COMMUNICATION DESIGN PROJECT

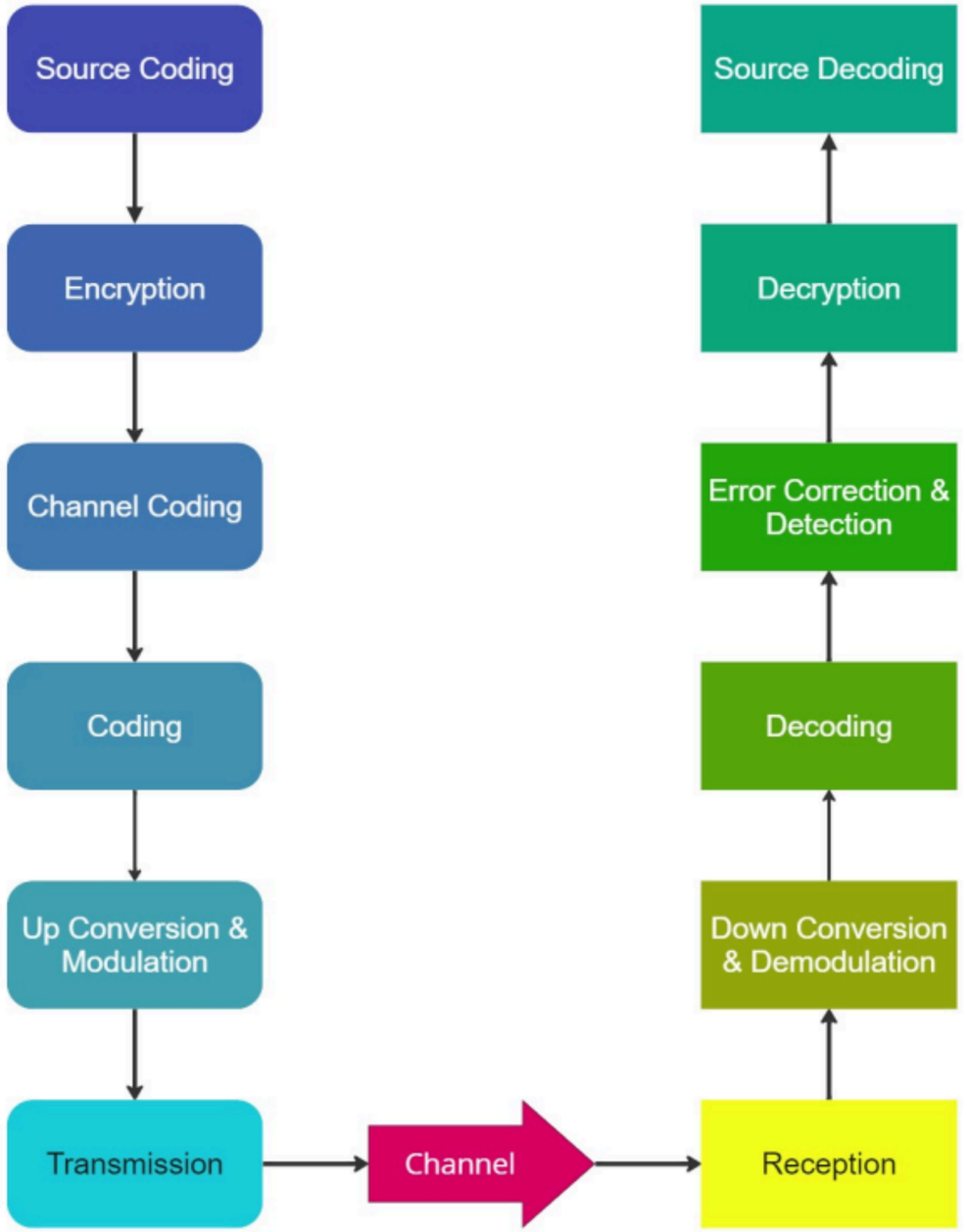
Team - Lotus

U.K.G.P.M.B. WIJETHILAKA	210724K
H.M.G.N.I. HERATH	210216F
T.R.WATHUDURA	210665L
W.K.D.D.D.WIJEWICKRAMA	210728C

Design Methodology

Step 01

Identifying and Designing the arrangement of blocks



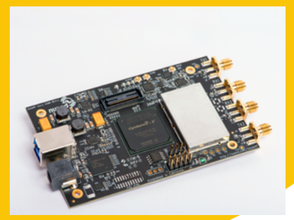
Design Methodology (Ctd.)

Step 2:
Design a Channel Model

- With identified **blocks, simulate transmission** with **virtual channel** model.
- Tool Used: **GNU Radio**

Step 3:
Transmit Through a Real Channel

- Tested and **implemented** the simulated system to **transmit and receive** types of data through a real channel.
- Device Used: **BladeRF**



Step 4:
Adding New Blocks per Design Requisites

- Added new blocks for
  - Reliability
  - Different **types of Data**
  - **Distance**
  - With and without **jamming.**

Wireless Communication

Devices

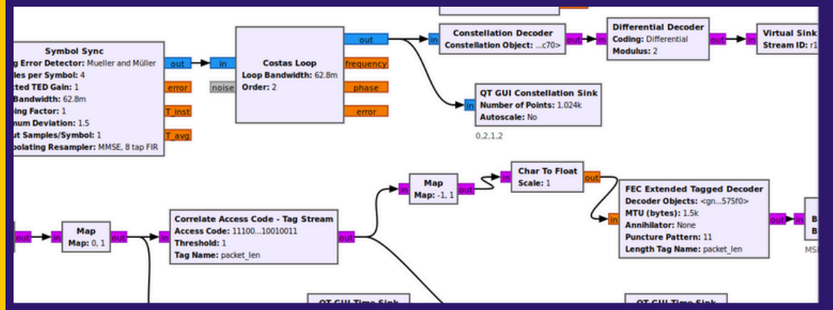
We utilize the **BladeRF** device for wireless communication, opting for practical implementation over simulation by using the **SoapyBladeRF Source** and **SoapyBladeRF Sink** blocks into the flowgraph to directly interface with the BladeRF hardware.

