Design and implementation of An OFDM transceiver

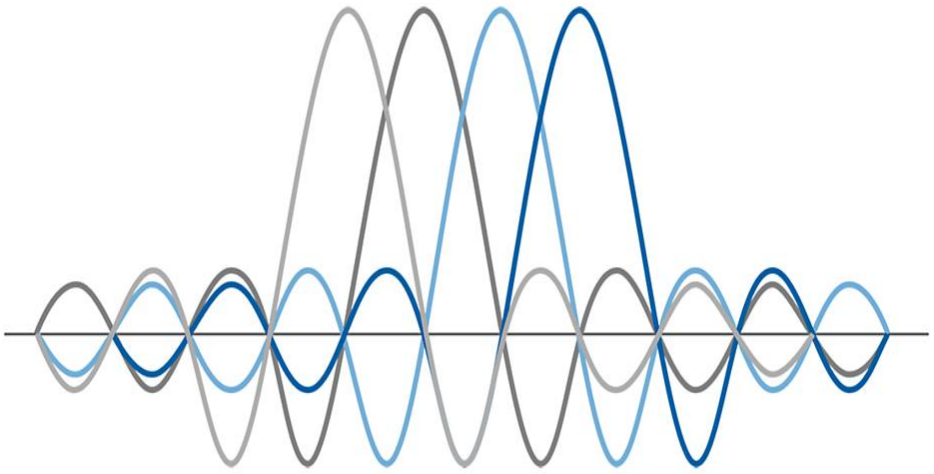
Submitted By:

Ahmed Khaled Gamal

Abdelrahman Mohamed Ibrahim

Mazen Ahmed Elaraby

Mohamed Hussein Mohamed



COMMUNICATION SYSTEMS ENGINEERING PROGRAM

Graduation Project Report

Supervisor:
Dr. Michael Ibrahim

Date:
23 June 2024

DECLARATION

We/I hereby certify that this material, which we/I now submit for assessment on the programme of study leading to the award of Bachelor of Science in communication Engineering is entirely our/my own work, that we/I have exercised reasonable care to ensure that the work is original, and does not to the best of our/my knowledge breach any law of copyright, and have not been taken from the work of others and to the extent that such work has been cited and acknowledged within the text of our/my work.

Signed: by all students

Name Ahmed Khaled Gamal	Ahmed Khaled
Name Abdelrahman Mohamed Ibrahim	Abdelrahman Bekhet
Name Mazen Ahmed Elaraby	Mazen El-Araby
Name Mohamed Hussein Mohamed	Mohamed Hussein

Date: 30 January 2024.

TASKS DISTRIBUTION

Functional Specifications from Standard and Literature Review	
General Literature review of wireless channels and OFDM	All team members
MIMO-GAN Channel modelling	Mazen El-Araby
PAPR Encoder/Decoder neural network	Ahmed Khaled
Transmitter Hardware Implementation	
Receiver Hardware Implementation	Mohamed Hussein
Synchronization Hardware Implementation	Abdelrahman Bekhet

ACKNOWLEDGMENT

We would like to express our sincere gratitude to Dr. Michael Ibrahim, our esteemed supervisor, whose guidance, expertise, and unwavering support played a pivotal role in the successful completion of this group project. Dr. Michael Ibrahim's insightful feedback, valuable suggestions, and encouragement have been instrumental throughout the research and development phases.

We also extend our appreciation to our fellow group members for their collaborative efforts and dedication. Each team member's unique contributions and commitment significantly enriched the project, creating a synergistic environment for creativity and innovation.

Furthermore, we acknowledge the resources and facilities provided by the faculty, which greatly contributed to the realization of our project goals.

ABSTRACT

Reliable high-speed data transmission in wireless environments necessitates a thorough understanding of channel impairments and their impact on communication systems. This thesis delves into the intricate world of Orthogonal Frequency Division Multiplexing (OFDM) channel modeling, performance analysis, and synchronization techniques, paving the way for robust data transmission amidst the uncertainties of wireless channels.

