Middle East Technical University
Department of Computer Engineering
Wireless Systems, Networks and Cybersecurity (WINS) Laboratory

Orthogonal Frequency-Division Multiplexing (OFDM) Transmitter and Receiver with Software Defined Radios

CENG513
Wireless Communications and Networking
2020-2021 Spring
Term Project Report

Prepared by
Berker Acır
Student ID: 2098697
berker.acir@metu.edu.tr
Computer Engineering
8 July 2021

Abstract

Orthogonal Frequency Division Multiplexing (OFDM) is one of the most crucial techniques in wireless communication technologies in which, high data rates are achieved using densely packed multiple orthogonal carriers. In this term project, OFDM transmitter and receiver is designed with GNU Radio software and studied in details. The OFDM system is tested with USRP B210 SDRs with different configurations in terms of bit errors, reception time. At the receiver side, received OFDM signal's in-phase and quadrature values are saved for creating a signal dataset.

Table of Contents

List of Figures

List of Tables

1 Introduction

Growth in the number of devices connected to the networks and emerging new usage scenarios [1] such as enhanced mobile broadband (eMBB), ultra-reliable low-latency communications (URLLC), massive machine type communications (mMTC) for 5G wireless access technology, known as New Radio (NR), have a large influence on the development of more efficient and flexible wireless communications systems. OFDM (Orthogonal Frequency-Division Multiplexing) technique is one of the key element behind the advances in the wireless communication. OFDM is a part of LTE (Long-Term Evolution), WLAN (Wireless Local Area Network), DVB (Digital Video Broadcasting) and BWA (Broadband Wireless Access) standards due to its advantages over single-carrier schemes. These advantages are OFDM's efficiency in dealing with multipath fading, channel delay spread, enhancement of channel capacity, modification of modulation density and robustness of narrowband interference [2]. However, OFDM is highly sensitive to carrier frequency offsets, frequency phase noise and sampling clock offsets.

In OFDM, high rate encoded data is divided into parallel sub-streams and orthogonal carriers are used for modulating these parallel sub-streams that are transmitted at different frequencies simultaneously [3]. Pilot sub-carriers are used for detecting frequency offsets and phase noise and sub-carriers overlap without interfering and also maximum spectral efficiency is attained without causing adjacent channel interference due to orthogonality of sub-carriers [4]. Multipath channels result in losing orthogonality of carriers. Cyclic prefix (CP) is used for restoring the orthogonality back and it also allows the signal to be decoded even if the packet is detected after some delay.

Software defined radio (SDR) concept provides designing of flexible communication systems where the flexibility comes from reliance on the software implemented in a programmable device rather than specific hardware. Therefore, the system can be configured with different settings before operating and even reconfigured in run-time. Hence, SDRs paves the way for designing and testing digital communication systems in a very practical manner.

The goal of this work is designing transmitter and receiver system with SDRs which employ OFDM scheme in communication and then, signal dataset is created from the received signals in the system. Signal dataset contains I/Q samples of received signals under different configurations. This dataset may be further used for channel classification task.

The rest of this term project report is organized as follows. Section 2 shows basic of GNU Radio software and explains in-phase and quadrature of signals in brief. In Section 3, OFDM transmitter and receiver design in GNU Radio is studied in details. Information about experimental setup for this project is given in Section 4. Section 5 provides insight about how the system is operating, explains how the system is tested, shows the test results and the discussion about the test results and the system. Section 6 concludes this report.

2 Background Information

2.1 GNU Radio

GNU Radio is a free software development framework that provides signal processing functions for implementing software-defined radios. The framework offers a graphical design approach, namely GNU Radio Companion (GRC), in addition to supporting development in Python and C++. Supported globally by the open-source community and widely used in government, commercial and academic environments, GNU Radio gives users access to a diverse set of existing projects focused on wireless communications research and implementation of real-world radio systems.

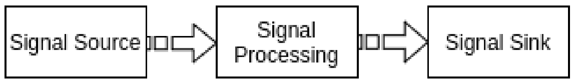
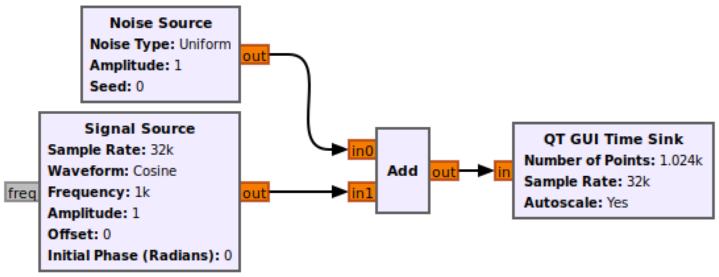


Figure 1 . Basic flow-graph of GRC

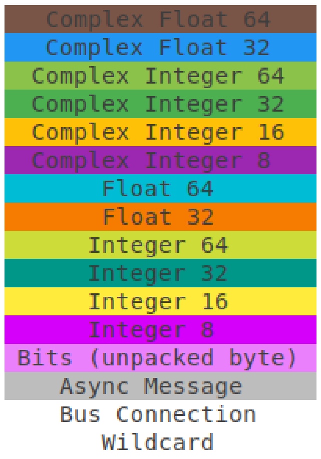


Noise Source
Noise Type: Uniform
Amplitude: 1
Seed: 0

Signal Source
Sample Rate: 32k
Waveform: Cosine
Frequency: 1k
Amplitude: 1
Offset: 0
Initial Phase (Radians): 0

QT GUI Time Sink
Number of Points: 1.024k
Sample Rate: 32k
Autoscale: Yes

Figure 2 . Simple flow-graph in GRC



Complex Float 64
Complex Float 32
Complex Integer 64
Complex Integer 32
Complex Integer 16
Complex Integer 8
Float 64
Float 32
Integer 64
Integer 32
Integer 16
Integer 8
Bits (unpacked byte)
Async Message
Bus Connection
Wildcard

Figure 3 . Data types in GNU Radio

The most basic structure of GNU Radio and simple GNU Radio Companion design are as shown in Fig 1 and Fig 2 . Signal Source block generates signal and there are different signal source blocks available in GRC such as USRP, sound card (microphone), signal source as in Fig 2 . In Signal Processing block, received signal is manipulated before it's passed to the next block. Most of the base-band signal processing blocks such as FIR filters, IIR filters, FFT, Multiplier etc. are already available in GNU Radio. Signal Sink block is where the signal is translated into the form that user wanted. Examples of signal sink blocks are sound card (speaker), USRP, data in integer form, graph (one example is *QT GUI Time Sink* in Fig 2) etc. After flow-graph design is finished in GRC, it generates a executable Python or C++ code. GNU Radio also provides interface for creating user-desired blocks.

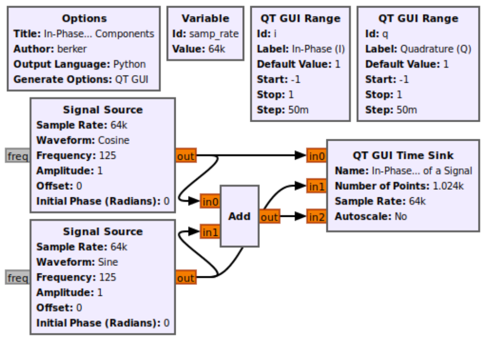
Source block produces signals in different data types as in Fig 3 and signal processing blocks may even change the signal's data type. For example in Fig 2 , Signal Source block generates Cosine wave and Noise Source block creates uniform noise in *float32* data type.