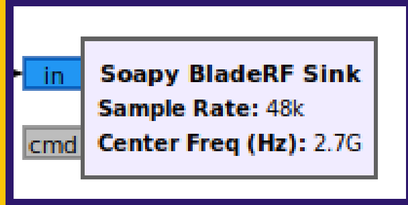
SoapyBladeRF source

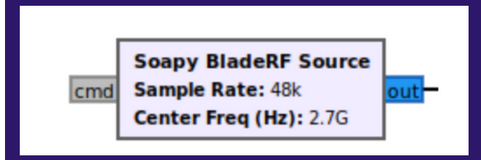
For wireless communication using the BladeRF device, the SoapyBladeRF Source block in GNU Radio receives signals from the BladeRF hardware. It establishes a connection, enabling the flowgraph to process and analyze incoming data.

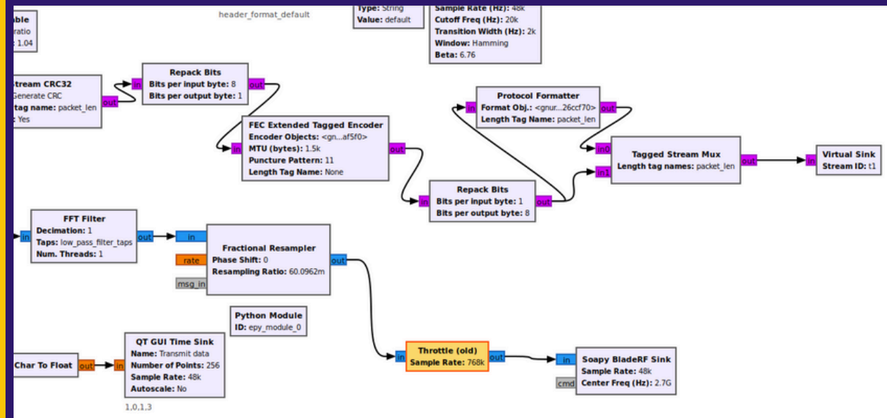
SoapyBladeRF sync

On the transmission side, the SoapyBladeRF Sink block facilitates signal transmission using the BladeRF device through the SoapySDR interface. It is responsible for sending processed signals to the BladeRF hardware for wireless transmission.









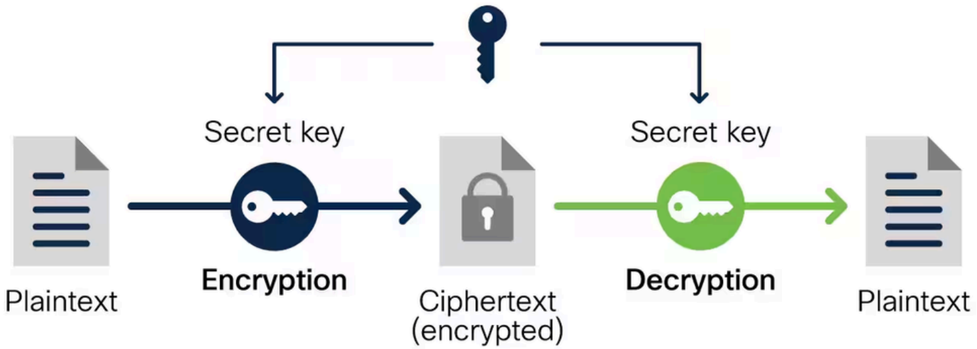
Encryption

Mainly 2 types :

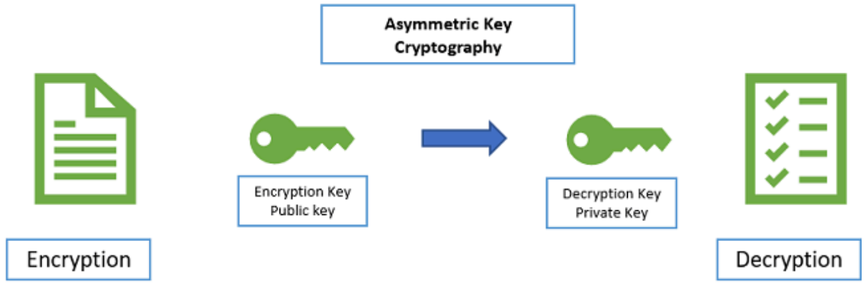
1. Symmetric key encryption (private key)
2. Asymmetric key encryption (public key)

The main difference:

1. Symmeteric - use same key to encrypt and decrypt
2. Assymetric - use pair of keys



Symmetric



Asymmetric

Symmetric Key Encryption

In here we use Symmetric key encryption.

Why?