We establish a theoretical foundation by exploring path-loss models, large-scale and small-scale fading models, and statistical models for characterizing fading channels. Subsequently, we delve into the principles of OFDM, its advantages over single-carrier transmission, modulation techniques, guard intervals, and channel estimation methods. Utilizing Jake's model, we develop and validate accurate channel models, encompassing both flat and frequency-selective scenarios with Rayleigh and Rician processes.

Theoretical and simulated Bit Error Rate (BER) curves reveal the impact of diverse channel conditions and SNR levels on OFDM performance. We evaluate the efficacy of various coding schemes in mitigating these effects and boosting system resilience. The focus then shifts to ensuring reliable data transmission through accurate synchronization. Simulation results assess the effectiveness of different techniques in estimating and mitigating symbol time offset (STO) and carrier frequency offset (CFO) under diverse channel conditions. Packet detection techniques are also explored, facilitating the initiation of communication and decoding of received data.

To validate our findings in a real-world context, we present a hardware model designed to replicate the theoretical and simulated scenarios. Additionally, we identify promising avenues for future research, including investigating advanced communication techniques, adaptive systems capable of dynamically adjusting to changing channel conditions, and incorporating novel hardware implementations.

This thesis contributes to the ongoing pursuit of robust and reliable wireless communication, offering valuable insights into OFDM channel modeling, performance analysis, and synchronization techniques. It paves the way for further advancements in this ever-evolving field, ensuring seamless and reliable data transmission over wireless channels.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

1 INTRODUCTION

1.1 OVERVIEW

The rapid evolution of wireless communication technologies has revolutionized connectivity, from mobile devices to smart homes. Central to this advancement is the efficient transmission of data over wireless channels, which are inherently unpredictable due to fading and other impairments. Optimizing performance in such dynamic environments requires a deep understanding of channel modeling and its integration into robust communication systems.

This thesis explores the realm of MIMO (Multiple Input Multiple Output) channel modeling and its application to the physical layer of IEEE 802.11 standards. It combines floating- and fixed-point analyses with advanced hardware implementations and innovative neural network methods to enhance communication system performance. Through theoretical analysis, simulations, and practical implementations, this research aims to contribute to the development of reliable and efficient wireless communication systems.

1.2 PROBLEM STATEMENT

Wireless communication channels are highly variable, suffering from path loss, multipath propagation, and fading, which degrade signal integrity and system performance. While MIMO technology improves data throughput and reliability, modeling these channels accurately remains challenging. Traditional methods often fail to capture the complexity and variability of real-world environments, necessitating advanced modeling techniques to ensure robust performance across diverse conditions.

1.3 PROJECT DESCRIPTION

This thesis addresses these challenges by developing and validating a MIMO-GAN (Generative Adversarial Network) model for MIMO channel simulation. The project includes:

- Implementing the physical layer of IEEE 802.11 standards
- Conducting floating and fixed point analysis
- Developing and testing transmitter, receiver, and synchronization hardware on FPGA
- Implementing a neural network method for PAPR (Peak-to-Average Power Ratio) reduction in RTL
- Creating a MIMO channel simulator using WGAN-GP (Wasserstein GAN with Gradient Penalty)
- These efforts aim to provide a comprehensive solution for accurate channel modeling and performance optimization in modern wireless communication systems.

1.4 THESIS ORGANIZATION

In this thesis, the organization follows a structured approach to present the research conducted on lung fibrosis and the role of the ACE2 enzyme. The thesis begins with an Introduction (Chapter 1) that provides an overview of the research topic, outlines the problem statement, describes the project, and briefly explains the thesis organization.

Chapter 2
The Literature Review delves into existing research and developments pertinent to the study. This chapter covers the implementation of Coded OFDM in 802.11 standards, the hardware implementation of OFDM, challenges related to the Peak to Average Power Ratio (PAPR), and innovative techniques for PAPR reduction using deep learning. Additionally, it reviews MIMO channel modelling and the principles of Generative Adversarial Networks (GANs).

