palette to generate the pulse-shaping filter. Set the input terminals as follows:
    a. "Modulation Type" to PSK,
    b. "Pulse Shaping Samples per Symbol" to your calculated value of $L$,
    c. "Pulse Shaping Filter" to "Root Raised Cos"

5. Connect the "pulse shaping filter coefficients" output terminal to the "Y" input terminal of a **Convolution** function. Connect the output from your **Upsample** function to the "X" input of the **Convolution** function.
6. Normalize the amplitude of your filtered message signal to a maximum absolute value of 1. The provided **Quick Scale 1D** function finds the maximum of the absolute value.
7. To observe the baseband waveform, connect the normalized signal to the "Y" input of a **Build Waveform** function. Given the IQ rate, obtain the corresponding value that should be set to the "dt" terminal of the **Build Waveform** function, and place an indicator to observe the waveform in the panel.
8. Next, convert the output of the normalized signal to complex values using the **Real and Imaginary to Complex** function.
9. To observe the baseband power spectrum, connect the normalized signal to the "Y" terminal of the **Cluster Properties** and the "dt" terminal should take the same value as the one obtained in step 7. The **Cluster Properties** function should already be on the template and is already connected to a **FFT Power Spectrum for 1 Chan (CDB)**, so to view the waveform simply place the graph "Baseband Power Spectrum" on your panel.

***Pilots and Channel Estimation***

*The "Insert Pilots" function inserts a specific sequence of symbols, which is known to the receiver and is used to estimate the channel transfer function. At the receiver, you will later insert the function "Channel Estimator" which will perform this task and adjust for the phase and amplitude error introduced by the channel.*

10. Connect the complex values to the **Insert Pilots** function provided, and add **AddFrameHeader (Complex)** function right after that.
11. Connect the output signal to the "data" node of the **niUSRP Write Tx Data (CDB)** function provided in the template (ensure that the input signal to the **Write Tx Data (CDB)** is a complex signal).
12. Assign the following values to the corresponding controls, and run the code.

| | |
|---|---|
| Carrier frequency | 400 MHz |
| IQ Rate | 200 kHz (Note: This sets the value of $1/T_x$) |
| Gain | 0 Db |
| Active Antenna | TX1 |
| Symbol rate | 10,000 symbols/s |
| Message Length | 1000 bits |

**Tasks:**

☞ Given the values set for $\frac{1}{T}$ and $\frac{1}{T_x}$, what is the corresponding value for the number of samples per symbol, L?

☞ Add the block diagram and the graphs from the front panel to your logbook (adjust the plots where necessary). Explain briefly your observations from the plots.

☞ From the spectrum plot, measure the "main lobe" bandwidth of the transmitted signal. Change the pulse shaping filter control to "none" to create rectangular pulses, and run the transmitter again. Compare the spectrum of the transmitted signal with the spectrum for root-raised-cosine pulses. Return the pulse shaping filter control setting to "Root Raised."

☞ How do the main lobe bandwidth and the spectral roll-off for root-raised cosine pulses compare with the same quantities when rectangular pulses are used?

## Exercise 2: BPSK Receiver

The BPSK signal that arrives at the receiver is given by

$$r(t) = \pm D g_{TX}(t) \cos(2\pi f_c t + \varphi),$$

where $D$ is a constant (usually much smaller than the constant $A$ in the transmitted signal), and the angle $\varphi$ represents the difference in phase between the transmitter and receiver carrier oscillators. If the receiver's carrier oscillator is set to the same frequency as the transmitter's carrier oscillator, the USRP receiver will do most of the work in demodulating the BPSK signal. The receiver's **Fetch Rx Data** function will provide a train of output samples, each given by

$$\tilde{r}[n] = \pm \frac{D}{2} g_{TX}[n] e^{j\varphi}$$

The sampling rate $1/T_z$ is set by the receiver's '*IQ rate*' parameter. This rate is set to provide $M$ samples every $T$ seconds, where $1/T$ is the symbol rate.

The steps needed to obtain the transmitted signals are:

1. Channel Estimation: This is required to remove phase ambiguity caused by the channel and the USRP oscillators. Since the Tx and Rx, are using free running oscillators, there will be some ambiguous phase offset that is highly dependent on the drift and skew of the oscillators themselves. The channel estimator attempts to correct this problem by reading the received pilots, which you inserted in the Tx signal, and performing least-square channel estimation to find the channel transfer function. The inverse of the channel transfer function is then applied to the Rx signal to remove the phase offset.

