# ECE 463 Lab 4: Frame Synchronization and Non-Coherent Modulation

## 1. Introduction

At this point, you have a working transmitter and receiver, but to confirm that they are working together correctly as a system, you need to compare the received bits with the transmitted bits and verify that the bit patterns are the same. Recall that you were able to locate the short bit sequence you transmitted (i.e. 1011100) in Lab 3. In practice for a much longer bit sequence, it is necessary that the receiver recognizes the beginning of the transmitted sequence. Therefore, the transmitter inserts a training sequence (header, preamble), which is a fixed data pattern, at the start of each data frame to mark the start of valid data. The receiver searches for this training sequence in each data frame and achieves frame synchronization when the correlation between the input data and the fixed pattern is high.

Digital modulation/demodulation types are classified into two groups: coherent and non-coherent modulation. Coherent modulation/demodulation systems need the carrier phase information to detect and decide the transmitted data. In contrast, non-coherent systems do not need such phase information. In this lab, we will introduce one of the non-coherent modulations, Differential Binary Phase Shift Keying (DBPSK).

### 1.1. Contents



### 1.2. Report

Submit the answers, figures and the discussions on all the questions. The report is due as a hard copy at the beginning of the next lab.

# 2. Frame Synchronization

In this section, you will implement the following subVIs.

- AddTraining.gvi: Inserts training sequence into a frame
- FrameSync.gvi: Detects the training sequence and synchronizes the frame
- sim_framesync.gvi: Simulates the above subVIs

## 2.1. Adding training symbols

- Design "AddTraining.gvi" subVI that inserts the training sequence of symbols into a frame at the transmitter with the following input and output.

| | Terminal name | Type | Description |
|---|---|---|---|
| Input | *Original frame* | Double array | Original frame of symbols |
| | *Training sequence* | Double array | |
| Output | *Frame with training sequence* | Double array | Frame with training sequence at the beginning |

- Use "Insert into Array" block to insert the training sequence at the beginning of the original frame (The "index" port of the "Insert into Array" should be "0").

## 2.2. Frame detection and synchronization

Design "FrameSync.gvi" subVI that locates the training sequence of symbols on the receiver side in order to align and synchronize the frames.

| | Terminal name | Type | Description |
|---|---|---|---|
| Input | *Unsynced frame* | Double array | Received frame before synchronization |
| | *Training sequence* | Double array | |
| | *Threshold* | Double | The minimum value to be detected as peaks |
| Output | *Synced frame* | Double array | Synchronized frame |
| | *Cross-correlation result* | Double array | Cross-correlation result between the received frame and the training sequence |
| | *Peak index* | Integer array | Index of the peaks detected in the cross-correlation result |

- Use "Signal correlation" block and select the "Cross-Correlation" option to calculate the cross-correlation of the "*Unsynced frame*" and the training sequence.

- Use "Threshold Detector" to find the peaks of the cross-correlation result and their indices. (Hint: "Threshold Detector" finds the peaks of the input sequence that exceed the threshold. It returns the indices of the peaks at the "indices" port and the number of peaks at the "count" port.)

- Remove the training sequence from the frame and pass on the synced frame to the corresponding output port. (Hint: Use "Index Array" to get the index of the first peak of the correlation result. Use "Array Subset" to get the desired "*Synced frame*" from the "*Unsynced frame".*)

## 2.3. Simulation

Create another subVI named "sim_framesync.gvi" to simulate and verify that "AddTraining" and "FrameSync" work correctly.

- Generate a sequence of data bits with the "MT Generate Bits" block. For example, you can configure the block with "Fibonacci" as the PN order, 50 as the total bits, and 5 as the PN sequence order.
- Map the bits to the symbols, e.g. 1 mapped to "1" and 0 mapped to "-1". Create an indicator to monitor the sequence of symbols created. It will be used as the original frame without the training sequence.
- Create a array constant with the 13-bit Barker code as the training symbol sequence. The 13-bit Barker code is **+1 +1 +1 +1 +1 −1 −1 +1 +1 −1 +1 −1 +1**.
- Use your "AddTraining" block to insert the training sequence at the beginning of the original frame, and generate the frame with the training sequence.
- Use your "FrameSync" to locate the training sequence in the frame generated from the last step. To do this, you need to observe the correlation results and choose an appropriate threshold value based on the cross-correlation to find desired peaks.

## 2.4. Questions

2.4.1. Where is the desired peak of the correlation? What's the relationship between the peak and the beginning of the original frame? Is your "FrameSync.gvi" subVI generating the synchronized frame?

2.4.2. A good training sequence should give a significant peak in the correlation so that it can be easily detected. Therefore, we may want to increase the length of the training sequence. Since the maximum length of the Barker code is 13, we concatenate two or more Barker codes. Create a 26-bit training sequence by duplicating the 13-bit one. (Hint: Use "Build Array" and check "Concatenate Inputs" box to concatenate two Barker codes.) How would you change the threshold value? Where is the peak of the correlation now? Where does the original frame start?

2.4.3. What is the advantage and disadvantage of longer training sequences?

# 3. Differential Binary Phase Shift Keying (DBPSK)

## Transmitter

Start with "TxBPSK.gvi" **from lab 3.** Copy it to the new project folder and rename it as "TxDBPSK.gvi". In this section, we will build a transmitter that performs Differential Binary Phase Shift Keying (DBPSK) modulation. The data is conveyed in the phase of the carrier signal just as BPSK. DBPSK is different from BPSK that its transmitted symbol depends not only on the current bit but also on the previous one.

