Analog and Digital Communications

2nd Year Electronic Laboratory

Imperial College London

Version 2.00

Prepared by:

S. Somuyiwa – D. Kurka – T. Tze-Yang – S. Sreekumar - M. Mohammadi Amiri – J. Puyol-Roig

C. Leung - D. Gunduz

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 rate.

Exercise 1: BPSK Transmitter

A BPSK signal is a train of pulses, each of the form

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

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 θ takes the value of either 0^0 or 180^0 to carry the desired information. Note that, we can also write the equation as:

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

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 and minimizes 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 (θ)	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.*

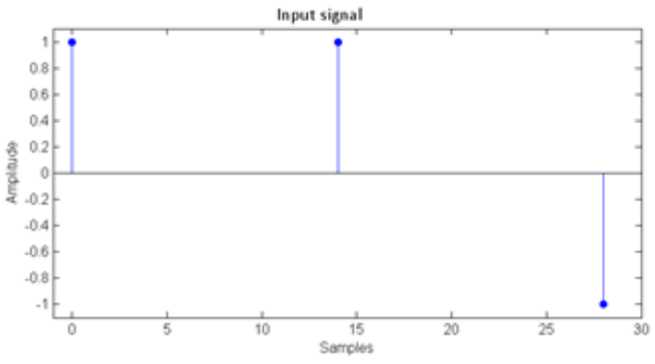
2. Upsampling: As a first step towards replacing symbols with pulses, we will place $L-1$ zeros after each symbol. This produces a sample interval of

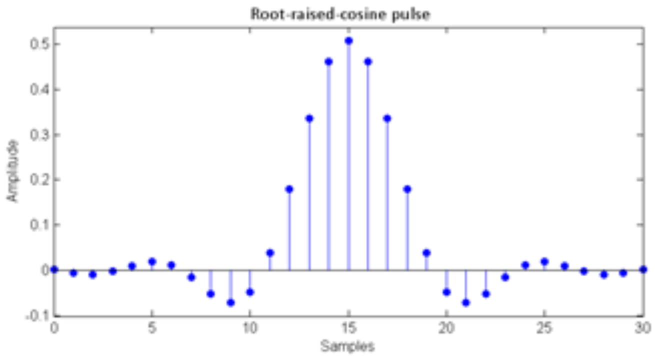
$$T_x = \frac{T}{L}$$

or a sample rate of:

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

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 the transmitted signal will not cause interference to signals at nearby carrier frequencies.





4. Modulation: The signal, consisting of a train of pulses of the form $(\underline{+}1 + j0)\, g_{TX}[n]$, 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 (upper) template and receiver (lower) template.*

1. Add an ***MT Generate*** **Bits (Fibonacci, PN Order)** 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 bit sequence sent from the transmitter. 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. Afterwards, you can convert the symbols to complex numbers.
3. Upsample the array of symbols using the **Upsample** function from the palette: **Analysis → Signal Processing → Signal Operation**. 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 Cosine"

5. Connect the “pulse shaping filter coefficients” output terminal to the “Y” input terminal of a **convolution** function (from the Analysis → Signal Processing → Signal Operation palette). 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. To ensure that your scaled message remains complex-valued (if you have already converted the symbols to complex numbers before this stage), use a separate division function to do the actual scaling.

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.

7. Next, connect the output of the normalized signal to the **Insert Pilots** function provided to you and add another **AddFrameHeader** function right after that.
8. Connect the normalised signal to a **Build Waveform** function, and create an indicator for the output waveform. Note that the Input to the ‘dt’ of the **Build Waveform** function is the reciprocal of the actual IQ rate. Also send the normalised signal to the **Write Tx Data (CDB)** function provided in the template (ensure that the input signal to the **Write Tx Data (CDB)** is a complex signal).
9. To observe the spectrum of the transmitted signal, connect the baseband waveform to the **FFT Power Spectrum for 1 Chan (CDB)** sub-VI provided, and create an indicator for the Power Spectrum graph.
10. 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
Pulse shaping filter	Root Raised

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 has the form of a train of pulses, each 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 φ 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 pulses, 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. <u>Channel Estimation</u>: This is required to remove phase ambiguity caused by the channel and the USRP oscillators. Because the Tx and Rx channels are both 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 then performing the LSE channel estimation to find the channel transfer function. The channel transfer function is then inverted on the Rx signal to remove phase offset.