2.2 In-Phase and Quadrature (I/Q)

A sinusoid with angle modulation can be decomposed into two amplitude-modulated sinusoids that are offset in phase by one-quarter cycle (90 degrees or $\pi/2$ radians). All three functions have the same center frequency, and such amplitude modulated sinusoids are known as the **in-phase** and **quadrature** components.

From the trigonometry identity $\sin(\alpha+\beta) = \sin(\alpha)\cos(\beta) + \cos(\alpha)\sin(\beta)$, every sinusoid can be expressed as the following:

$$\begin{aligned} x(t) &= A(t)\sin(\omega t + \phi) = A(t)\sin(\phi + \omega t) \\ &= [A(t)\sin(\phi)]\cos(\omega t) + [A(t)\cos(\phi)]\sin(\omega t) \\ &= \mathbf{I(t)}\cos(\omega t) + \mathbf{Q(t)}\sin(\omega t) \end{aligned}$$


Options
Title: In-Phase... Components
Author: berker
Output Language: Python
Generate Options: QT GUI

Variable
Id: samp_rate
Value: 64k

QT GUI Range
Id: i
Label: In-Phase (I)
Default Value: 1
Start: -1
Stop: 1
Step: 50m

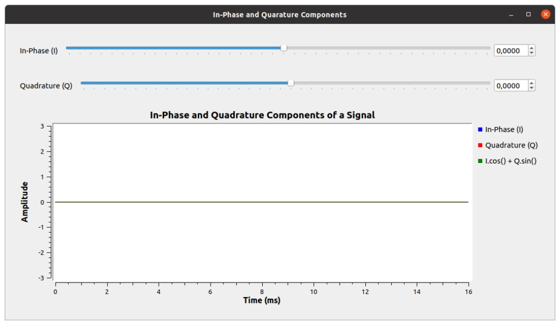
QT GUI Range
Id: q
Label: Quadrature (Q)
Default Value: 1
Start: -1
Stop: 1
Step: 50m

Signal Source
Sample Rate: 64k
Waveform: Cosine
Frequency: 125
Amplitude: 1
Offset: 0
Initial Phase (Radians): 0

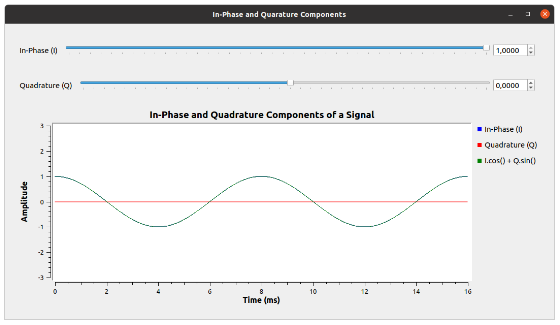
Signal Source
Sample Rate: 64k
Waveform: Sine
Frequency: 125
Amplitude: 1
Offset: 0
Initial Phase (Radians): 0

QT GUI Time Sink
Name: In-Phase... of a Signal
Number of Points: 1.024k
Sample Rate: 64k
Autoscale: No

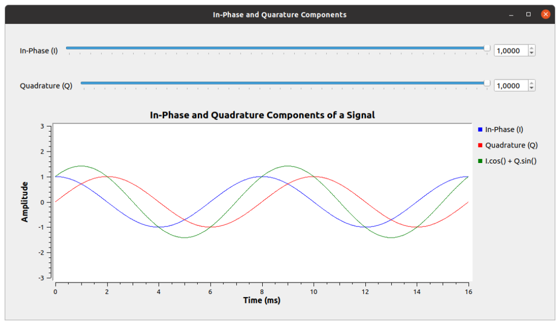
(a) GRC flow-graph for generating I/Q signal


In-Phase and Quarature Components
In-Phase (I) 0,0000
Quadrature (Q) 0,0000

(b) I = 0, Q = 0


In-Phase and Quarature Components
In-Phase (I) 1,0000
Quadrature (Q) 0,0000

(c) I = 1, Q = 0


In-Phase and Quarature Components
In-Phase (I) 1,0000
Quadrature (Q) 1,0000

(d) I = 1, Q = 1

Figure 4 . GRC flow-graph and its run-time frames for generating signal with different in-phase and quadrature values.

There is a GRC flow-graph in Fig 4 a for visualization of the in-phase and quadrature components of a signal. It simply generates $\mathbf{I} \cdot \cos()$ and $\mathbf{Q} \cdot \sin()$ waves and it adds this waves for creating a new signal. Then, these waves are shown in amplitude vs. time plot.

In-phase and quadrature values are actually used for representing the complex numbers as seen in modulation schemes. For example:

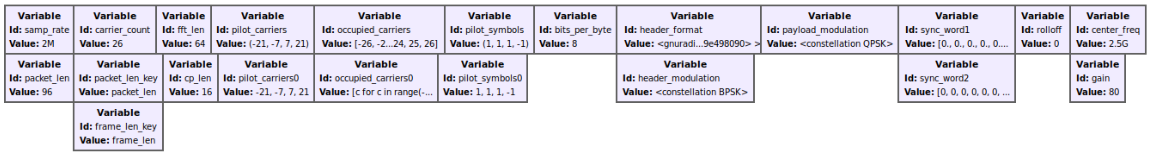
- In Binary Phase Shift Keying (**BPSK**): $I(t) \in \{-1, 1\}$ while $Q(t) = 0$
- In Quad Phase Shift Keying (**QPSK**): $I(t), Q(t) \in \{-1/\sqrt{2}, 1/\sqrt{2}\}$
- In 4-Quadrature Amplitude Modulation (**4-QAM**): $I(t), Q(t) \in \{-1, 1\}$
- In 16-Quadrature Amplitude Modulation (**16-QAM**): $I(t), Q(t) \in \{-3, -1, 1, 3\}$

3 Implementation

GNU Radio Companion flow-graph implementation is consisted of two flow-graphs: **OFDM Transmitter** and **OFDM Receiver**.

3.1 Variables

3.1.1 Common Variables



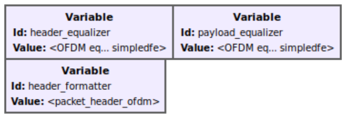
```
Variable
Id: samp_rate
Value: 2M

Variable
Id: carrier_count
Value: 26

Variable
Id: fft_len
Value: 64

Variable
Id: pilot_carriers
Value: (-21, -7, 7, 21)

Variable
Id: occupied_carriers
Value: [-26, -2...24, 25, 26]

Variable
Id: pilot_symbols
Value: (1, 1, 1, -1)

Variable
Id: bits_per_byte
Value: 8

Variable
Id: header_format
Value: <gnuradi...9e498090> >

Variable
Id: payload_modulation
Value: <constellation QPSK>

Variable
Id: sync_word1
Value: [0., 0., 0., 0., 0....

Variable
Id: rolloff
Value: 0

Variable
Id: center_freq
Value: 2.5G

Variable
Id: packet_len
Value: 96

Variable
Id: packet_len_key
Value: packet_len

Variable
Id: cp_len
Value: 16

Variable
Id: pilot_carriers0
Value: -21, -7, 7, 21

Variable
Id: occupied_carriers0
Value: [c for c in range(-...

Variable
Id: pilot_symbols0
Value: 1, 1, 1, -1

Variable
Id: header_modulation
Value: <constellation BPSK>

Variable
Id: sync_word2
Value: [0, 0, 0, 0, 0, 0, ...

Variable
Id: gain
Value: 80

Variable
Id: frame_len_key
Value: frame_len
```