1. Faster than asymetric key - beacause we use large scale files and real time transmission
2. Easy to key management

Examples:

- Data Encryption Standard (DES):
  - Short key - 56-bit key
  - Vulnerable to brute-force attacks due to short key length.
- Advanced Encryption Standard (AES):
  - variable length key
  - Stronger security with longer key lengths

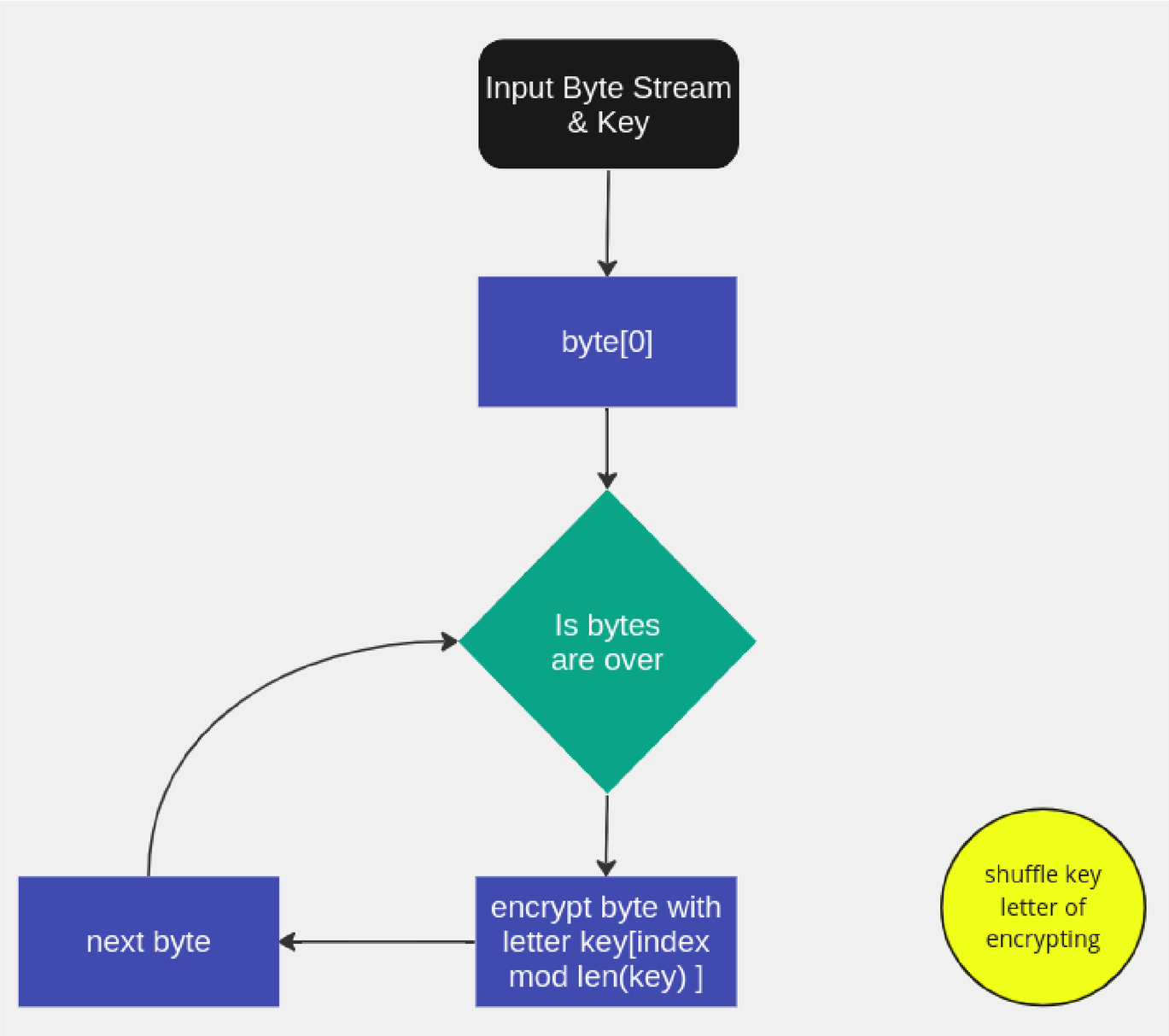
In here we use Static key XOR method to encrypt and decrypt

Our encryption method

enc1.py > ...

```python
# Python code for encryption

# take path of the file and key as an input
path = input('Enter the file path: ')
key = input('Enter Key for decryption: ')

try:

    fin = open(path, 'rb')  # open file for reading purpose
    file = fin.read() # storing file's data in variable "file"
    fin.close()
    file = bytearray(file) # converting file into byte array to perform decryption easily on numeric data

    # performing XOR operation on each value of bytearray
    for index, values in enumerate(file):
        # Convert key character to integer for XOR operation
        key_byte = ord(key[index % len(key)])
        file[index] = values ^ key_byte

    # opening file for writing purpose
    with open(path, 'wb') as fin:
        # writing decrypted data in the file
        fin.write(file)
    # print path of file and decryption key that we are using
    print('The path of file: ', path)
    print('Key for decryption: ', key)
    print('Encryption Done...')

except FileNotFoundError:
    print("File not found. Please enter a valid file path.")
except Exception as e:
    print(f"An error occurred: {str(e)}")

```

Encryption algorithm



Our Decryption method

dec1.py > ...

```python
# Python code for decryption

# take path of the file and key as an input
path = input('Enter the file path: ')
key = input('Enter Key for decryption: ')
```