2. Matched Filtering: We will use a root-raised-cosine filter. This filter's impulse response $g_{RX}[n]$ is matched to the pulse shape $g_{TX}[n]$ of the transmitted signal. The matched filter is the optimal linear filter that maximizes the signal-to-noise ratio in the presence of additive stochastic noise.

3. Pulse Synchronization: The matched filter output is an analog baseband signal that must be sampled once per symbol time, i.e., once every *T* seconds. Because of filtering, propagation delays, and distortion caused by the communication channel, it is necessary to determine the optimum time to take these samples. A sub-VI called **PulseAlign** has been provided to align the baseband signal.

4. Sampling: The **Decimate** function will sample the aligned baseband waveform at index 0 and every *T* seconds thereafter.

5. Detection: Once the baseband waveform has been sampled, each sample must be examined to determine whether it represents a symbol 1 or -1.

6. Symbol Mapping: The detected symbol values must be converted to bits.

**Following the steps below, construct a BPSK receiver:**

1. Add the **FrameSync (Complex)** sub-VI after the output of the **Fetch Rx Data** function. Connect the output of **Fetch Rx Data** to the "Sampled Input" terminal of **FrameSync (Complex)** and leave its remaining inputs unconnected. This module will find the header indicating the beginning of the Tx signal, and leave out all received symbols prior to the header, including the header.
2. Use the **Channel Estimator** function provided to eliminate phase offset. Connect the input of the module to the "Aligned Samples" output of the **FrameSync (Complex)** function.

3. To implement the receiver's matched filter, use the **MT Generate Filter Coefficients** function, just as you did for the transmitter. Set the input terminals as follows:
    a. "Modulation Type" to PSK,
    b. "pulse shaping filter" to "Root Raised Cos"
    c. "matched samples per symbol" to "$M$", calculated from the "IQ rate" $(1/T_z)$ and the symbol rate $(1/T)$ obtained from the front-panel control.
4. Extract the real part of the output of the **Channel Estimator** by using the **Complex to Real and Imaginary** function from the palette, and convolve the real part of the signal with the "matched filter coefficients" using the **Convolution** function.
5. To observe the baseband output power spectrum, connect the output of the convolution to the "Y" terminal of the **Cluster Properties,** which is already be on the template, and connect the "dt" terminal to its corresponding value.
6. Place the **PulseAlign (Real)** sub-VI on your block diagram and wire the baseband output waveform (the output of the convolution) to the "input signal" terminal, and wire the $M$ samples/symbol to the "receiver sampling factor" terminal.
7. Once the baseband waveform is aligned, it can be sampled with the **Decimate (single shot)** function. Note that the "decimating factor" is the value $M$.
8. Place a **FrameSync (Real)** sub-VI immediately after the **Decimate** function, using the same inputs and outputs as in steps 1 and 2.
9. Next, determine if each element of the output of the "Aligned Samples" is a 1 or a 0 using a **For Loop**.
   *Hint: You can do this by using the **Greater Than 0?** function inside a **For Loop**. Then use a **Boolean to Integer** function to convert this to an integer array.*
10. Connect the output array to the 'array' input terminal of an **Array Subset** function (set the 'index' input terminal to zero, and the 'length' input terminal to the length specified by the 'message length' control). Display the output of Array Subset function as 'Output bits' on the receiver front panel.
11. Measurement of the bit error ratio (BER) can be automated using the **MT Calculate BER** function from the palette. From the Configure ribbon of the **MT Calculate BER** function, choose "PN Fibonacci." Set the "BER trigger threshold" to 0.4. Connect indicators to the "BER" and "trigger found?" outputs. Connect the "Output bits" to the "input bit stream" of the **MT Calculate BER** function. When you run the program, "trigger found?" will be true whenever the measured BER is below the "BER trigger threshold". If "trigger found" remains false during the execution of the program, run it again until it becomes true. (Alternatively, you can code your own BER calculator by using necessary functions from the palette) You should particularly code your own BER calculator if you have used a random seed with the **MT Generate Bits**).
12. Assign the following values to the corresponding controls.

| | |
|---|---|
| Carrier frequency | 400 MHz |
| IQ Rate | 200 kHz (Note: This sets the value of $1/T_z$) |
| Gain | 0 dB |
| Active Antenna | RX2 |
| Symbol rate | 10,000 symbols/s |
| Message Length | 1000 bits |

**Tasks:**

☞ Run the transmitter and receiver several times. What value(s) of BER do you observe?

