ECE 463 Lab 10: IoT – RFID & LoRa

1. Introduction

Passive RFID tags do not use a radio transmitter. Instead, they respond by modulating the reflection coefficients of the tag antenna, thereby backscattering an information signal to the RFID reader. In this lab, we will capture the transaction between the RFID reader and tag.

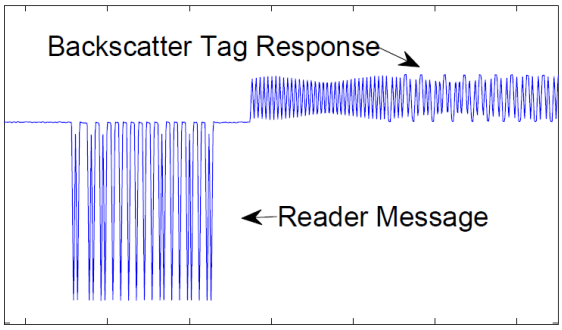
LoRa (Long Range) is a spread spectrum modulation technique derived from chirp spread spectrum (CSS). LoRa devices and wireless radio frequency technology is a long-range, low power wireless platform that enables smart IoT applications. In this lab, we will implement LoRa transmitter & receiver and demonstrate them in wireless.

1.1. Contents

1.2. Report

Submit the answers, figures and the discussions on all the questions. The report is due as a soft copy (.pdf or .docx) by email to your Lab Instructor (Thomas Moon or Jayden Guan) by Friday Dec. 14th at 6pm.

2. RFID

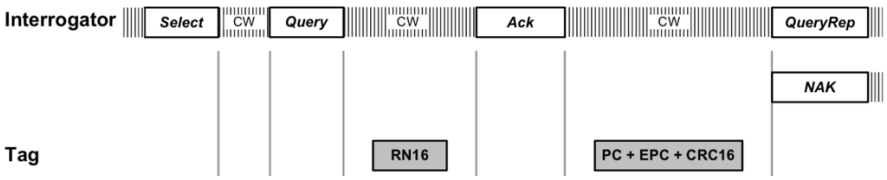


In this section, we will monitor the RFID reader message and backscatter tag response. The instructor will place an RFID reader and a tag near your USRP. During the RFID transactions, capture the received signals using your USRP (use a RF antenna at Rx) and find the backscatter tag response as shown in the figure. The RFID reader will send OOK modulated signal at 918.250MHz carrier. To rectify the received OOK signal, use **Complex to Polar** block to take its magnitude (You do not need to decode or match-filter the signal). Use the following configuration for the receiver:

- IQ rate: 500kHz
- Carrier frequency: 918.200MHz
- Number of samples: 5000

2.1. Question

2.1.1. The tag will respond two packets as shown in the following figure: one short packet to reply to the query and one long packet including the data. The instructor will configure the tag-to-reader encoding as **Miller code 2 and 4**. Find the two packets and measure the length of them for each case. Explain the results.



2.1.2. Measure the signal power of the reader and the tag. What is the difference of the power in dB?

3. LoRa: Tx

In this section, we will implement the blocks required in the LoRa transmitter. The following subVI's will be implemented:

- **generate_UPchirp.gvi**
- **generate_DOWNchirp.gvi**
- **spectrogram.gvi**
- **CSS_modulate.gvi**
- **CSS_Symbol_encoder.gvi**

3.1. Generate Up/Down Chirp Spread Spectrum

CSS uses a linear chirp signal that continuously sweeps frequency with time in linear. The discretized up-chirp signal is defined as

- $s[n] = \exp\left(j2\pi\frac{\hat{\alpha}}{2}n^2 + j2\pi\hat{f_0}n\right),\ n = 0, \dots, N-1$
- $chirp\ slope\colon\ \hat{\alpha} = \frac{B}{Nf_s}$
- $starting\ frequency\colon\ \hat{f_0} = -\frac{B}{2f_s}$
- $spreading\ factor\colon\ SF = \log_2 N$

where $\boldsymbol{B}$ is the bandwidth of the chirp signal, $\boldsymbol{f_s}$ is the sampling frequency, and $\boldsymbol{N}$ is the number of samples per chirp. Note that the frequency linearly increases with time.
Implement a block that generates up-chirp CSS.