Chapter 3
focuses on the Project Design, detailing the design methodology and the specific processes involved in the transmitter and receiver processing chains. This chapter includes diagrams and explanations of various stages such as mapping, subcarrier allocation, IFFT, and synchronization.

Chapter 4
Titled System Modelling Results presents the theoretical and simulated BER curves, USRP block diagram verification, HDL simulation results, and synthesis reports. This chapter evaluates the performance of different system models and implementations.

Chapter 5
Introduces MIMO-GANS, providing a comprehensive overview of the proposed neural surrogate model for MIMO channels. This chapter includes a discussion on the motivation, key ideas, model architecture, dataset used, and the training and implementation details. It concludes with the results obtained from the proposed solution.

Finally, the thesis concludes with References, listing all the sources and literature reviewed and cited throughout the research. This structured organization ensures a logical flow of information, guiding the reader through the background, methodology, results, and conclusions of the study.

2 LITERATURE REVIEW

2.1 CODED OFDM IN 802.11 STANDARDS

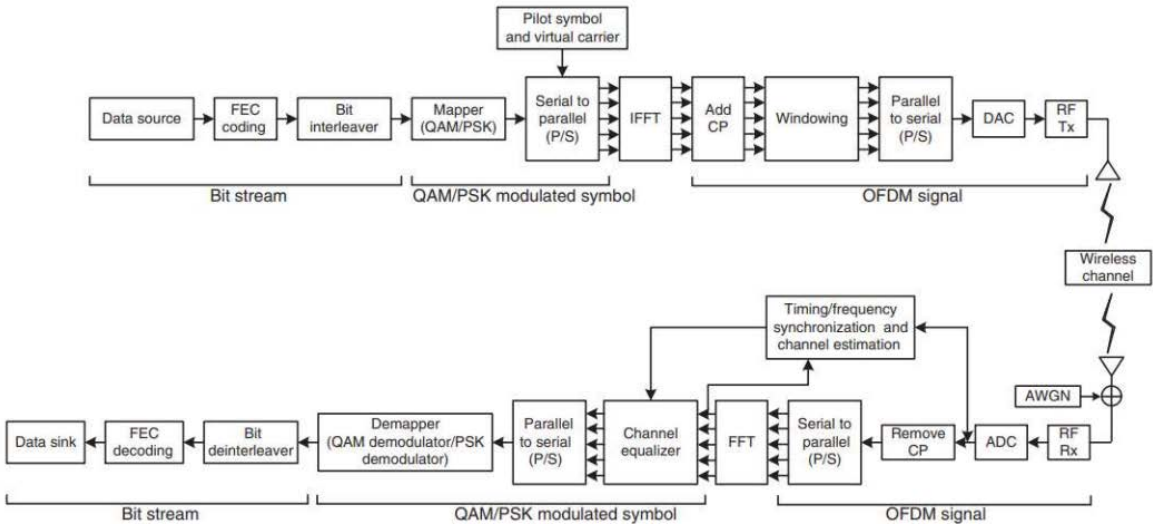
This chapter is adapted from reference [1]

Orthogonal Frequency Division Multiplexing (OFDM) is a key technology used in the physical layer (PHY layer) of various wireless communication standards, including the 802.11 series, which defines the specifications for wireless local area networking (WLAN). OFDM is employed in several 802.11 standards to enable high data rates and improve robustness in wireless communication

The OFDM system implemented is the PHY layer of a communication system it takes a Physical Service Data Unit from the MAC layer adds headers and modulates the packet to be transmitted so our system mainly serves 2 purposes:

1. Mapping the PSDU to PPDU
2. Transmitting and receive through a Wireless Medium (WM) between 2 stations (STAs) using OFDM

The PSDU is a packet received from higher layers in the system, it consists of the data we want to send over the network, this data can be a text message, in the coming sections we will go through the whole flow of how the system maps the text message to an OFDM frame and send it through the network.