Set the default parameter values in the panel:

- Carrier frequency: 1G
- IQ rate: 400k
- Gain: 0
- Active antenna: TX1
- Message length: 1000
- Symbol rate: 10k
- Enabled Channels: 0

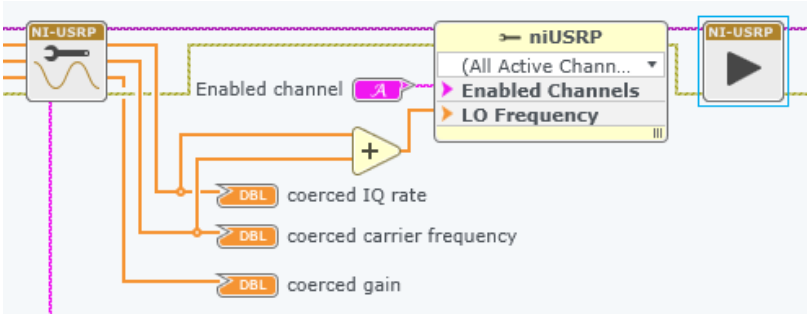
1. (Training sequence) Create a training sequence constant and use the "AddTraining" block from section 2 to insert a training sequence to the symbol sequence. (Hint: Differentiate symbols from bits and samples)

2. (Differential encoding) Implement the differential encoding on the frames with the training sequence (Hint: Referring to the lecture slides and the example table below).

| Input symbol | -1 | -1 | 1 | 1 |
|---|---|---|---|---|
| Previous symbol | -1 | 1 | -1 | 1 |
| Current symbol | 1 | -1 | -1 | 1 |

3. Reuse the rest parts of the transmitter chain (e.g. pulse-shaping filters, convolution, etc.).

4. Add the "LO Frequency" configuration to the niUSRP properties blocks in **both transmitter and receiver**.



## Receiver

Start with "RxBPSK.gvi" from lab 3. Rename the file as "RxDBPSK.gvi".

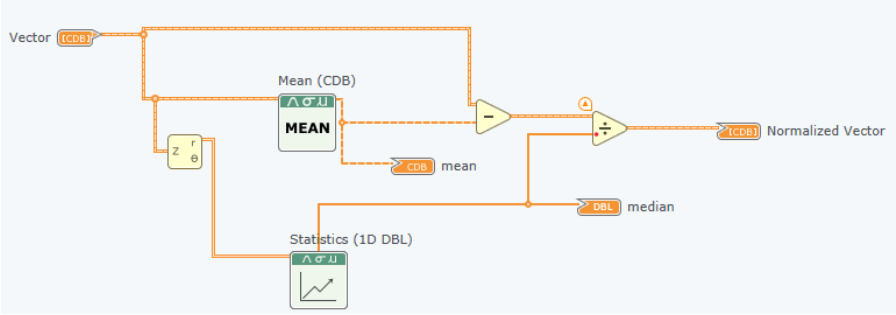
Set the default parameter values in the panel:

- Carrier frequency: 1G
- IQ rate: 400k
- Gain: 0
- Active antenna: RX2
- Message length: 1000
- Symbol rate: 10k

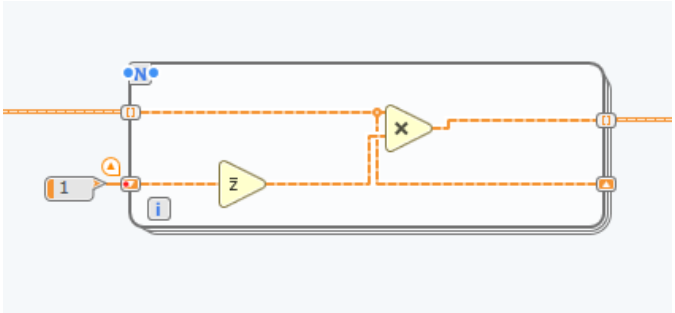
1. Calculate the number of samples to capture **two frames**, and use it as the number of samples for the "niUSRP Fetch Rx Data" block.

2. Reuse the matched filter, convolution, pulse alignment, and decimation chain.

3. We will build another type of normalization block called "**norm1D_median.gvi**". Instead of normalizing by the maximum value (norm1D from Lab 3), we will normalize the vector by its median. This is a more robust way to normalize when the vector contains outliers. Create a subVI as the following diagram and use it to normalize the received symbols.




4. (Differential decoding) Implement the following diagram to decode the normalized symbols.




5. Take the real value of the decoded symbols. Use the "FrameSync" block and the right training sequence to synchronize the frame. We suggest to use 18 as the threshold value.

6. (Symbols to Bits) Map symbols to bits in the same way as BPSK.

7. (BER) Let's compute the bit error ratio and verify that the decoded bits are correct, by comparing with the bit sequence transmitted: Create a PN sequence with the same configuration and the parameters as that in the transmitter (i.e. Fibonacci, PN sequence order, total bits, seed in, etc.). Compare the decoded bit array with the reference using a for loop. Count the number of errors (Hint: Use "Not Equal?", "Boolean to Integer", "Add" and a shift register.). Divide the number errors by the message length to get the BER.

8. (Constellation) Place a graph in the panel. Change its x-axis name as I and its y-axis as Q (click the label to change). Set the "Autoscale" to none and set min/max of the scale as -1.5 and 1.5. Display the decoded symbols on the graph.

## 3.1. Questions

3.1.1. Report the BER and the constellation of DBPSK.

3.1.2. Repeat Q3.1.1 for BPSK (Tip: Use Case structure to switch between DBPSK and BPSK. Simply bypass the differential encoding/decoding part for BPSK). Compare and explain the difference between their constellation.

3.1.3. Understand and explain the how the differential decoding diagram (step 4 in the receiver) works.

3.1.4. (Optional) Reduce the transmitted signal power by multiplying the samples with some small constants (e.g. 0.01) and report changes in the BER.