☞ Set the Rx and Tx gain as shown in the table below, and run each setting 5 times. Record the BER value you observe at each run, and the average BER for each setting.

| Tx Gain (dB) | Rx Gain (dB) |
|---|---|
| 0 | 0 |
| -35 | -15 |
| -37 | -15 |
| -40 | -15 |

## Exercise 3: Differential Phase Shift Keying (DPSK)

In DPSK, the transmitter sends the difference between two adjacent bits and not the bits themselves. The table below shows how the difference is obtained for possible pairs of symbols. You can see from the table that the output of the encoded sequence is obtained by $b_n = b_{n-1} \times a_n$.

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| Information symbols $\{a_n\}$ | | 1 | -1 | -1 | 1 | -1 | -1 | 1 | 1 |
| $\{b_n\}$ | | 1 | 1 | -1 | 1 | 1 | -1 | 1 | 1 |
| Differentially encoded sequence $\{b_n\}$ | 1 | 1 | -1 | 1 | 1 | -1 | 1 | 1 | 1 |
| Output of lowpass filter (polarity) | | + | - | - | + | - | - | + | + |
| Decision | | 1 | -1 | -1 | 1 | -1 | -1 | 1 | 1 |

**Note:** The symbol 1 at the beginning of the encoded sequence is the reference symbol

Following the steps below, you will create a differential phase shift keying (DPSK) communication system by adding a differential encoder to your BPSK transmitter, and a differential decoder to your receiver.

1. Make new copies of your BPSK transmitter and receiver, and use the new copies to create your DPSK system.
2. Design a differential encoder and add it to the BPSK transmitter. The encoder should be added after the **AddFrameHeader** function after the symbol mapping (the frame header also gets differentially encoded).
3. Remove the "**Insert Pilots**" and "**AddFrameHeader(Complex)**" functions that were placed before the "**niUSRP Write Tx Data**" module and simply convert the output of the "**Quick Scale 1D**" to complex values and connect the complex signal to the input data node of the "**niUSRP Write Tx Data**" module.
4. At the receiver, remove the "**Channel Estimator**" and the "**Frame Sync (Complex)**" functions that were placed after the "**niUSRP Fetch Rx Data**" module, and connect the received data directly to the **Convolution** function.
   *Remark: Your received data is complex, therefore make sure you use the right functions, e.g.,* ***Convolution (CDB)****,* ***PulseAlign (Complex)*** *and* ***FrameSync (Complex)*** *subsequently.*
5. Design a differential decoder and add it to the BPSK receiver. Place the differential decoder immediately after the **Decimate** function, which is the receiver's sampler, and before **FrameSync (Complex)**.
   *Remark: Since the received data are complex-valued at this point, design your decoder to form the product of the current sample and the complex conjugate of the previous sample. Then take the real part of the result.*

**Tasks:**

☞ Add the block diagram of your DPSK system to your logbook

☞ Run your code several times using the initial configuration used for the BPSK system. Explain your observation(s) on the BER performance of the system.

☞ Compare the performance of BPSK and DPSK, and explain your findings.

## Exercise 4: Error Correction Coding (Bonus)

In this exercise, we will perform forward error correction (FEC) by adding redundancy to the transmitted information. In FEC, the transmitter sends each data bit 3 times (this makes a *triplet*). Due to the noise in the channel, the receiver might receive 8 versions of triplets, and decode the corresponding bit as shown in the table below (*majority vote decoder*).

| Triplet received | Bit decoded |
|---|---|
| 000 | 0 |
| 001 | 0 |
| 010 | 0 |
| 011 | 1 |
| 100 | 0 |
| 101 | 1 |
| 110 | 1 |
| 111 | 1 |

This implies that an error in any one of the three samples is corrected by “majority vote”. To implement FEC, we will make the transmitter send each bit of the message three times consecutively. The receiver in turn will receive and demodulate these bits (plus the noise introduced). Afterwards, it will decode the bits as shown in the table above.

**Following the steps below, construct a BPSK modulation system with error correction coding:**

1. Duplicate the **BPSK.gvi** you constructed in the previous exercise and rename it as **BPSK_FEC.gvi**.
2. Design an FEC Encoder by tripling each message bit from ***MT Generate* Bits (PN Order - Fibonacci)** consecutively. For example, if the message bits from ***MT Generate* Bits (PN Order - Fibonacci)** is 011, convert it to 000111111.
3. At the receiver, modify the “length” node of the **Array Subset** function to 3x the message length.
4. Design an FEC Decoder to decode the bits from the output terminal of the **Array Subset** function as in the table above.
5. Connect the output of your FEC Decoder to the **MT Calculate BER** function and/or to the BER calculator that you coded by yourself.