2.1-1: Block Diagram of OFDM in WIFI

2.1.1 Frame Generation in 802.11 PHY layer

We first start with a PSDU packet from a higher layer that we tend to send over the OFDM system, we add headers, pilots and preamble to the data then we send it as bursts through the channel to the receiver to demodulate and detect the sent data.

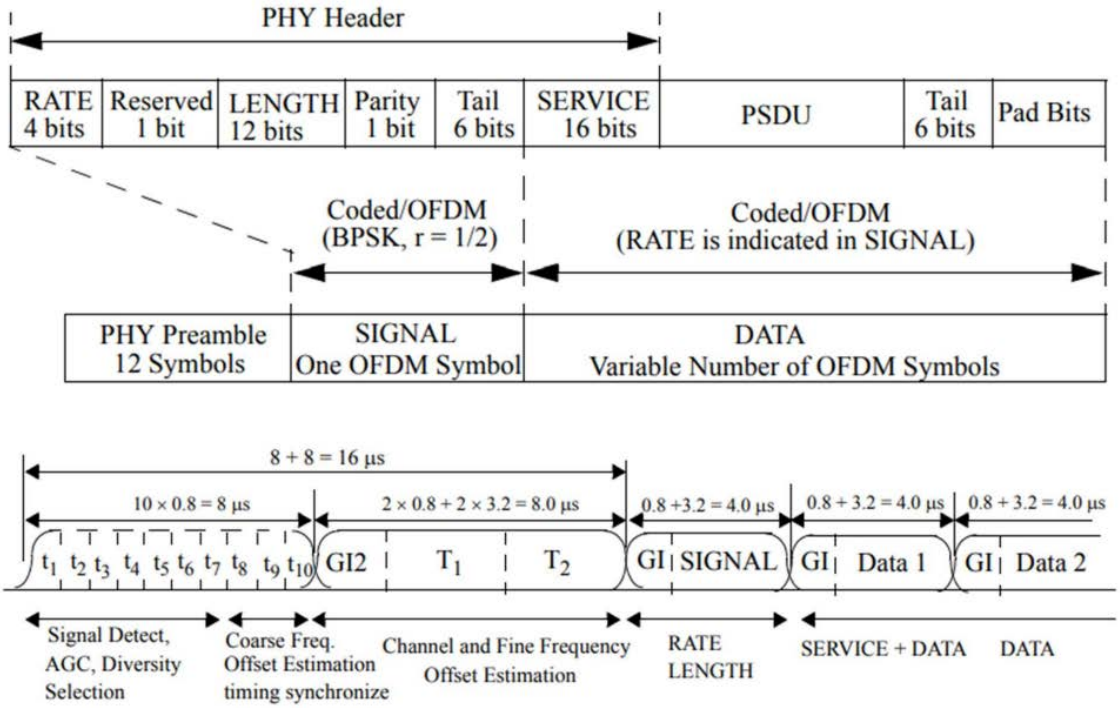


Figure 2-1: Generated OFDM frame

For different modulation schemes there are different parameters regarding the data rate, coding rate and coding bits. This was standardized from the IEEE 802.11 WIFI standard by the table below:

Table 2-1: Modulation scheme parameters

Modulation	**Coding rate (*R*)**	**Coded bits per subcarrier (N_{BPSC})**	**Coded bits per OFDM symbol (N_{CBPS})**	**Data bits per OFDM symbol (N_{DBPS})**	**Data rate (Mb/s) (20 MHz channel spacing)**	**Data rate (Mb/s) (10 MHz channel spacing)**	**Data rate (Mb/s) (5 MHz channel spacing)**
BPSK	1/2	1	48	24	6	3	1.5
BPSK	3/4	1	48	36	9	4.5	2.25
QPSK	1/2	2	96	48	12	6	3
QPSK	3/4	2	96	72	18	9	4.5
16-QAM	1/2	4	192	96	24	12	6
16-QAM	3/4	4	192	144	36	18	9
64-QAM	2/3	6	288	192	48	24	12
64-QAM	3/4	6	288	216	54	27	13.5