Figure 5 . Common variables both in OFDM Transmitter and Receiver

samp_rate	Bandwidth
packet_len	How many bytes are packed into a packet
packet/frame_len_key	Key for packet/frame_len variables
carrier_count	Used for determining the occupied sub-carriers
fft_len	FFT size and also the maximum width of the OFDM symbols
cp_len	Cyclic-Prefix size
pilot_carriers	Pilot carriers for synchronizing
occupied_carriers	Occupied carriers for data symbols
pilot_symbols	Pilot symbols for channel state estimation
header_format	Packet header format
header_modulation	Packet header modulation scheme
payload_modulation	Packet payload modulation scheme
sync_words	Synchronization words for pilot carriers
rolloff	Roll-off flank length in samples
center_freq	Center frequency of carrier signal
gain	TX/RX antenna gain

Table 1. Common variables and what they are used for both in transmission and reception

3.1.2 OFDM Receiver Variables



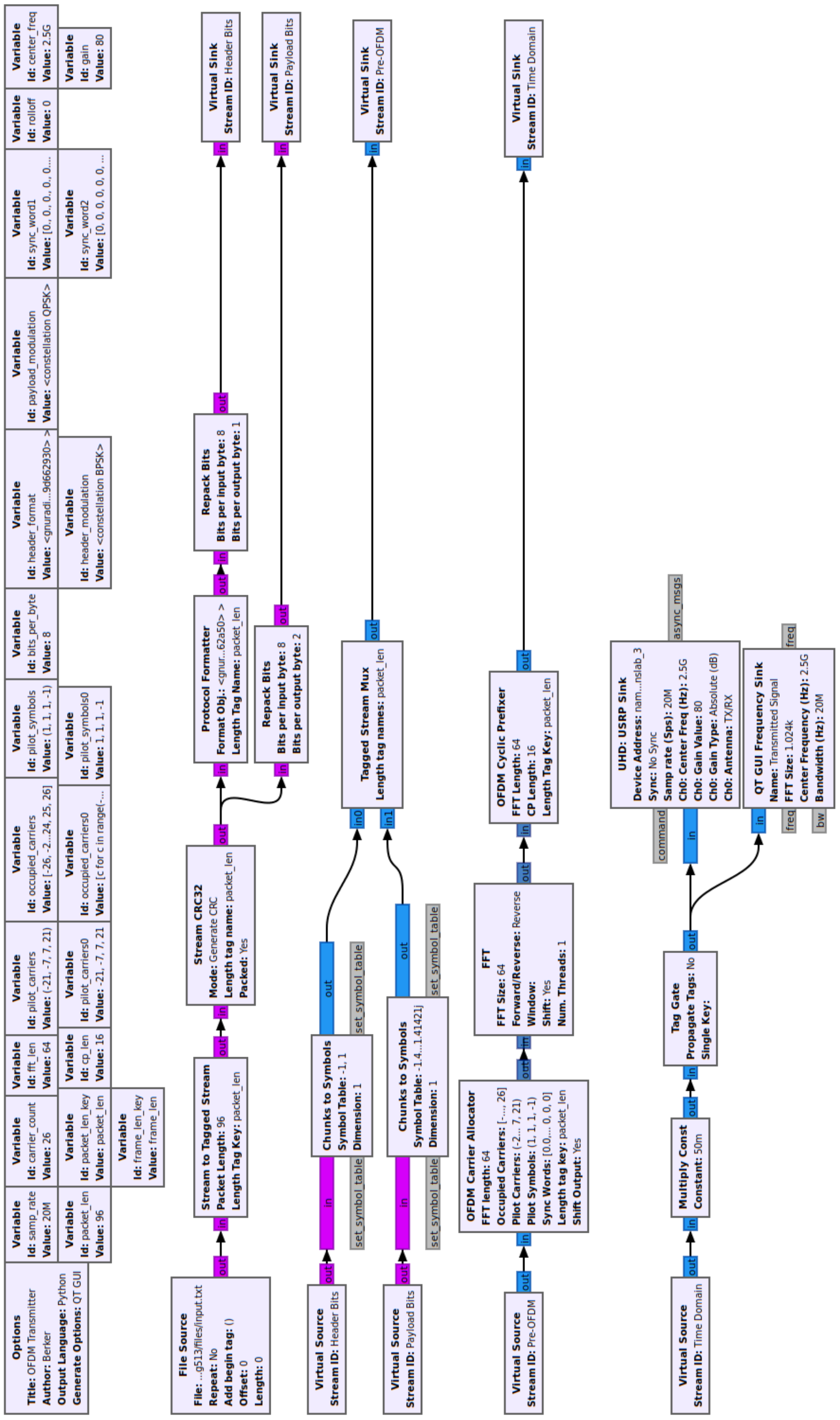
```
Variable
Id: header_equalizer
Value: <OFDM eq... simpledfe>

Variable
Id: payload_equalizer
Value: <OFDM eq... simpledfe>

Variable
Id: header_formatter
Value: <packet_header_ofdm>
```

Figure 6 . Variables in OFDM Receiver only

header_equalizer	Received header packet equalizer
header_formatter	Received header packet formatter
payload_equalizer	Received payload packet equalizer

Figure 7 . Variables in OFDM Receiver only and what they are used for in reception

3.2 OFDM Transmitter


Options Title: OFDM Transmitter; Author: Berker; Output Language: Python; Generate Options: QT GUI

Variable Id: samp_rate; Value: 20M
Variable Id: carrier_count; Value: 26
Variable Id: fft_len; Value: 64
Variable Id: pilot_carriers; Value: (-21, -7, 7, 21)
Variable Id: occupied_carriers; Value: [-26, -2...24, 25, 26]
Variable Id: pilot_symbols; Value: (1, 1, 1, -1)
Variable Id: bits_per_byte; Value: 8
Variable Id: header_format; Value: <gnuradi...9d662930> >
Variable Id: payload_modulation; Value: <constellation QPSK>
Variable Id: sync_word1; Value: [0., 0., 0., 0., 0....
Variable Id: rolloff; Value: 0
Variable Id: center_freq; Value: 2.5G
Variable Id: packet_len; Value: 96
Variable Id: packet_len_key; Value: packet_len
Variable Id: cp_len; Value: 16
Variable Id: pilot_carriers0; Value: -21, -7, 7, 21
Variable Id: occupied_carriers0; Value: [c for c in range(-...
Variable Id: pilot_symbols0; Value: 1, 1, 1, -1
Variable Id: header_modulation; Value: <constellation BPSK>
Variable Id: sync_word2; Value: [0, 0, 0, 0, 0, 0, ...
Variable Id: gain; Value: 80
Variable Id: frame_len_key; Value: frame_len

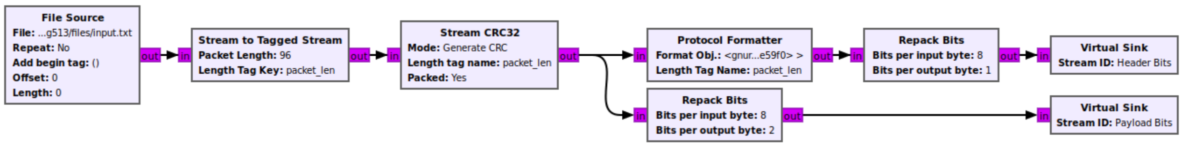
File Source File: ...g513/files/input.txt; Repeat: No; Add begin tag: (); Offset: 0; Length: 0
Stream to Tagged Stream Packet Length: 96; Length Tag Key: packet_len
Stream CRC32 Mode: Generate CRC; Length tag name: packet_len; Packed: Yes
Protocol Formatter Format Obj.: <gnur...62a50> >; Length Tag Name: packet_len
Repack Bits Bits per input byte: 8; Bits per output byte: 1
Virtual Sink Stream ID: Header Bits
Repack Bits Bits per input byte: 8; Bits per output byte: 2
Virtual Sink Stream ID: Payload Bits