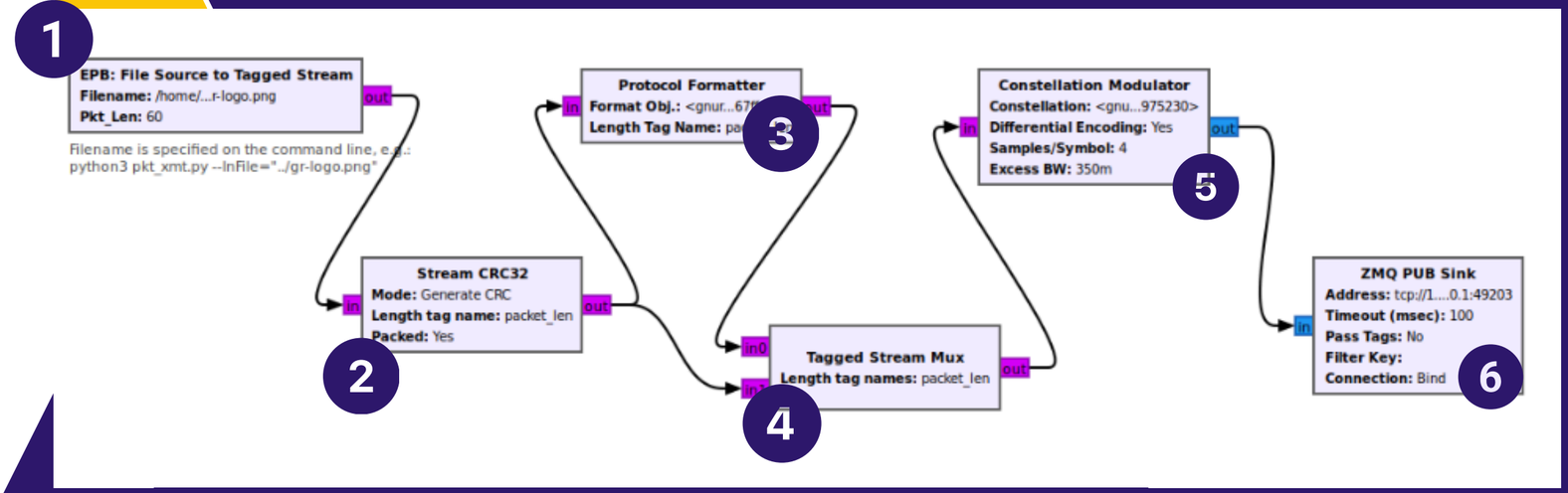
ge6.py

```python

try:

    fin = open(path, 'rb') # open file for reading purpose
    file = fin.read()    # storing file's data in variable "file"
    fin.close()
    file = bytearray(file)  # converting file into byte array to perform decryption easily on numeric data

    # performing XOR operation on each value of bytearray
    for index, values in enumerate(file):

        key_byte = ord(key[index % len(key)])  # Convert key character to integer for XOR operation
        file[index] = values ^ key_byte

    # opening file for writing purpose
    with open(path, 'wb') as fin:
        fin.write(file)     # writing decrypted data in the file

    print('Decryption Done...')

    # print path of file and decryption key that we are using
    print('The path of file: ', path)
    print('Key for decryption: ', key)

except FileNotFoundError:
    print("File not found. Please enter a valid file path.")
except Exception as e:
    print(f"An error occurred: {str(e)}")
```

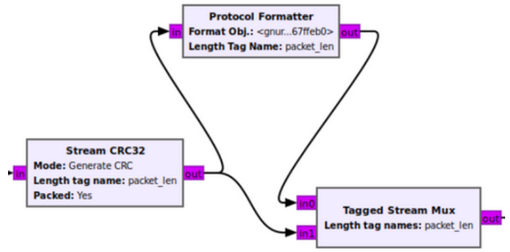
Transmitter

When the packet length is higher it is more efficient for transmitting and reduces overhead

- read data from a file and convert it into a tagged stream.



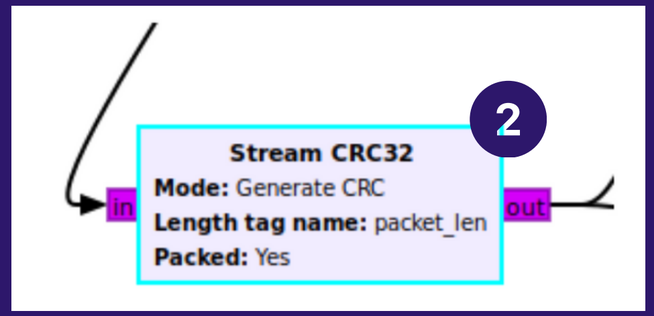
Packetizing



- The file source is converted into a tagged stream and packetized, with CRC bits and headers added to each packet before undergoing further processing.

What is a tagged stream?

- refers to a data stream that includes additional metadata, known as tags, associated with specific elements of the data
  - Helps in synchronization
  - Helps organize data in stream

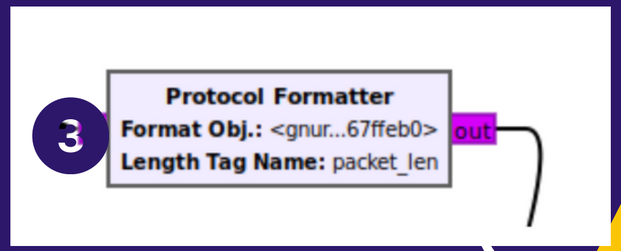


CRC

CRC is an error-checking technique used in computing and data communication to detect changes or errors in data during transmission.

- Each packet is added CRC bits.
- We have used Stream CRC block, which processes data in a continuous stream.

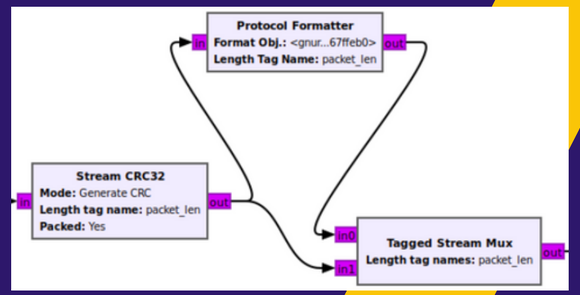
(Asynchronous CRC is used when data arrives irregularly)



- Creates a header.