generate_UPchirp.gvi	Terminal name	Type	Description
Input	*BW*	Double	Bandwidth
	IQ rate	Double	Sampling frequency
	SF	Unsigned integer	Spreading factor
Output	*Output chirp*	Complex double array (1D)	

- Find **Pattern Generator** block and choose **Ramp pattern by Samples** to create a sequence from 0 to *N-1*.

- There are a couple of different ways to create a complex exponential signal. One way is to use **cosin** and **sin** blocks for its real and imaginary part, and then use **Real and Imaginary to Complex** block to combine them to complex signal.

- Implement **generate_DOWNchirp.gvi** that generates a chirp signal whose frequency linearly decreases over time. This can be simply done by taking **Complex Conjugate** of the output of **generate_UPchirp block.**

3.2. Spectrogram

A spectrogram is a visual representation of the spectrum of signal over time. We will implement a simple spectrogram block that computes the spectrum by using a sliding window (no overlapping) to divide the signal into several chunks of data.

spectrogram.gvi	Terminal name	Type	Description
Input	*Input signal*	Complex double array (1D)	Time-domain signal
	Window size	Unsigned integer	Length of window size
Output	*Output Spectrogram*	Double array (2D)	

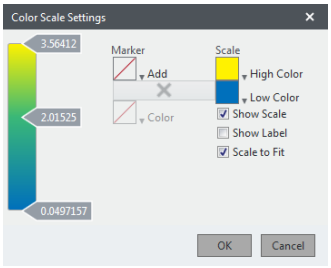
To compute the output of the spectrogram, this block completes the following process.

1. Break the input signal into chunks by the window size. Use ***S2P.gvi*** block from the previous lab to perform this process.

2. Use a for loop and auto indexing to perform FFT by row-wise. **Check shift? option** in the FFT block to shift the DC component at the center of the FFT output.

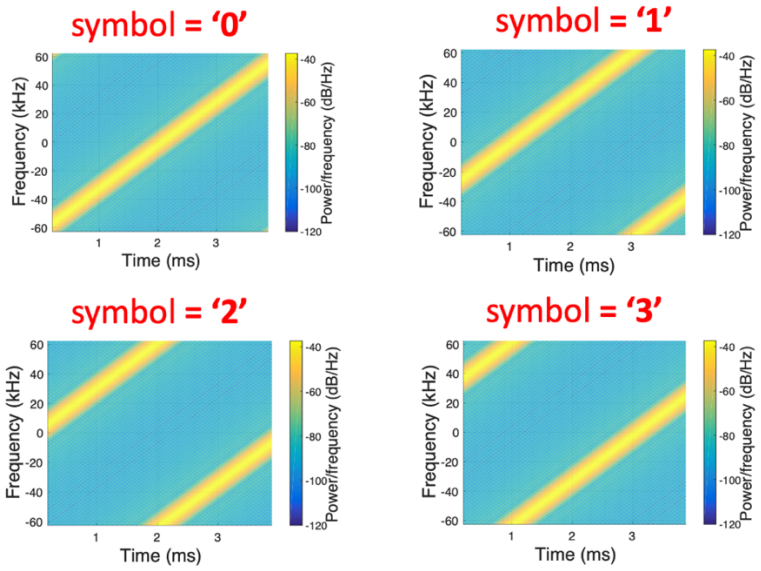
3. Use auto-indexing to make the output 2D array. Use **Complex to Polar** to compute the absolute value of the result.

4. Use **Intensity Graph** in front panel to display the output (Do not use Graph). Click the intensity graph and find **Color Scale** button in Item panel. Change the color scheme for better visualization. A recommended color scheme is shown in the following figure (ignore the numbers).