Virtual Source Stream ID: Header Bits
Chunks to Symbols Symbol Table: -1, 1; Dimension: 1
Virtual Source Stream ID: Payload Bits
Chunks to Symbols Symbol Table: -1.4...1.41421j; Dimension: 1
Tagged Stream Mux Length tag names: packet_len
Virtual Sink Stream ID: Pre-OFDM

Virtual Source Stream ID: Pre-OFDM
OFDM Carrier Allocator FFT length: 64; Occupied Carriers: [-..., 26]; Pilot Carriers: (-2... 7, 21); Pilot Symbols: (1, 1, 1, -1); Sync Words: [0.0,... 0, 0, 0]; Length tag key: packet_len; Shift Output: Yes
FFT FFT Size: 64; Forward/Reverse: Reverse; Window: ; Shift: Yes; Num. Threads: 1
OFDM Cyclic Prefixer FFT Length: 64; CP Length: 16; Length Tag Key: packet_len
Virtual Sink Stream ID: Time Domain

Virtual Source Stream ID: Time Domain
Multiply Const Constant: 50m
Tag Gate Propagate Tags: No; Single Key:
UHD: USRP Sink Device Address: nam...nslab_3; Sync: No Sync; Samp rate (Sps): 20M; Ch0: Center Freq (Hz): 2.5G; Ch0: Gain Value: 80; Ch0: Gain Type: Absolute (dB); Ch0: Antenna: TX/RX
QT GUI Frequency Sink Name: Transmitted Signal; FFT Size: 1.024k; Center Frequency (Hz): 2.5G; Bandwidth (Hz): 20M

Figure 8 . Complete GRC flow-graph of OFDM transmitter


File Source
File: ...g513/files/input.txt
Repeat: No
Add begin tag: ()
Offset: 0
Length: 0

Stream to Tagged Stream
Packet Length: 96
Length Tag Key: packet_len

Stream CRC32
Mode: Generate CRC
Length tag name: packet_len
Packed: Yes

Protocol Formatter
Format Obj.: <gnur...e59f0> >
Length Tag Name: packet_len

Repack Bits
Bits per input byte: 8
Bits per output byte: 1

Virtual Sink
Stream ID: Header Bits

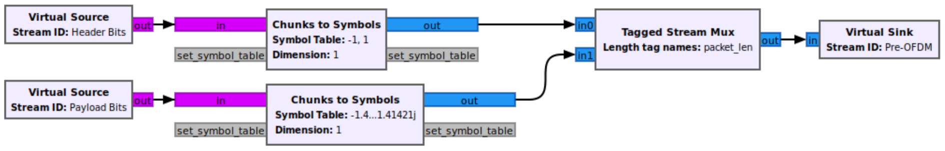
Repack Bits
Bits per input byte: 8
Bits per output byte: 2

Virtual Sink
Stream ID: Payload Bits

Figure 9 . GRC flow-graph from file bytes to header and payload bits

Complete GRC flow-graph of OFDM transmitter is shown in Fig 8 . In Fig 9 :

1. Bytes from a file is read as byte stream in File Source block.
2. Bytes from stream is put into a packet of specified length.
3. 32-bit Cyclic Redundancy Check code is generated from the packet data and inserted to the packet as suffix.
    3.1. Packet header is generated from the data and transformed again into byte stream. However, header is not directly converted into byte stream. Bits from the header is turned into bytes according to the header's modulation scheme. For example, if the header's modulation scheme is BPSK then 1-byte from the header is represented with 8-bytes.
    3.2. Bits from the payload is turned into bytes according to the payload's modulation scheme. For example, if the payload's modulation scheme is QPSK then 1-byte from the packet is represented with 4-bytes.


Virtual Source
Stream ID: Header Bits

Chunks to Symbols
Symbol Table: -1, 1
Dimension: 1

Virtual Source
Stream ID: Payload Bits

Chunks to Symbols
Symbol Table: -1.4...1.41421j
Dimension: 1

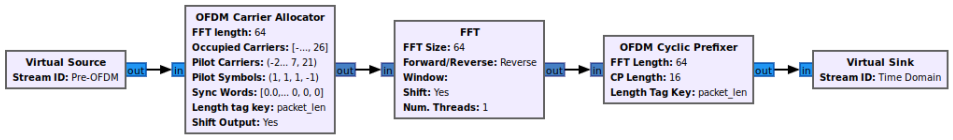
Tagged Stream Mux
Length tag names: packet_len

Virtual Sink
Stream ID: Pre-OFDM

Figure 10 . GRC flow-graph from header and payload bits to pre-OFDM symbols

In Fig 10 :

1. Byte streams for header and payload bits are translated into symbols according to their modulation schemes so byte streams are turned into complex streams.
2. Complex streams for header and payload symbols are then combined into a single complex stream by multiplexing.


Virtual Source
Stream ID: Pre-OFDM

OFDM Carrier Allocator
FFT length: 64
Occupied Carriers: [-..., 26]
Pilot Carriers: (-2... 7, 21)
Pilot Symbols: (1, 1, 1, -1)
Sync Words: [0.0,... 0, 0, 0]
Length tag key: packet_len
Shift Output: Yes

FFT
FFT Size: 64
Forward/Reverse: Reverse
Window:
Shift: Yes
Num. Threads: 1

OFDM Cyclic Prefixer
FFT Length: 64
CP Length: 16
Length Tag Key: packet_len

Virtual Sink
Stream ID: Time Domain

Figure 11 . GRC flow-graph from pre-OFDM symbols to complex stream in time domain

In Fig 11 :

1. Complex stream is then converted into vector of frequency domain OFDM symbols and these symbols are placed onto sub-carriers. Pilot symbols, synchronization words are also placed onto the carriers. In addition, this block attaches short and long preambles.

2. Inverse Fast Fourier Transform is calculated from frequency domain OFDM symbol vector.
3. Cyclic prefixes are added on OFDM symbols in order to prevent Inter-Symbol Interference (ISI) and other delays. Pulse shaping is also performed on OFDM symbols such that OFDM symbol vectors are converted into time domain complex stream.

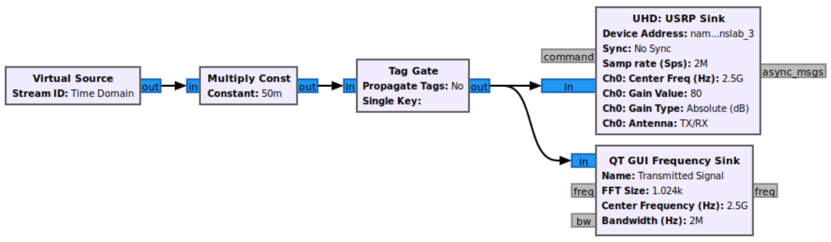


Figure 12 . GRC flow-graph from complex signal to USRP sink and frequency visualization sink

In Fig 12 :

1. Amplitude of complex stream is then attenuated in order to avoid power problems with the digital analog converters.
2. In Tag Gate block, tags are removed from the complex stream.
3. Pure complex stream is then transmitted from the USRP device.