Frame Length / Error Checking Information / Timestamps

- Defines the overall structure of the communication frame or packet.
- Access code is added in each packet during this stage



Payload is sent as a parallel tagged stream, then they are muxed togeather

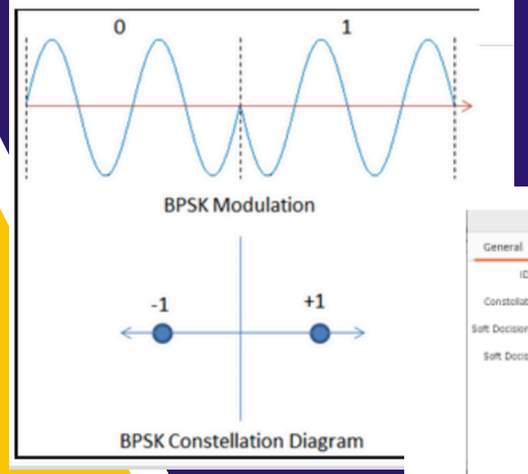
Which Modulation scheme?

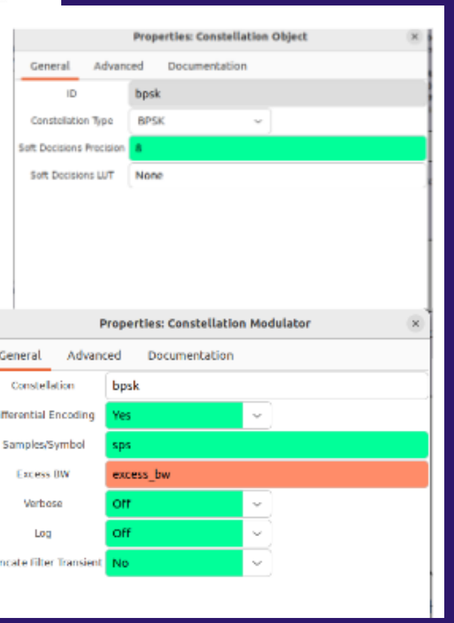
- Amplitude Shift Keying ?
- Phase Shift Keying ?
- 64 QAM?
- QPSK?
- BPSK?

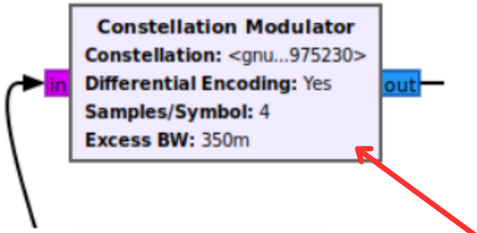
BPSK modulation

Binary Phase Shift Keying (BPSK) is a digital modulation technique using two phases of a carrier signal to represent binary data. It is simple and reliable, suitable for applications like wireless communication. BPSK involves altering the carrier signal's phase to encode binary 0s and 1s, offering robust transmission over noisy channels

- BPSK is relatively simple and easy to implement d
- Another advantage of BPSK is its excellent noise immunity. By using two different phases to represent binary data, BPSK can effectively combat noise and interference.
- BPSK also has a high spectral efficiency, enabling the transmission of large amounts of data within limited bandwidth.

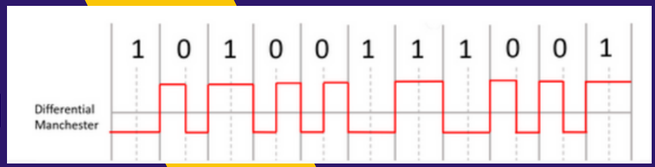




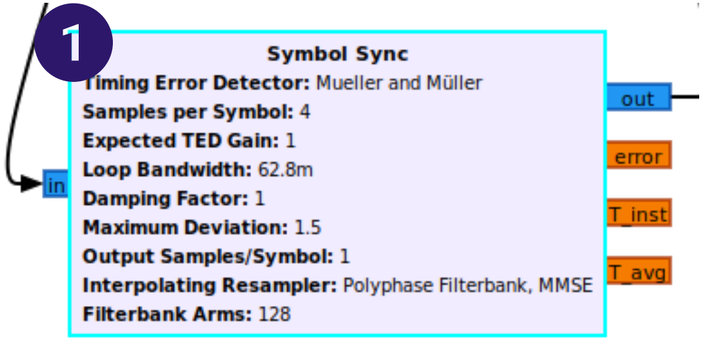


"A block in digital signal processing that engages in modulation by associating digital symbols with coordinates in a complex plane."

Differential Encoding: Channel Coding Method

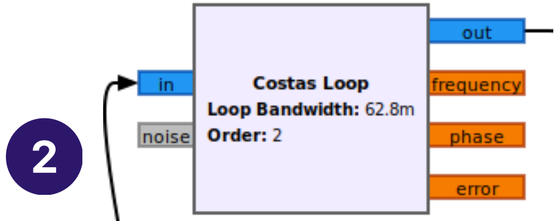


Receiver



- **Symbol Sync:** syncs the timing of a received.

  Signal. Synchronization ensures that the receiver **samples** the received signal precisely at the **symbol boundaries.**