3.3. CSS Modulation

Modulating CSS can be done by circular-shifting the base chirp (symbol 0). The following figure shows an example of encoding 2 bits per symbol (SF=2). The symbol always spans from 0 to *N-1*. Note that the base chirp signal is *advanced* by the value of the symbols.



CSS_modulate.gvi	Terminal name	Type	Description
Input	*BW*	Double	Bandwidth
	IQ rate	Double	Sampling frequency
	SF	Unsigned integer	Spread factor
	symbols	Unsigned integer array (1D)	Input stream of symbols
Output	*Modulated CSS*	Complex double array (2D)	

- Use **generate_Upchirp.gvi** to generate the base signal (symbol 0). Use a for loop, auto-indexing and a **Rotate 1D Array** block to perform circular-shift for each symbol. Note that **Rotate 1D Array** block *delays* the input array when n is *positive*. Therefore, we need to make the symbols negative before use them for circular-shift. Use **Negate** and **To Signed 32-bit Integer** blocks to reverse the sign of the symbols.

- The n-th row of output array corresponds to the modulated CSS signal by the n-th symbol.

3.4. Bits-to-CSS symbols

In CSS, the input bit stream maps to the symbols from 0 to N-1. Modify **Symbol_encoder.gvi** from Lab8 to perform this. Instead of mapping the bits to the input symbol map, use the index number for the output symbols.

CSS_Symbol_encoder.gvi	Terminal name	Type	Description
Input	*Bit stream*	Signed integer array (1D)	
	SF	Unsigned integer	
Output	*Symbols*	Unsigned integer array (1D)	

3.5. Questions

3.5.1. Generate four chirps by **CSS_modulate** block with the following configuration (Do not use CSS_Symbol_encoder block for this question.):

- BW = 500k
- IQ rate = 500k
- SF = 7

The first chirp is the base chirp (start frequency = -B/2). Find the right symbols for the rest three chirps whose starting frequencies are equally spread out over the bandwidth (e.g. the second chirp starts at -B/4, the third at DC, and the fourth at B/4). Use **P2S** block from lab8 to serialize the 2D output chirp signal from **CSS_modulate** block, then use spectrogram block to display the four chirp signals. Use 16 for the window size of the spectrogram. Report the followings:

1) Four symbols
2) Output spectrogram

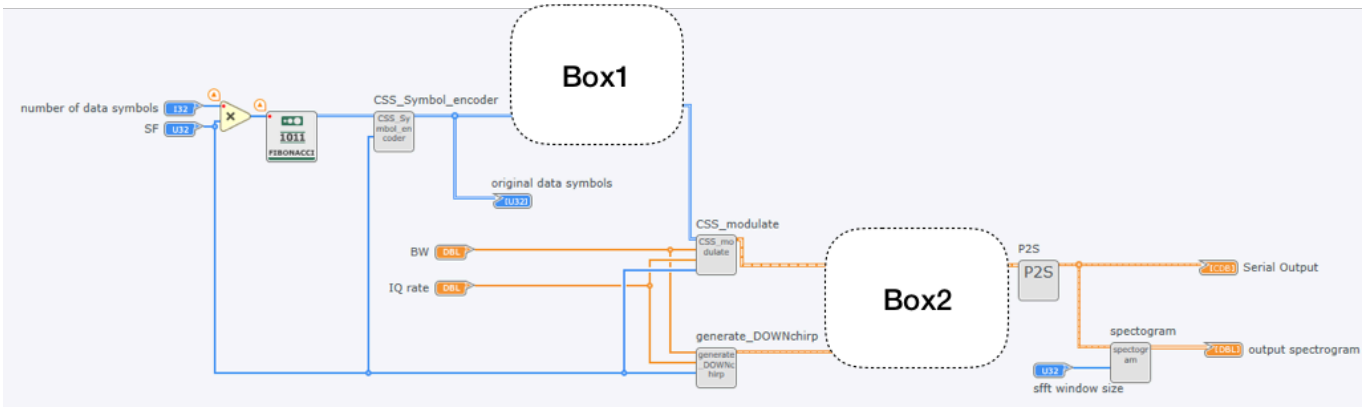
3.5.2. We will generate a complete LoRa packet including a preamble sequence (8 base chirps), a sync sequence (2 down-chirps) and the data. Complete the following figure: Box1 adds 8 base chirps (symbol 0) as a preamble sequence, and Box2 inserts two down-chirps after the preamble sequence. Use **Initialize Array** and **Build Array** for Box1 and **Insert into Array** and **Build Array** for Box2. Use the following configuration.