2. <u>Matched Filtering</u>: We will use a root-raised-cosine receiver filter. This filter's impulse response $g_{RX}[n]$ is matched to the pulse shape $g_{TX}[n]$ of the transmitted pulses. The matched filter gives optimum performance in the presence of additive white Gaussian noise.

3. <u>Pulse Synchronization</u>: 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. <u>Sampling</u>: The **Decimate** function will sample the aligned baseband waveform at index 0 and every *T* seconds thereafter.

5. <u>Detection</u>: Once the baseband waveform has been sampled, each sample must be examined to determine whether it represents a symbol of value 1 or a 0.

6. <u>Symbol Mapping</u>: The detected symbol values must be converted to bits. For binary PSK, this step is easily included in the detection step.

Following the steps below, construct a BPSK receiver:

1. Calculate the '*number of samples*' for the **Fetch Rx Data**, using the message length and the symbol rate used for the transmitter. *(Hint: the 'number of samples' is equivalent to the 'number of samples' going into the **Write Tx Data (CDB)**).* Double the number of samples in a frame so that the receiver will fetch two frames of data. Since the receiver's starting point is random, this ensures that there will be one complete frame of received data in the block of samples fetched.
2. Connect the **Frame Sync** function at the output of the **Fetch Rx Data** function. This will find the header indicating the beginning of the Tx signal and cut off all received symbols prior to the header, including the header.

3. Use the **Channel Estimator** function provided to eliminate phase offset. Connect the module after the **Frame Sync** function.
4. 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 Cosine"
    c. "matched samples per symbol" to "M", calculated from the "actual IQ rate" $(1/T_z)$ and the symbol rate $(1/T)$ obtained from the front-panel control.
5. Connect the "*matched filter coefficients*" output to the "Y" input of a **Convolution** function. "X" value is the output of the **Phasesync**.
6. Place the **PulseAlign** sub-VI on your block diagram and wire the baseband output waveform 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 M.
8. Place **FrameSync** sub-VI immediately after the **Decimate** function. Connect the output of the **Decimate** function to the "Sampled Input" terminal of **FrameSync** sub-VI. Leave the remaining inputs of **FrameSync** sub-VI open.
9. To determine whether each received sample is more likely to represent a 1 or a 0, the sample must be compared with a threshold. Since the message is a polar signal, compare the *'Aligned Samples'* output terminal of **FrameSync** to zero. The result of this comparison is the receiver's digital output. The output of the comparison will be a Boolean array.
10. Use a **Boolean to Integer** function to convert this to an integer array. Connect this 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 rate (BER) can be automated using the **MT Calculate BER After Trigger** function from the Modulation Toolkit (Analysis → Communications → Digital → Measurements subpalette). From the Configure ribbon, 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 After Trigger** function. When you run the program, "trigger found?" will be true whenever the measured BER is below the "BER trigger threshold".
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
Pulse shaping filter	Root Raised

Tasks:

☞ Run the transmitter and receiver several times. If you built them separately, run the transmitter first, and ensure that it is transmitting before you run the receiver. What value(s) of BER do you observe?

☞ It is shown in the Background section above that the received baseband signal $\tilde{r}[n]$ includes a factor $e^{j\varphi}$, where φ is any phase difference that may exist between the transmitter and receiver carrier oscillators. Explain what would happen if you omitted the phase synchronization step in the receiver. Specifically, what would be the receiver output if φ just happened to take the value $\pi/2$?

☞ What would the receiver output be if we used an envelope detector for demodulation?

☞ Change the values assigned to the Gain of the transmitter to -20dB, and of the receiver to -10dB. Run the code. What values of BER do you observe? Explain your observation(s).

Exercise 3: Error Correction Coding

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.

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 3 separate blocks of bits, i.e., 3 different arrays of same bit elements, instead of 1 block of bits. The receiver in turn will receive and demodulate these 3 blocks of bits (plus the noise introduced) separately. Afterwards, it will decode the signal and recover a single block of bits.