The Frame Generation is as follows:

i. Generating the short training sequence section of the preamble
ii. Generating the long preamble sequence section of the preamble
iii. Scrambling and zeroing the tail bits
iv. Encoding the DATA with a convolutional encoder and puncturing

v. Data interleaving
vi. Pilot insertion
vii. Mapping into complex 16-QAM symbols
viii. Transforming from frequency to time and adding a circular prefix
ix. Windowing
x. Concatenating the OFDM symbols into a single, time-domain signal

The preamble generation is mentioned separately in section 2.9 as it is related to synchronization, in this chapter we will only deal with the PSDU only

The system is defined by the coding rate, number of used subcarriers and the modulation order used the data bits to be sent is determined from the upper layers, so the PSDU size is fixed, we need to calculate the number of symbols needed to transmit this frame, this can be done using the calculations below to add PAD bits for an integer number of symbols

So, for an M-order modulation, coding rate and a number of used subcarriers (N_{used}):

```
k = log2(M)
Nbpsc = k
Ncbps = Nused * k
Ndbps = Ncbps * coding_rate
Nsym = int(ceil(data_no_pad_size/Ndbps))        #number of symbols in a frame
Ndata = Nsym * Ndbps                            #data size after padding
Npad = Ndata - data_no_pad_size                 #number of pad bits
```

The system we have will have a coding rate of 0.5, number of used subcarriers of 64, 48 used, 4 pilot, 12 guard bands, for the modulation order, we will run SER simulations for different modulations schemes with modulation orders in order to test the whole system performance.

2.1.1.1 Scrambling and Descrambling

Scrambling and descrambling are fundamental techniques used in various fields like telecommunications, data storage, and security. They are used to modify data streams in a way that makes them unintelligible without a special key or algorithm. This section aims to explain the concepts of scrambling and descrambling, their applications, and the different techniques used to implement them.

2.1.1.1.1 What is Scrambling?

Scrambling, also known as randomizing, is the process of manipulating a data stream before transmission or storage. This manipulation can involve various techniques like:

Bit inversion: Flipping individual bits in the data stream. Byte transposition: Swapping the positions of bytes within the data stream. Substitution: Replacing specific data patterns with other patterns.
XOR encryption: Applying bitwise XOR operation with a secret key to the data stream.

The main objective of scrambling is to achieve the following:

- Improve synchronization: Scrambling helps to prevent long sequences of identical bits, which can disrupt synchronization in communication systems.
- Reduce DC component: Scrambling helps to remove the DC bias from the data stream, which can be detrimental to certain transmission channels.

- Enhance security: Scrambling can provide a basic level of security by making the data unintelligible without prior knowledge of the scrambling algorithm.

2.1.1.1.2 What is Descrambling?

Descrambling is the inverse process of scrambling. It involves applying the same algorithm used for scrambling, but in reverse order, to recover the original data. The descrambler must possess the same key or secret information as the scrambler to perform this operation successfully.

2.1.1.1.3 Applications of Scrambling and Descrambling

Scrambling and descrambling find applications in various domains, including:

- Telecommunications: Scrambling is used in satellite, radio, and PSTN communication systems to improve synchronization and reduce DC component.
- Data storage: Scrambling can be used to protect sensitive data stored on magnetic or optical media.
- Digital audio and video: Scrambling is employed in cable and satellite TV to restrict access to premium content.
- Wireless networks: Scrambling is used in Wi-Fi and cellular networks to enhance security and privacy
- Techniques for Scrambling and Descrambling
- Different techniques can be used for scrambling and descrambling data, depending on the specific requirements of the application. Some common techniques include:
- Linear feedback shift registers (LFSRs): These are simple and efficient circuits that generate pseudorandom sequences used for scrambling.
- Block ciphers: These are complex algorithms that encrypt data in blocks using a secret key. Scrambling can be achieved by applying a block cipher in a specific mode.
- Stream ciphers: These are algorithms that encrypt data one bit at a time using a secret key. Stream ciphers are particularly suitable for real-time applications like voice and video communication.