- BW = 500k
- IQ rate = 500k
- SF = 7
- Number of data symbols = 4
- Window size = 16

Report the followings:

1) original data symbols
2) output spectrogram

sim_Tx.gvi	Terminal name	Type	Description
Input	*BW*	Double	Bandwidth
	IQ rate	Double	Sampling frequency
	SF	Unsigned integer	Spreading factor
	Number of data symbols	Unsigned integer	
	Window size	Unsigned integer	
Output	*Serial output*	Complex double array (1D)	
	Output spectrogram	Double array (2D)	
	Original data symbols	Unsigned integer array (1D)	



4. LoRa: Rx

In this section, we will implement the blocks required in the LoRa receiver. The following subVI's will be implemented:

- **CSS_demodulate.gvi**
- **coarse_alignment.gvi**
- **fine_alignment.gvi**
- **CSS_decoder.gvi**

4.1. CSS Demodulation

CSS demodulation is done by multiplying the received signal with the down-chirp signal. Implement a block that divides the received CSS signal with length N and multiplies each segment with the down-chirp signal.

CSS_demodulate.gvi	Terminal name	Type	Description
Input	*BW*	Double	Bandwidth
	IQ rate	Double	Sampling frequency
	SF	Unsigned integer	Spreading factor
	Received CSS	Complex double array (1D)	
Output	*Demodulated CSS*	Complex double array (1D)	

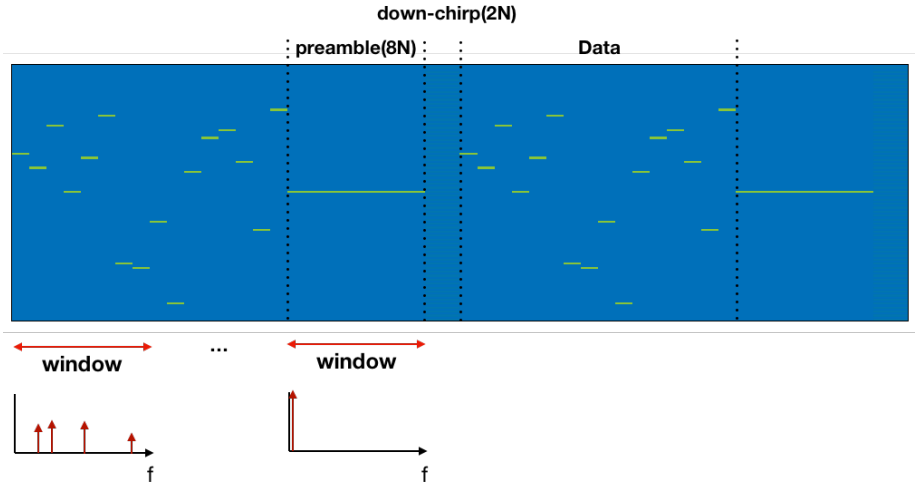
- Use **S2P** block to divides the input signal with length *N* ($\boldsymbol{N = 2^{SF}}$). Use a for loop and auto indexing to access a row of the output 2D array of the S2P block and multiply with the down-chirp generated by **generate_DOWNchirp** block**.**

- Serialize the output 2D array (demodulated signal) from the for loop by **P2S** block.