3.3 OFDM Receiver

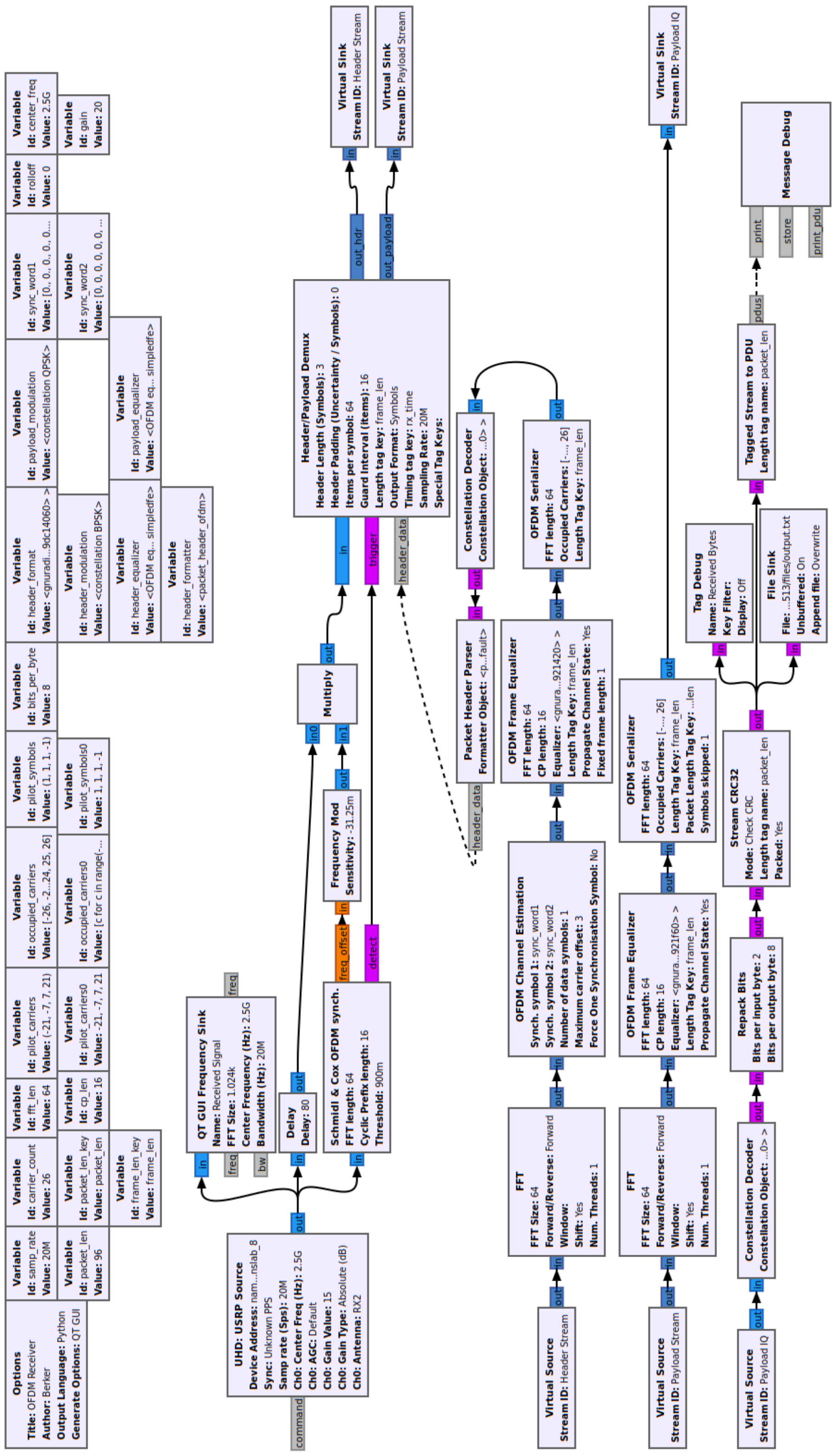


Figure 13 . Complete GRC flow-graph of OFDM receiver

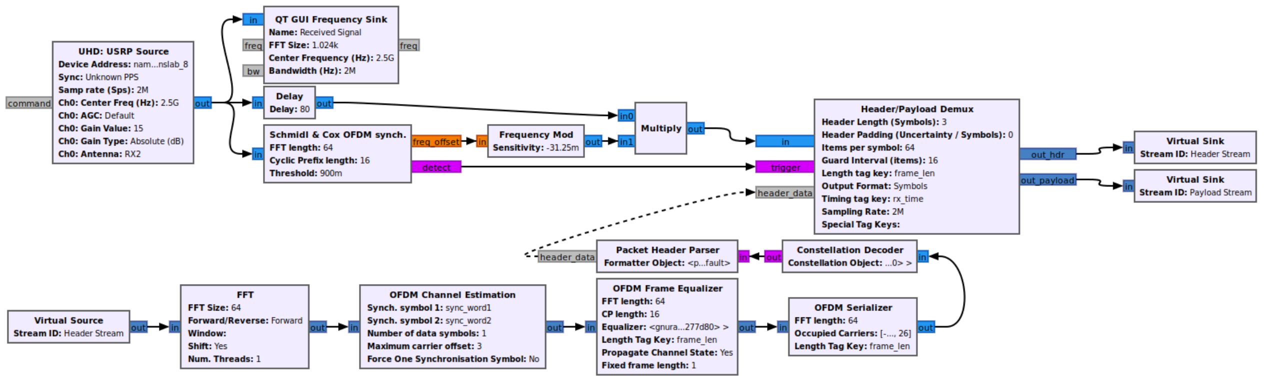


Figure 14 . GRC flow-graph from received complex signal in time domain from the USRP source to header and payload vector streams

Complete GRC flow-graph of OFDM receiver is shown in Fig 13 . Fig 14 is the most crucial part of the OFDM receiver:

1. USRP device receives the transmitted signal as a complex stream.
2. Schmidl & Cox OFDM Synchronization block restores the symbol and frequency offset of the received signal. Also, this block triggers the Header/Payload Demultiplexing block upon detecting the beginning of the frame.
3. Header/Payload Demultiplexing block simply forwards received complex stream to header stream as OFDM symbol vector upon triggered. After the header data is received, received complex stream is forwarded to payload stream as OFDM symbol vector.
4. Fast Fourier Transform is calculated from OFDM symbol vector stream.
5. OFDM Channel Estimation block estimates the channel and coarse frequency offset for OFDM from the preambles. Synchronization words are used for this task and synchronization symbols are removed the OFDM symbol vector stream.
6. OFDM Frame Equalizer block corrects coarse carrier offset and performs equalization on OFDM frame.
7. OFDM Serializer block acts as opposite to OFDM Carrier Allocator block in the transmitter. It basically serializes the complex modulations symbols from OFDM subcarriers. It transforms OFDM symbol vector stream into complex stream.
8. Complex stream's constellation is decoded according to the header's modulation scheme. Constellation Decoder block converts complex stream into byte stream.
9. Packet Header Parser block accumulates bits for header. When the header is constructed from received header bits, it sends header data (such as payload length etc.) as a feedback to the Header/Payload Demultiplexing resulting in flow in payload stream.

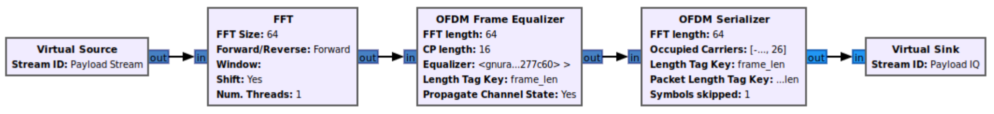
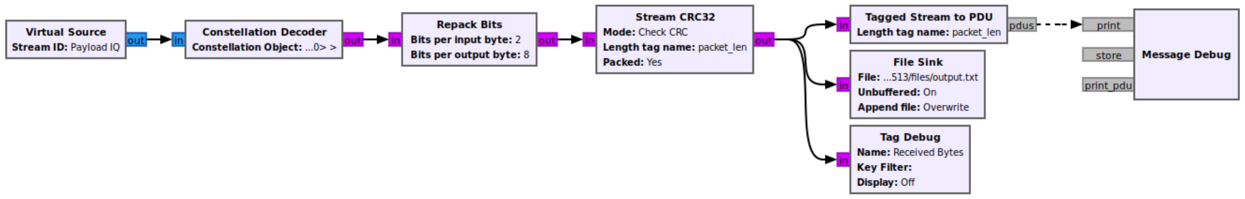


Figure 15 . GRC flow-graph from OFDM symbol vector to I/Q demodulated complex stream

In Fig 15 :

1. Fast Fourier Transform is calculated from OFDM symbol vector stream for payload stream.
2. OFDM Frame Equalizer block corrects coarse carrier offset and performs equalization on OFDM frame.
3. OFDM Serializer serializes the complex modulations symbols from OFDM sub-carriers and transforms OFDM symbol vector stream into complex stream.