2.1.1.1.4 Scrambling and descrambling in 802.11

A simple implementation of the Scrambler consists of 7 shift registers and 2 XORs as shown in Figure 5. The Scrambler is of length-127, meaning it repeatedly generates a 127-bit sequence for a given pseudo-random initial state[2]. Each incoming data bit is XORed with the current bit in the 127-bit sequence.
PPDU synchronous scrambler uses the generator polynomial S(x) as follows[3]:

$$S(x) = x^7 + x^4 + 1$$

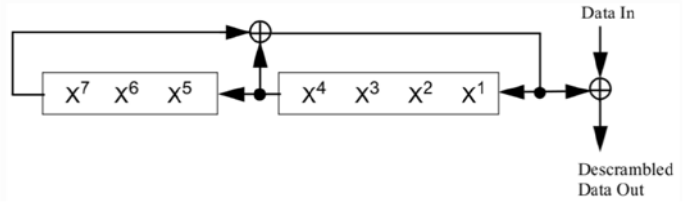


Figure 2-2: scrambler diagram

2.1.1.2 Interleaver & Deinterleaver

In wireless communication systems, errors can occur due to various factors such as channel noise, fading, and interference. These errors can corrupt the transmitted data, making it impossible for the receiver to decode the message correctly. Interleaving and deinterleaving are two techniques that are used to combat these errors and improve the performance of wireless communication systems.

2.1.1.2.1 Types of errors

To understand the functions of an interleaver/deinterleaver, understanding of error characteristics is essential. There are two types of errors that are our main concern, they are burst error and random error.

2.1.1.2.1.1 Random Errors:

Random errors, also known as independent errors or channel memoryless errors, are unpredictable and uncorrelated occurrences that can corrupt or interfere with the transmission of data in wireless communication systems. These errors occur independently of each other, meaning that the occurrence of an error at one location does not affect the probability of an error occurring at another location. This characteristic distinguishes random errors from burst errors, which tend to occur in clusters within a short time interval [4].

2.1.1.2.1.2 Burst Errors:

The term burst error means that 2 or more bits in the data unit have changed from 1 to 0 or from 0 to 1. The figure shows the effect of a burst error on a data unit. In this case, 0100010001000011 was sent, but 0101110101100011 was received. A burst error does not necessarily mean that the errors occur in consecutive bits. The length of the burst is measured from the first corrupted bit to the last corrupted bit. Some bits in between may not have been corrupted.

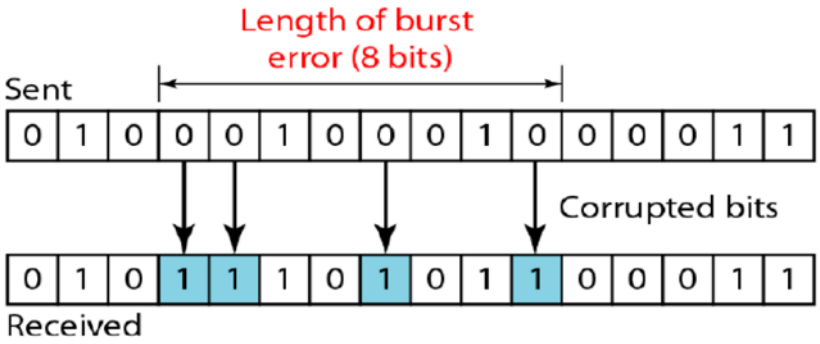


Figure 2-3: burst error illustration

Figure 2 shows the complex spectrum of a fading multipath channel, in a specific communication system, the fading, or weakening of a signal, can be quite severe. This fading can lead to the loss of an entire carrier signal. When a carrier is lost, damaged, or eliminated, the bits of data associated with that carrier are corrupted and considered errors. If these corrupted bits happen to be adjacent to each other, they form a burst error.
The purpose of an interleaver is to spread out adjacent bits across different carriers. This way, if a carrier is lost or corrupted, the affected bits are scattered, and the resulting errors are more likely to be random rather than burst. This makes it easier for the decoder to correct the errors.