4.2. Coarse Alignment

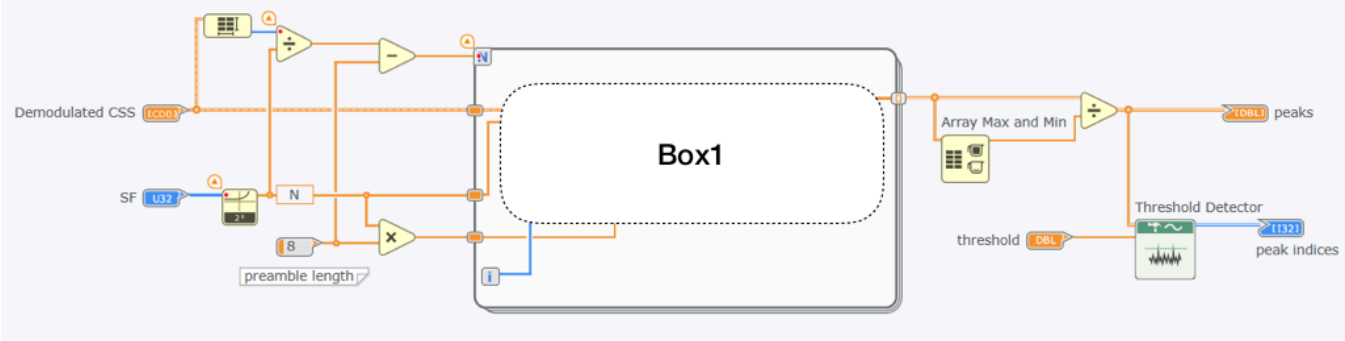
In this section, we will implement a block that detects the preamble and aligns the packet in *chirp-wise*. The block assumes the first sample of the received signal is the beginning of a chirp. The residual misalignment will be covered in the next section (fine alignment).

The following figure shows a spectrogram of unaligned demodulated CSS. In this example, the length of preamble is *8N*. The coarse alignment detects the preamble by sliding a window of length *8N* (the length of preamble) and computes the magnitude of FFT. When the window aligns with the preamble, the magnitude is maximized because the energy of the demodulated signal is summed up at a single FFT bin.



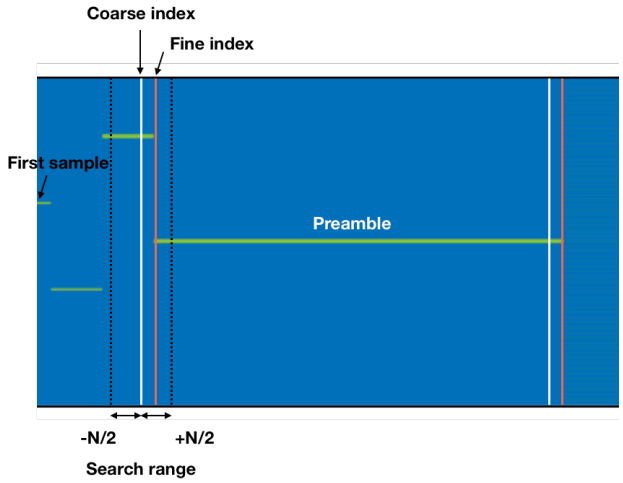
coarse_alignment.gvi	Terminal name	Type	Description
Input	*Demodulated CSS*	Complex double array (1D)	
	SF	Unsigned integer	Spreading factor
	threshold	Double	
Output	*peaks*	Double array (1D)	
	Peak indices	Signed integer array (1D)	

Complete Box1 in the following figure. Box1 performs FFT by sliding window and returns the maximum magnitude of the FFT. The peak array represents the maximum magnitude of each FFT.