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

1. Perform necessary actions to transform your transmitter and receiver system (It is advisable for you to retain and keep the original copies of your codes) such that your transmitter sends 3 separate blocks (3 different arrays) of same length and same bit-elements, and your receiver system receives these 3 blocks of bits separately.
2. After these 3 blocks of bits have been received, implement a decoder to retrieve a single bit by “majority vote”. To do this, you can use the **MathScript Node** or **C Node** function from the function palette. This allows you to write MathScript or C programming language within the Labview environment.
3. Calculate the BER of your decoded bits. You will not be able to use the **MT Calculate BER After Trigger** function here. To calculate the BER, compare the decoded bit array, element by element, with the transmitted bit array, which is generated by the **MT Generate Bits (Fibonacci, PN Order)** function. You can use the **MathScript Node** or **C Node** function to perform this comparison.

Tasks:

☞ Using the same system configurations as with the BPSK system without error correction, but with transmitter gain set to -20dB and receiver gain set to -10dB, run your code 10 different times and record the average BER obtained. Compare the average BER with that obtained for the same configuration of the BPSK system without error correction.

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

Exercise 4: 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 (the frame header also gets differentially encoded).
3. In the receiver, remove the phase synchronizer, and connect the received data directly to the **Convolution** function. Your received data is complex, therefore make sure you use the right functions, e.g., **Convolution (CDB)** instead of Convolution **(DBL)** and **PulseAlign(Complex)** instead of **PulseAlign(real)**.
4. 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(real)**. 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.

Appendix A: 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:

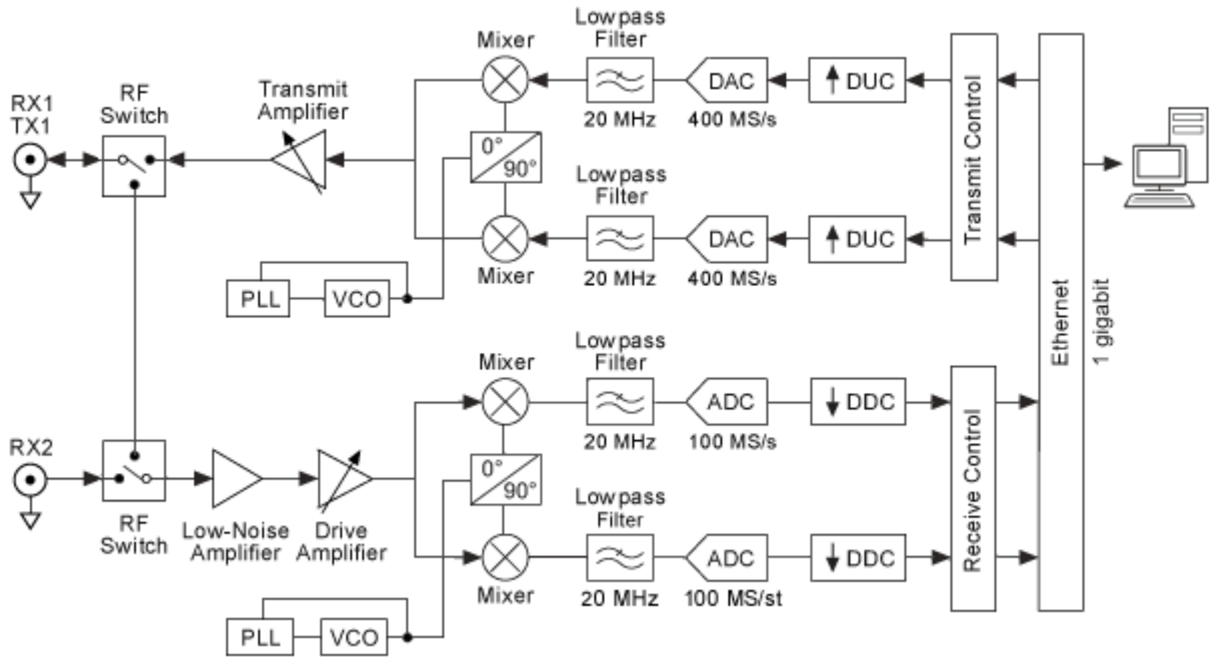


Figure A1: USRP's internal circuit.