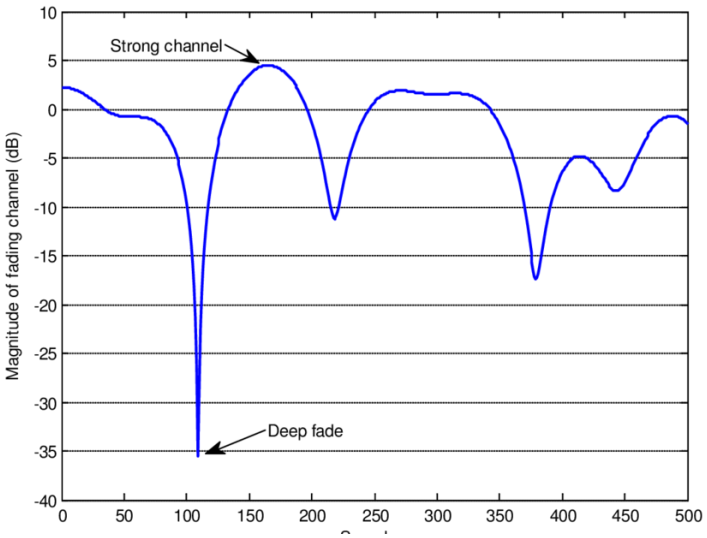


Figure 2-4: Deep fading

2.1.1.2.2 Interleaving

In digital communication systems, burst errors occur when a group of consecutive bits are corrupted due to factors such as fading, interference, or noise. These errors can significantly impact the integrity of the transmitted data and hinder the performance of the communication system. To mitigate the effects of burst errors, interleaving and deinterleaving techniques are employed.
Interleaving is a technique that intentionally spreads out adjacent bits across different carriers or time slots. This randomization process disrupts the burst error pattern, making it appear random to the decoder. By converting burst errors into random errors, interleaving allows the decoder to utilize its error correction capabilities more effectively[5].

2.1.1.2.3 Types of Interleaver

Two types of interleaves are commonly used:

- block interleaver
- convolutional interleaves.

2.1.1.2.3.1 Block Interleaver:

Block interleaver operate on a block of data, typically consisting of multiple codewords. The interleaver is filled row by row with each codeword, and the data is transmitted column by column. This process ensures that adjacent bits from the same codeword are spread out across different carriers or time slots. At the receiver, the data is interleaved before being decoded.

2.1.1.2.3.2 Convolutional Interleaver:

Convolutional interleaver process data in a continuous manner, eliminating the need for block buffering. They achieve this by maintaining a memory structure that stores the data as it is received. The interleaver output is generated by selecting data from the memory structure based on a specific pattern. Convolutional interleaver offer reduced memory requirements and eliminate the delay associated with block interleaver filling.

2.1.1.2.4 Advantages of Interleaving

Effective burst error correction: Interleaving effectively converts burst errors into random errors, improving the decoder's ability to correct errors.

Reduced decoder complexity: Interleaving allows the use of simpler decoders designed for random errors, reducing the complexity and power consumption of the receiver.

Flexibility: Interleaving can be applied to various modulation schemes and channel conditions.

2.1.1.2.5 Disadvantages of Interleaving

Delay: Interleaving introduces a delay in the transmission of data, which may be unacceptable for real-time applications.
Reduced performance for burst-error correcting codes: Interleaving can mask the structure of burst errors, potentially reducing the effectiveness of burst-error correcting codes.