Virtual Source
Stream ID: Payload IQ

Constellation Decoder
Constellation Object: ...0> >

Repack Bits
Bits per input byte: 2
Bits per output byte: 8

Stream CRC32
Mode: Check CRC
Length tag name: packet_len
Packed: Yes

Tagged Stream to PDU
Length tag name: packet_len

File Sink
File: ...513/files/output.txt
Unbuffered: On
Append file: Overwrite

Tag Debug
Name: Received Bytes
Key Filter:
Display: Off

Message Debug

Figure 16 . GRC flow-graph from complex stream to byte stream for printing, writing a file, debugging

In Fig 16 :

1. Complex stream's constellation is decoded according to the payload's modulation scheme. Constellation Decoder block converts complex stream into byte stream.
2. Payload byte stream is repacked according to the payload's modulation scheme. For example, if payload's modulation scheme is QPSK then incoming 4-bytes are repacked into 1-byte.
3. Cyclic Redundancy Check is done on the received packet. If 4-bytes of CRC code is correct, CRC code is removed from the packet and it is forwarded to the next block/s. Otherwise, packet is dropped due to bit errors.
4. Packet is forwarded to File Sink for writing into a file. For debugging, received packet contents with its tags are printed with Tag Debug and Message Debug blocks.

4 Experimental Setup

This section details the hardware and software features of the project's implementation. In this project, a real-time SDR using two USRPs connected to a personal computer running GNU Radio. One USRP is used for transmission and the other one is used for reception. Computer and hardware components are listed in Table 2 and 3.

PC Component	**Version**
OS	Ubuntu 20.04.2
UHD	3.15.0
C++	9.3.0
Boost	1.71.0
GNU Radio	3.8.1.0
Python	3.8.10

Table 2. Computer components and their versions

HW Component	**Type**
CPU	Intel Core i7-7700HQ
RAM	16 GB
SDR Transmitter	USRP B210
SDR Receiver	USRP B210
Transmitting Antenna	VERT900
Receiving Antenna	VERT900

Table 3. Hardware components and their types

Parameter	Value
Total sub-carriers	52 (48 data, 4 pilot)
Bandwidth	20 MHz
Occupied bandwidth	16.6 MHz
Sub-carrier spacing	312.5 KHz
Modulation	BPSK (Header), QPSK (Payload)
FFT length	64
CP length	16
Center Frequency	2.5 GHz

Table 4. OFDM Parameters

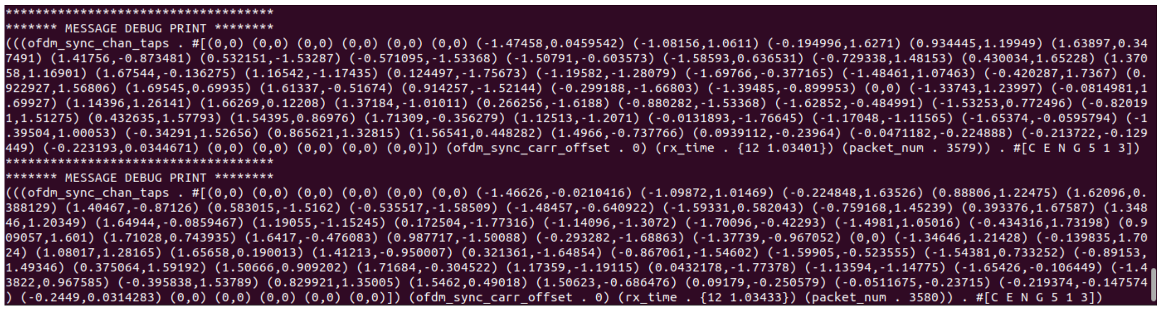
Initially, OFDM parameters from IEEE 802.11a are used to test transmission and reception of data. This parameters can be seen in Table 4. In addition to these parameters, gain values for transmitter and receiver antennas are configured manually while communicating.

5 Results and Discussion

5.1 Methodology

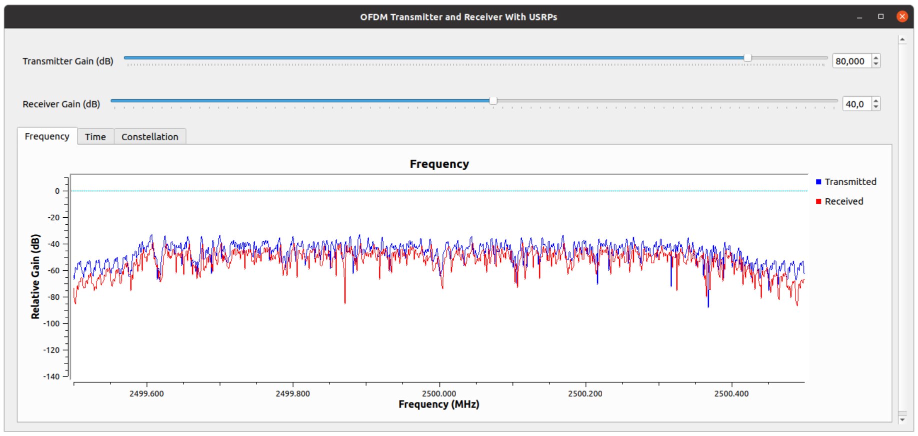
A text file is created with the ”CENG513” string. Packet length is set to 7 since the size of string ”CENG513” is 7 characters (7-bytes). File Source block’s Length field is also set to 7 as only ”CENG513” string is transmitted repeatedly. Python script for OFDM Receiver is run and then the script for transmitter is run. OFDM receiver script prints the received signal which can be seen in Fig 17 . Console output of the receiver script contains I/Q data (*ofdm_sync_chan_taps*) of received OFDM signal, carrier frequency offset (*ofdm_sync_carr_offset*), reception time, packet number and the data detected from this signal which is ”CENG513” string. Scripts visualize the transmitted and received signals in time, frequency and constellation domains and this is shown in Fig 18 . Also, . As mentioned in Subsection 4, gain values of transmitter and receiver require tuning for better communication between the USRPs. For that reason, transmitter and receiver gain values are inserted into flow-graph as run-time customizable variables. This customizable variables can be seen in Fig 18 . USRP B210 specifications denotes that:

- TX antenna’s gain range is 0.0 dB to 89.8 dB with step 0.2 dB
- RX antenna’s gain range is 0.0 dB to 76.0 dB with step 1.0 dB
- Both antennas’ frequency range is 50 MHz to 6000 MHz
- Both antennas’ bandwidth range is 200 KHz to 56 MHz



```
************************************
******* MESSAGE DEBUG PRINT ********
(((ofdm_sync_chan_taps . #[(0,0) (0,0) (0,0) (0,0) (0,0) (0,0) (-1.47458,0.0459542) (-1.08156,1.0611) (-0.194996,1.6271) (0.934445,1.19949) (1.63897,0.34
7491) (1.41756,-0.873481) (0.532151,-1.53287) (-0.571095,-1.53368) (-1.50791,-0.603573) (-1.58593,0.636531) (-0.729338,1.48153) (0.430034,1.65228) (1.370
58,1.16901) (1.67544,-0.136275) (1.16542,-1.17435) (0.124497,-1.75673) (-1.19582,-1.28079) (-1.69766,-0.377165) (-1.48461,1.07463) (-0.420287,1.7367) (0.
922927,1.56806) (1.69545,0.69935) (1.61337,-0.51674) (0.914257,-1.52144) (-0.299188,-1.66803) (-1.39485,-0.899953) (0,0) (-1.33743,1.23997) (-0.0814981,1
.69927) (1.14396,1.26141) (1.66269,0.12208) (1.37184,-1.01011) (0.266256,-1.6188) (-0.880282,-1.53368) (-1.62852,-0.484991) (-1.53253,0.772496) (-0.82019
1,1.51275) (0.432635,1.57793) (1.54395,0.86976) (1.71309,-0.356279) (1.12513,-1.2071) (-0.0131893,-1.76645) (-1.17048,-1.11565) (-1.65374,-0.0595794) (-1
.39504,1.00053) (-0.34291,1.52656) (0.865621,1.32815) (1.56541,0.448282) (1.4966,-0.737766) (0.0939112,-0.23964) (-0.0471182,-0.224888) (-0.213722,-0.129
449) (-0.223193,0.0344671) (0,0) (0,0) (0,0) (0,0) (0,0)]) (ofdm_sync_carr_offset . 0) (rx_time . {12 1.03401}) (packet_num . 3579)) . #[C E N G 5 1 3])
************************************
******* MESSAGE DEBUG PRINT ********
(((ofdm_sync_chan_taps . #[(0,0) (0,0) (0,0) (0,0) (0,0) (0,0) (-1.46626,-0.0210416) (-1.09872,1.01469) (-0.224848,1.63526) (0.88806,1.22475) (1.62096,0.
388129) (1.40467,-0.87126) (0.583015,-1.5162) (-0.535517,-1.58509) (-1.48457,-0.640922) (-1.59331,0.582043) (-0.759168,1.45239) (0.393376,1.67587) (1.348
46,1.20349) (1.64944,-0.0859467) (1.19055,-1.15245) (0.172504,-1.77316) (-1.14096,-1.3072) (-1.70096,-0.42293) (-1.4981,1.05016) (-0.434316,1.73198) (0.9
09057,1.601) (1.71028,0.743935) (1.6417,-0.476083) (0.987717,-1.50088) (-0.293282,-1.68863) (-1.37739,-0.967052) (0,0) (-1.34646,1.21428) (-0.139835,1.70
24) (1.08017,1.28165) (1.65658,0.190013) (1.41213,-0.950007) (0.321361,-1.64854) (-0.867061,-1.54602) (-1.59905,-0.523555) (-1.54381,0.733252) (-0.89153,
1.49346) (0.375064,1.59192) (1.50666,0.909202) (1.71684,-0.304522) (1.17359,-1.19115) (0.0432178,-1.77378) (-1.13594,-1.14775) (-1.65426,-0.106449) (-1.4
3822,0.967585) (-0.395838,1.53789) (0.829921,1.35005) (1.5462,0.49018) (1.50623,-0.686476) (0.09179,-0.250579) (-0.0511675,-0.23715) (-0.219374,-0.147574
) (-0.2449,0.0314283) (0,0) (0,0) (0,0) (0,0) (0,0)]) (ofdm_sync_carr_offset . 0) (rx_time . {12 1.03433}) (packet_num . 3580)) . #[C E N G 5 1 3])
```

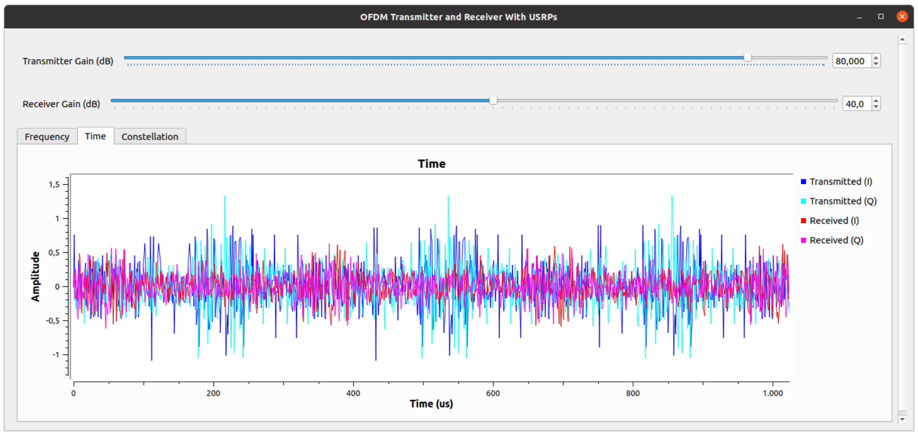
Figure 17 . Console output of OFDM receiver


OFDM Transmitter and Receiver With USRPs

Transmitter Gain (dB) 80,000

Receiver Gain (dB) 40,0

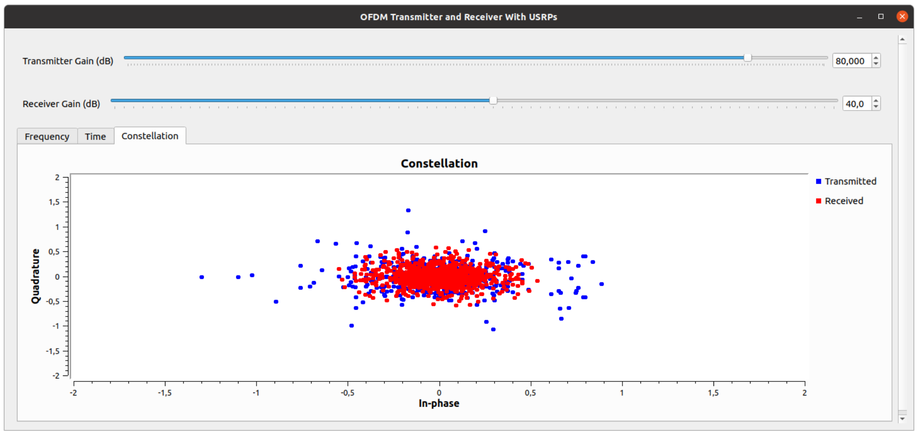
(a) Transmitted and Received Signals in Frequency Domain


OFDM Transmitter and Receiver With USRPs

Transmitter Gain (dB) 80,000

Receiver Gain (dB) 40,0

(b) Transmitted and Received Signals in Time Domain


OFDM Transmitter and Receiver With USRPs

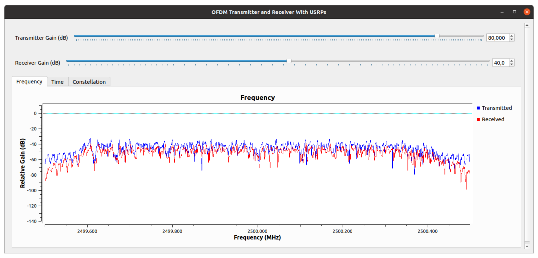
Transmitter Gain (dB) 80,000

Receiver Gain (dB) 40,0

(c) Transmitted and Received Signals in Constellation Domain

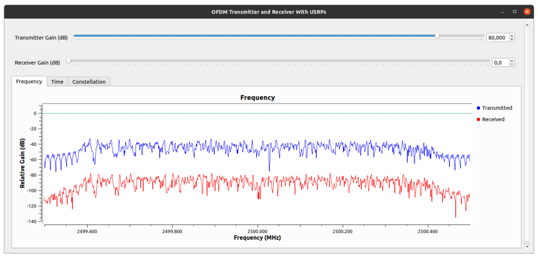
Figure 18 . Graphical interface of transmitter and receiver scripts

As shown in Fig 19 , transmitter and receiver antennas' gain values are crucial in terms of successful communication. However, transmitter antenna's gain is more effective. When transmitter's gain value is low (as in Fig 19 c and 19 d), OFDM receiver struggles with the reception.