**Tasks:**

☞ Using the same system configurations as with the BPSK system without FEC, set the Rx and Tx gain to the following settings, and run each setting 5 times to obtain an average for the BER:

| Tx Gain (dB) | Rx Gain (dB) |
|---|---|
| 0 | 0 |
| -35 | -15 |
| -37 | -15 |
| -40 | -15 |

☞ Compare your BER results from Exercise 2 with what you obtained here. Has the FEC improved the BER of the system?

☞ What are the trade-offs involved in the error correction coding system? What are the advantages and disadvantages of the system?

## Appendix: NI USRP Hardware Diagram

The NI USRP connects to a host PC creating a software defined radio. Incoming signals at the SMA connector inputs are mixed down using a direct-conversion receiver to baseband I/Q components, which are sampled by an analog-to-digital converter (ADC). The digitized I/Q data follows parallel paths through a digital down-conversion (DDC) process that mixes, filters, and decimates the input signal to a user-specified rate. The down-converted
samples are passed to the host computer.

For transmission, baseband I/Q signal samples are synthesized by the host computer and fed to the USRP at a specified sample rate over Ethernet, USB or PCI express. The USRP hardware interpolates the incoming signal to a higher sampling rate using a digital up-conversion (DUC) process and then converts the signal to analog with a digital-to-analog converter (DAC). The resulting analog signal is then mixed up to the specified carrier frequency.

The steps mentioned above can be seen in the following figure:

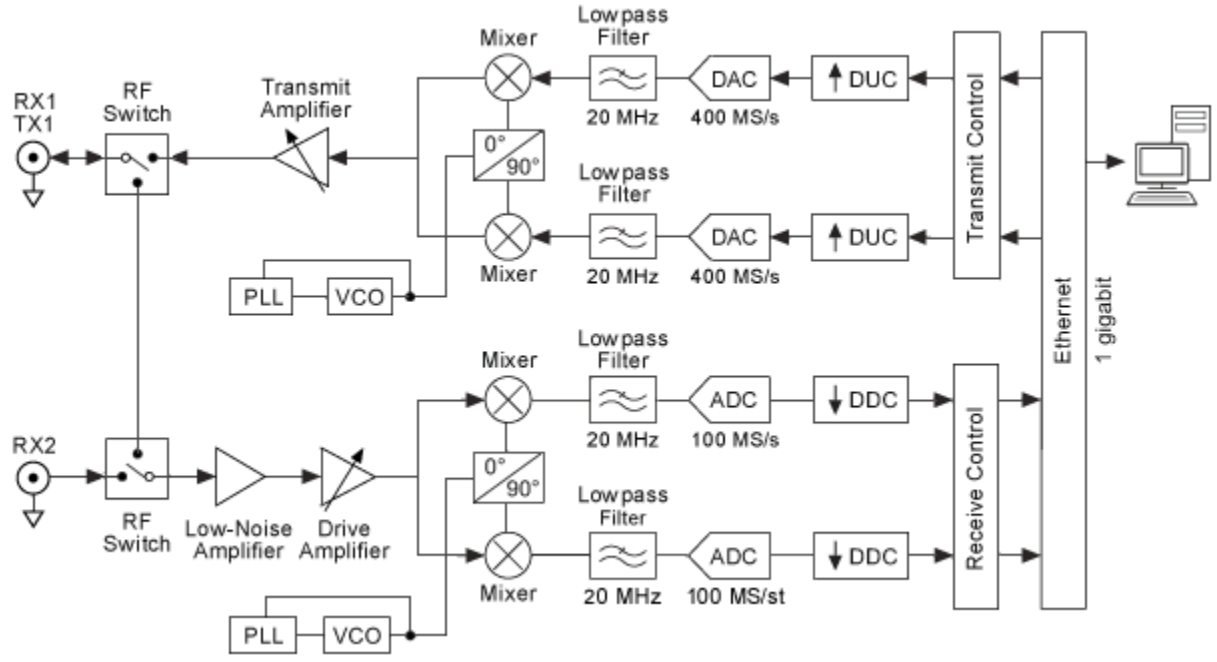



**Figure A1: USRP's internal circuit.**