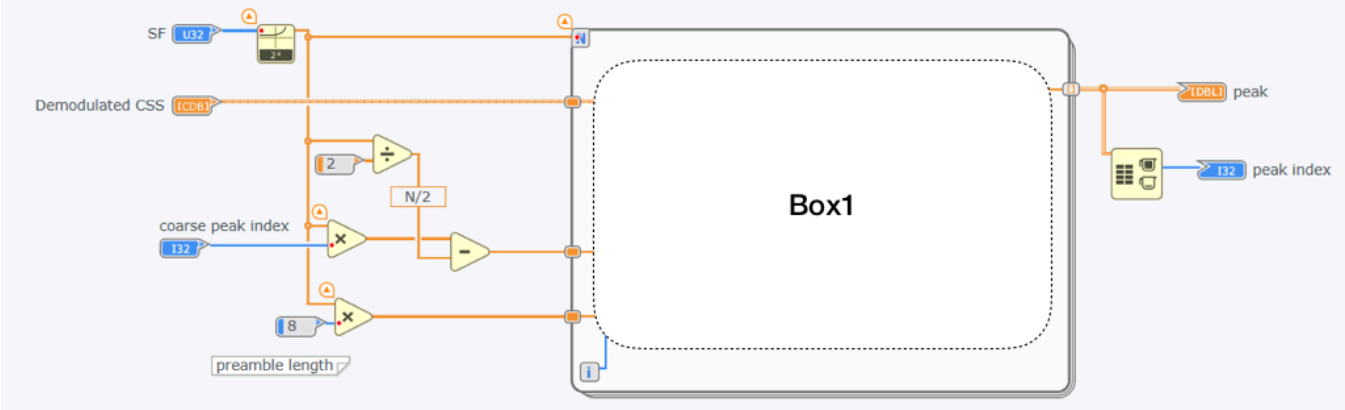
4.3. Fine Alignment

In the coarse alignment, it assumes the first sample of the received signal is the beginning of a chirp. However, the receiver can start sampling at any time, so the first sample can be any portion of a chirp. In fine alignment, we search the index from -N/2 to N/2 centered at the coarse-aligned index. The block aligns the packet in *sample-wise.* The following figure shows the first sample is not the beginning of a chirp which leads the coarse index off by a few samples. The fine alignment block finds the exact start of the preamble.



fine_alignment.gvi	Terminal name	Type	Description
Input	*Demodulated CSS*	Complex double array (1D)	
	SF	Unsigned integer	Spreading factor
	Coarse peak index	Signed integer	Index of the first coarse peak
Output	*peak*	Double	
	Peak index	Signed integer	

Complete Box1 in the following figure. Box1 performs FFT by sliding window and returns the maximum magnitude of the FFT.



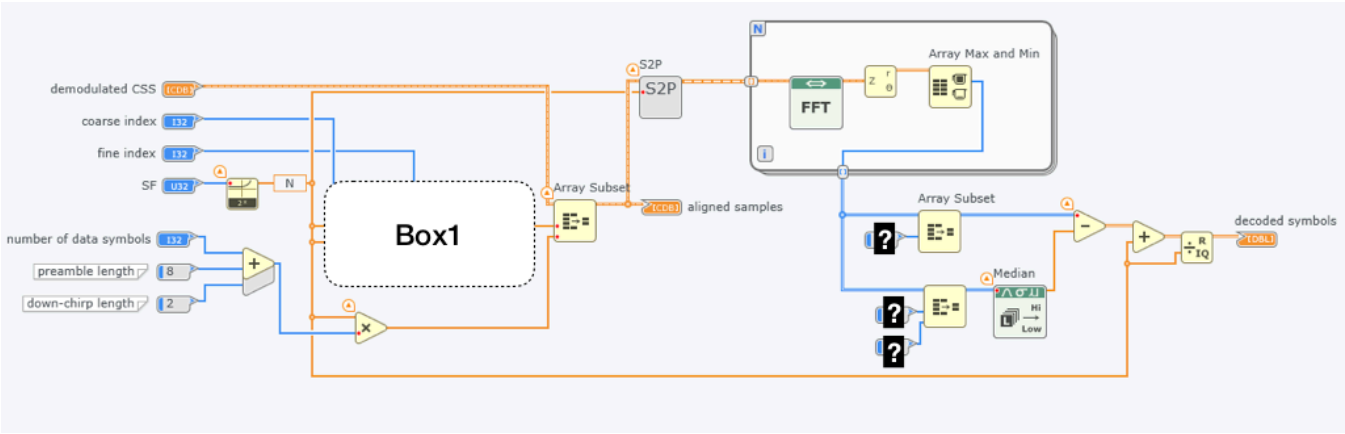
4.4. Decoding

The decoder block completes the following process:

1. Align the packet by using the coarse and fine index. The aligned packet starts from the preamble.
2. Divide (or parallelize) the aligned packet by chirps (N samples), then performs FFT.
3. Find the index that has the maximum magnitude of the FFT.
4. Calculate the frequency shift caused by CFO or STO on the preamble whose frequency bin should be 0.
5. Correct the frequency shift on the data symbols.

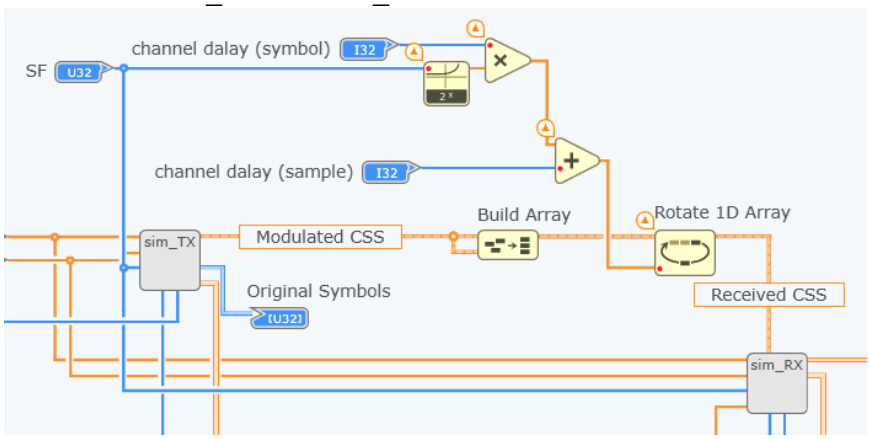
CSS_decoder.gvi	Terminal name	Type	Description
Input	*Demodulated CSS*	Complex double array (1D)	
	Coarse index	Signed integer	
	Fine index	Signed integer	
	SF	Unsigned integer	Spreading factor
	Number of data symbols	Unsigned integer	
Output	*Decoded symbols*	Signed integer array (1D)	
	Aligned samples	Complex double array (1D)	

Complete Box1 in the following figure. Box1 calculate the right index to align the packet. Find the right indices hidden by the question marks to obtain the data symbols and the preamble.



4.5. Question

4.5.1. Verify the transmitter and receiver blocks by simulation. Build **sim_RX** block similar to **sim_TX** block (Connect all four blocks of Rx together). Simulate the channel delay as the following figure. The detail connections with sim_TX and sim_RX can be different.



Try the following configuration for the simulation:

- SF = 7
- Number of data symbols = 32
- Channel delay (symbol) = 10
- Channel delay (sample) = 24

Report the followings:

- Spectrogram of the aligned demodulated CSS (use 128 for the window size)
- First 8 symbols each for the original symbols (Tx) and the decoded symbols (Rx)

5. LoRa: USRP

Once the blocks are verified by the simulation, implement them in USRP. You can reuse sim_Tx and sim_Rx blocks to simplify the connections. Note that there is no need for up/down-sampling and pulse-shaping/match-filtering. In the receiver, calculate the number of samples to capture two LoRa packets (preamble + data).

5.1. Question

5.1.1. Run your LoRa transceiver in wireless. Use two USRPs: one for Tx and the other for Rx. Use the following configuration:

- Carrier frequency = 915MHz
- BW = 500kHz
- IQ rate = 500kHz
- Number of data symbols = 64
- SF = 7

Report the followings:

- Spectrogram of the modulated CSS at Tx (use 8 for the window size)
- Spectrogram of the aligned demodulated CSS at Rx (use 128 for the window size)
- First 8 symbols each for the original symbols (Tx) and the decoded symbols (Rx)
- Total number of symbol errors