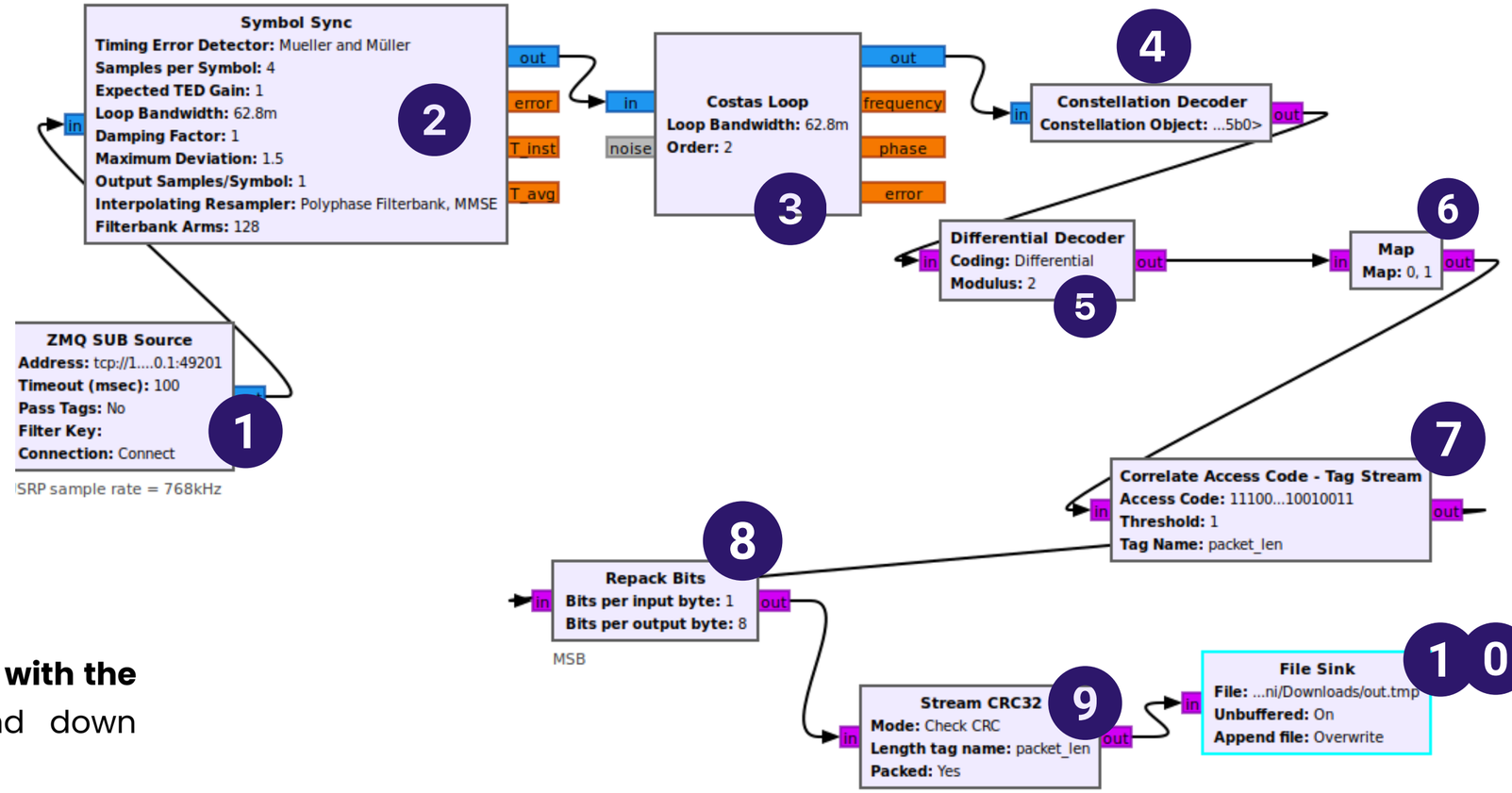


Order of Costa's loop is 2 since we are using BPSK

- **Costa's loop** aims to precisely **synchronize itself with the carrier frequency** of the incoming signal and down converts signal into baseband .
- The Costa's loop utilizes a phase error detector to measure and **adjust the local oscillator's phase,** enabling precise alignment with the received signal for accurate tracking.

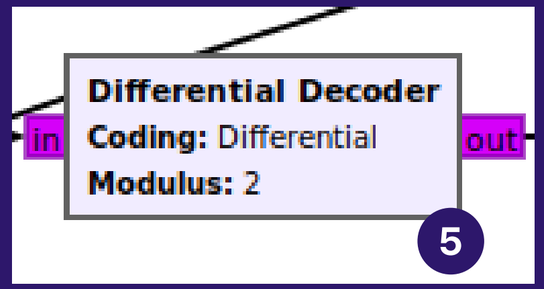
Why didn't we use the Pholyphase Clock Sync?

Symbol sync ensures accurate demodulation (alignments of recived signals)



Decoding Process: The decoding process involves interpreting the received signal to recover the original information.

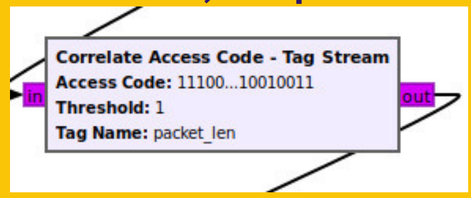
Constellation Decoder:
convert complex points in a **signal space** (typically representing modulated symbols) back into their **corresponding symbols**, using a constellation object.



Differential Decoder:
designed for decoding differentially encoded symbols.

Access Code Check

is a sequence of characters used to gain entry or access to a communication system, network, or specific service.