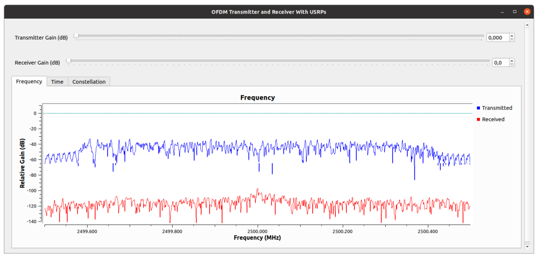
Transmitter Gain (dB) 80,000
Receiver Gain (dB) 40,0

(a) TX Gain: 80 dB, RX Gain: 40 dB



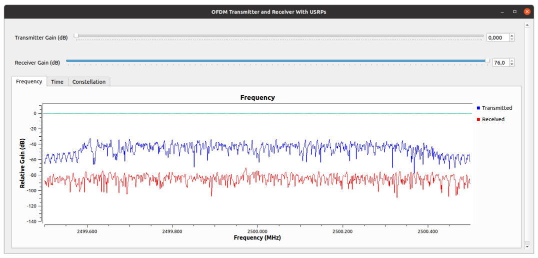
Transmitter Gain (dB) 80,000
Receiver Gain (dB) 0,0

(b) TX Gain: 80 dB, RX Gain: 0 dB



Transmitter Gain (dB) 0,000
Receiver Gain (dB) 0,0

(c) TX Gain: 0 dB, RX Gain: 0 dB



Transmitter Gain (dB) 0,000
Receiver Gain (dB) 76,0

(d) TX Gain: 0 dB, RX Gain: 76 dB

Figure 19 . Transmitted and received signals in frequency domain with different gain values

Also, a Python script is written for counting bit errors in received data. For this task, only one packet with ”CENG513” string in the payload is transmitted predetermined ($N = 10000$) times. Also, 32-bit Cyclic Redundancy Check blocks are disabled as they drop the packet in case of bit error detection. However, this script is not completely accurate for bit error calculations since some of the OFDM signals even goes undetected to the OFDM receiver due to different interference sources etc. For overcoming that problem, same packet is sent predetermined times. This bit error script provides insight about comparison of different employed parameters in OFDM transmission and reception.

I/Q samples are logged with the console output and logged file is processed with another script to create a dataset.

5.2 Results

Same packet is transmitted $N = 10000$ times for each test, and 5 tests are run for each configurations. 32-bit CRC blocks are disabled for testing purpose. At the receiver side:

- **Bit error rate** is calculated with received packets: Since the transmitted data is known, received packets' bits are compared with known packet data.
- **Missed packet count** is calculated: Since the transmitted packet count (N) is known, this can be calculated.
- **Reception time** of N times transmitted packets are calculated.

Bandwidth	Payload Modulation	Bit Error Rate	Missed Packets	Reception Time
1 MHz	BPSK	0.0	16.2	15.173640 s
2 MHz	BPSK	0.0000173873	36.4	7.573393 s
5 MHz	BPSK	0.0000235600	50.6	3.025443 s
1 MHz	QPSK	0.0000910421	31.2	8.773306 s
2 MHz	QPSK	0.0001312673	68.6	4.372114 s
5 MHz	QPSK	0.1146895491	3133.2	2.20873336 s

Table 5. Calculations of bit error rates, missed packet counts and reception times of different OFDM configurations

5.3 Discussion

Test results are shown in Table 5. When the modulation scheme gets denser or bandwidth is increased, reception times are decreasing. However, increasing the bandwidth or changing the modulation scheme to much more compact scheme increases the bit error probability and chance to miss incoming packets.

From OFDM transmitter & receiver system, and the test results, it's observed that each configuration requires extra tuning in the system in order to achieve lower bit error rate and missed packet count. For example, if the bandwidth is increased to 20 MHz from 2 MHz with configurations (such as antenna gains) specific to 2 MHz bandwidth, the OFDM receiver misses most of the frames, even it cannot detect any incoming signals.

In the testing part, different constellation schemes such as 8-PSK, 16-QAM are tried for payload modulation. However, OFDM receiver couldn't detect the incoming frames.

OFDM transmitter and receiver works well with 32-bit CRC coding but this is not enough for reliable data transmission. Since BEC (Backward Error Correction) is not implemented, dropped packets due to CRC or undetected packets cannot be restored. ARQ (Automatic Repeat Request) or H-ARQ blocks for reliable data transmission could be written in Python or C++, and then this user-blocks can be introduced into the OFDM transmitter and receiver flow-graph. FEC (Forward Error Correction) mechanism could be introduced to system for correcting erroneous packets. FEC blocks already exist in GNU Radio.

6 Conclusion

In this term project, data transmission and reception with OFDM technique is studied. For real-time data transmission and reception, OFDM transmitter and receiver system is designed in GNU Radio and it is explained in details. The system is tested with text data transmission while the receiver side calculating the bit error rate, missed packet counts and reception time. Test results show that changing system parameters such as bandwidth, modulation schemes affects the systems performance. Increasing the bandwidth results in quicker data transmission while increasing the bit error or missing packet probability. It is also observed that system needs fine-tuning for each different configuration in order to reduce bit rate error or dropped packet count. Moreover, the system is also affected by its environment so that it would be very advantageous designing dynamically self-tuning system for achieving better spectral efficiency.

OFDM transmitter and receiver system designed in the project can be further improved by introducing BEC and FEC capabilities. GNU Radio has already FEC blocks ready to use. For BEC, new blocks can be created for both transmitter and receiver to implement ARQ or H-ARQ. System's test scope can be further improved by expanding tested OFDM parameters. Tested parameters can be extended to center frequency, FFT length, CP length, allocated sub-carriers, different header and payload modulation schemes. In fact, these parameters can be changed into run-time variable which might ease the system's usage.

In this project, received OFDM signals are saved as I/Q samples for creating dataset. This dataset could be further used for channel classification task.

References

[1] X. Lin, J. Li, R. Baldemair, J.-F. T. Cheng, S. Parkvall, D. C. Larsson, H. Koorapaty, M. Frenne, S. Falahati, A. Grovlen, and K. Werner, ‘‘5g new radio: Unveiling the essentials of the next generation wireless access technology,’’ *IEEE Communications Standards Magazine*, vol. 3, no. 3, pp. 30--37, 2019.

[2] H. Liu and G. Li, *OFDM-Based Broadband Wireless Networks.* John Wiley Sons, Ltd, 2005. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/0471757195

[3] C. Anjana, S. Sundaresan, T. Zacharia, R. Gandhiraj, and K. Soman, ‘‘An experimental study on channel estimation and synchronization to reduce error rate in ofdm using gnu radio,’’ *Procedia Computer Science*, vol. 46, pp. 1056--1063, 2015, proceedings of the International Conference on Information and Communication Technologies, ICICT 2014, 3-5 December 2014 at Bolgatty Palace Island Resort, Kochi, India. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1877050915000186

[4] L. Litwin and M. Pugel, ‘‘The principles of ofdm,’’ *RF signal processing*, vol. 2, pp. 30--48, 2001.