- **Identifies a specified access code in a stream, extracts a header containing the frame length, and outputs a tagged stream with payload bits based on the detected access code.**

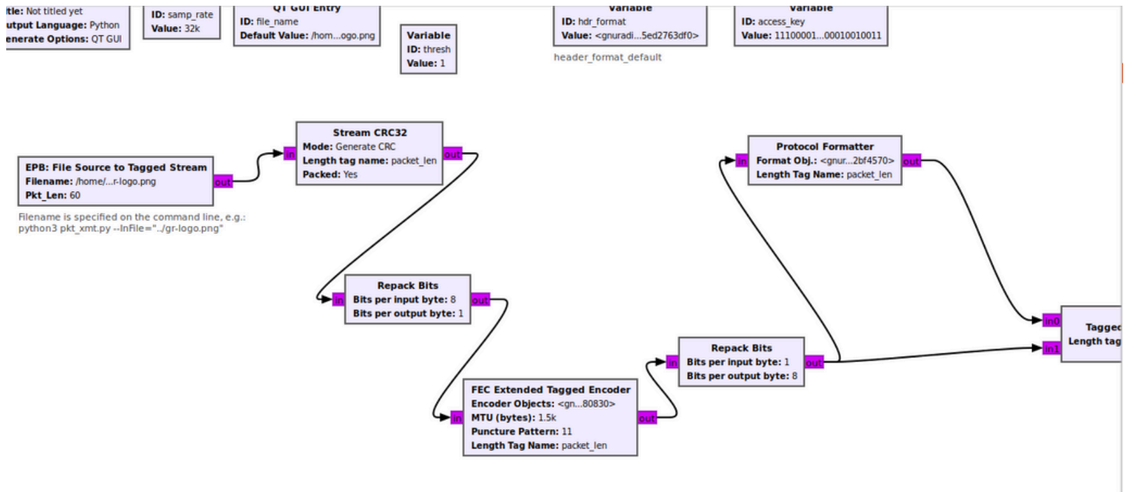
systems can ensure that only authorized devices or users can access and communicate on the network.

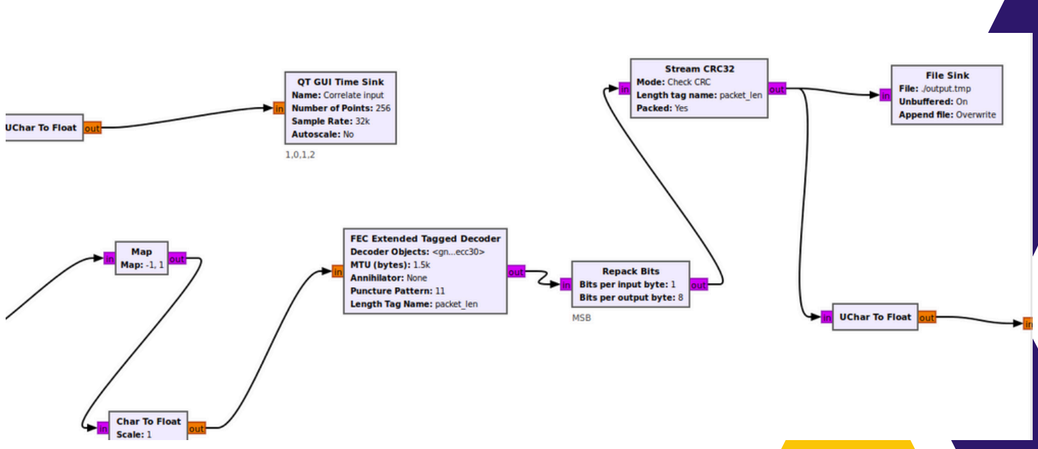
Preamble: Predefined bits sent before the data transmitted which helps the receiver to sync its clock and frame timing with incoming data.

We use a python code to remove the preamble of the recived file at the reciver.

Error correction for reliability

- In the project, we have used FEC (forward error correction) method.
- **FEC:** FEC **corrects errors in transmitted data without requiring retransmission,** enhancing communication reliability. This is done by by **adding redundant information to the transmitted data, allowing the receiver to detect and correct errors based on the redundant bits** without needing to request retransmission.
- For the implementation of FEC, we have used the blocks, **FEC extended tagged encoder and FEC extended tagged decoder.**

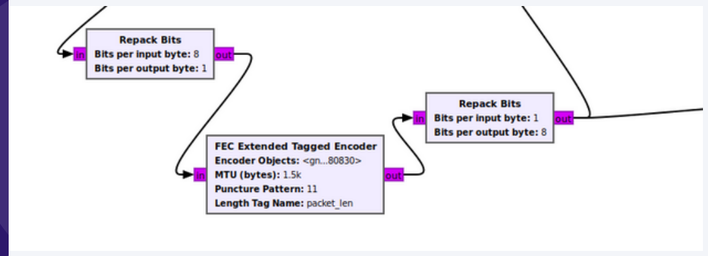




Foward Error Correction

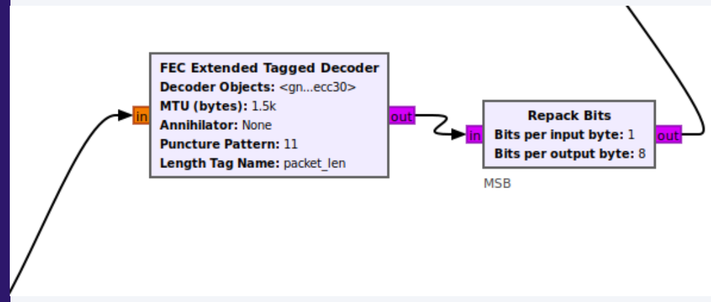
FEC extended tagged encoder

- The Forward Error Correction **(FEC) Tagged**
- **Encoder** is applies a forward error
- **Correction code** to the data. FEC introduces redundancy into the stream, allowing the receiver to detect and correct errors. The tags are preserved during this process.



FEC extended tagged decoder

- This block is designed to **decode a tagged strea**m
- That has undergone FEC encoding. It processes the **encoded data, detects errors**, and **attempts to correct them** based on the **redundancy** introduced during encoding.

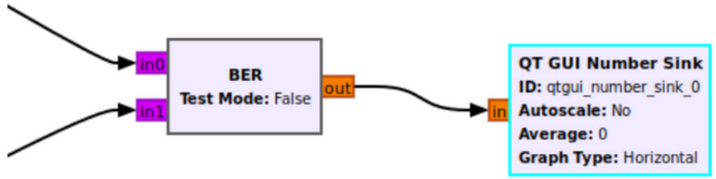


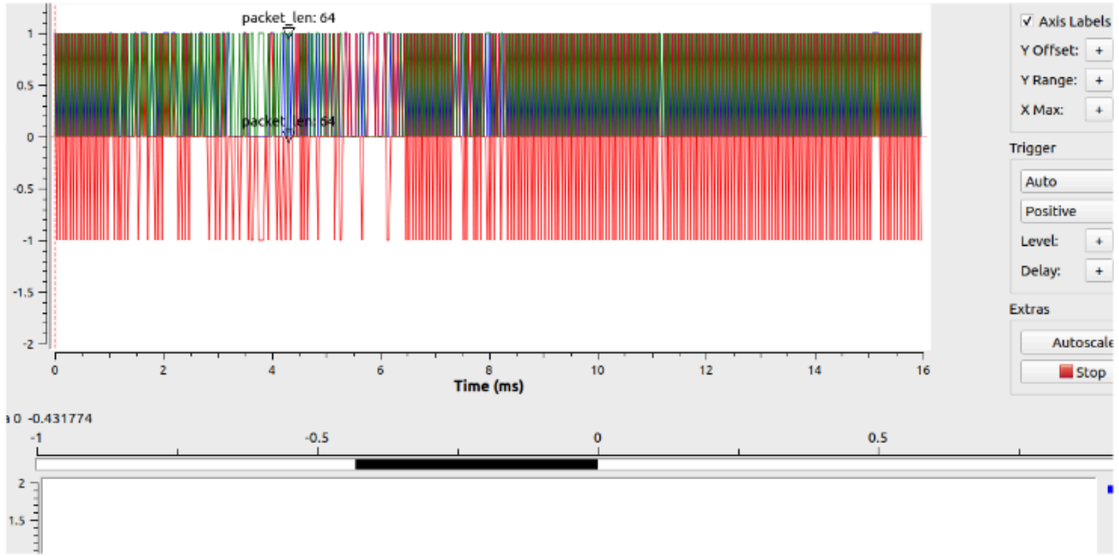
Repack bits block

- *Before the Encoder*: Takes **groups of 8** bits (1 byte) from the input and **repacks** them **as individual** bits in the output.
- *After the encoder*: Takes individual bits from the FEC-encoded data and **repacks them into groups of 8** bits (1 byte).

BER

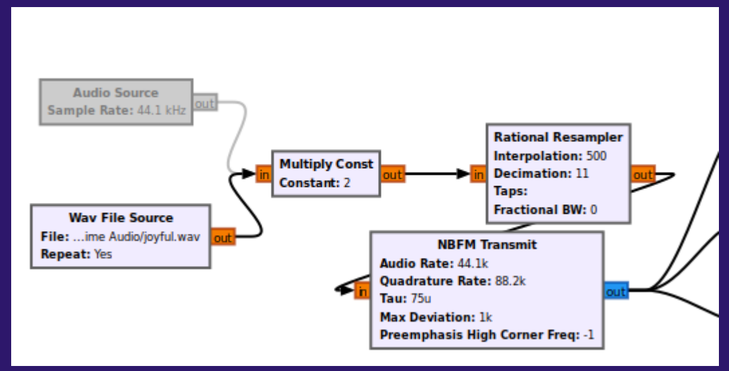
- We checked the BER between the transmitted signal and the received signal.
- **BER:** the ratio of the number of bits received in error to the total number of bits transmitted.



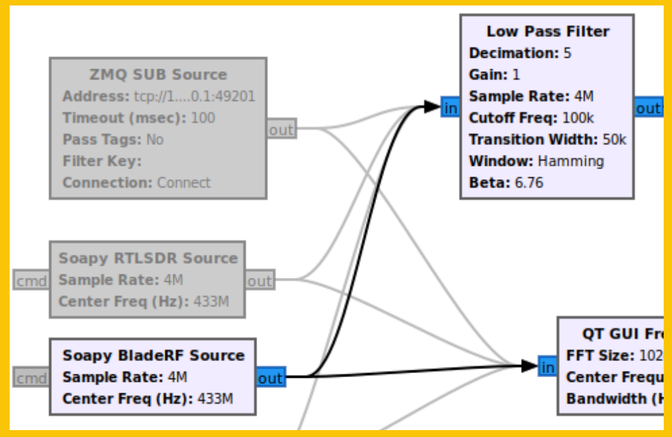


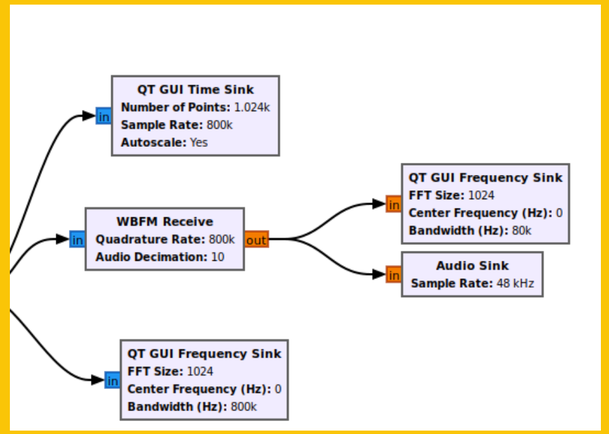
Real-Time Audio Transmission with NBFM

Transmittor



Reciever





- NBFM: Modulates carrier frequency based on audio signal amplitude.
- Maintains constant carrier amplitude during frequency modulation.
- Creates frequency sidebands around the carrier for transmission.
- Efficient spectrum use with narrow frequency deviation.

THANK YOU