2.1.1.2.6 Illustration of an Interleaver (block interleaver)

In digital communications, a block interleaver is often employed. Assume that the N is input bits to the interleaver after error correction coding. These N bits are rearranged and supplied in a predetermined order.

Consider a N = 14-bit block for demonstration purposes. The interleaver output bit index is bit inverted from the input bit index. The sequence of the input and output bit indexes is shown in figure 3. It can be seen that the interleaver output bit index is no longer continuous.
Deinterleaving is simply another bit reversal to get back to the original bit stream index.

Decimal Input Bit Index	Binary Input Bit Index	Decimal Output Bit Index	Binary Output Bit Index
0	0000	0	0000
1	0001	8	1000
2	0010	4	0100
3	0011	12	1100
4	0100	2	0010
5	0101	10	1010
6	0110	6	0110
7	0111	14	1110
8	1000	1	0001
9	1001	9	1001
10	1010	5	0101
11	1011	13	1101
12	1100	3	0011
13	1101	11	1011

Figure 2-5: Interleaving and bit index

2.1.1.2.7 Interleaver Used in the IEEE 802.11a

In the IEEE 802.11a, the interleaver is defined by a two-step permutation. The first permutation converts the input bit index k to index i according to
the following formula:

$$\boldsymbol{i = \frac{N_{cbps}}{16}(k \bmod 16) + int\frac{k}{16}} \qquad (2.1\text{-}2)$$

where **Ncbps** is the number of coded bits per symbol, **int** denotes the integral part, and **mod** is the remainder after the division.
The second permutation converts the index i to index j according to the following formula:

$$\boldsymbol{J = n\left(int\left(\frac{i}{n}\right)\right) + \left(i + N_{cbps} - int\left(\frac{16i}{N_{cbps}}\right)\right) mod(n)} \qquad (2.1\text{-}3)$$

where **n** depends upon the number of bits per subcarrier, N_{bpsc}, and is given here:

$$\boldsymbol{n = max\left(\frac{N_{bpsc}}{2}, 1\right)} \qquad (2.1\text{-}4)$$

For illustration purposes, consider BPSK used in the IEEE 802.11a, each OFDM symbol has 24 data bits and 48 coded bits after 1/2 convolutional coding[6].

2.1.1.2.8 Deinterleaver Used in the IEEE 802.11a

The deinterleaver in IEEE 802.11a is the inverse process of the interleaver described in the previous Section. It's done after two permutations[3]. The first permutation, which changes index **j** to index **i**, is given by the equation:

$$\boldsymbol{i = n\left(int\left(\frac{j}{n}\right)\right) + \left(j + int\left(\frac{16j}{N_{cbps}}\right)\right) mod(n)} \qquad \textit{(2.1-5)}$$

The second permutation which converts the index **i** back to the original index **k** and is given by the following equation:

$$\boldsymbol{k = 16i - (N_{cbps} - 1) int\left(\frac{16i}{N_{cbps}}\right)} \qquad \textit{(2.1-6)}$$

For the same example defined above, the interleaver shown converts index k = 1 to index i = 16. In the deinterleaver, the reverse process converts the index I = 16 back to index k = 1.

2.1.1.3 Forward Error Encoding and decoding

Forward Error Correction (FEC) is a technique used in communication systems to enhance the reliability of data transmission by adding redundancy to the transmitted data. The primary purpose of forward error coding is to detect and correct errors that may occur during data transmission.

The system employs an encoding technique derived from the IEEE 802.11 standard. This technique utilizes a convolutional encoder with a code rate of 0.5, implying the addition of one redundant bit for each data bit.

The generator polynomials of the encoder are:

- $S_1 = 1 + x^2 + x^3 + x^5 + x^6$
- $S_2 = 1 + x^1 + x^2 + x^3 + x^6$

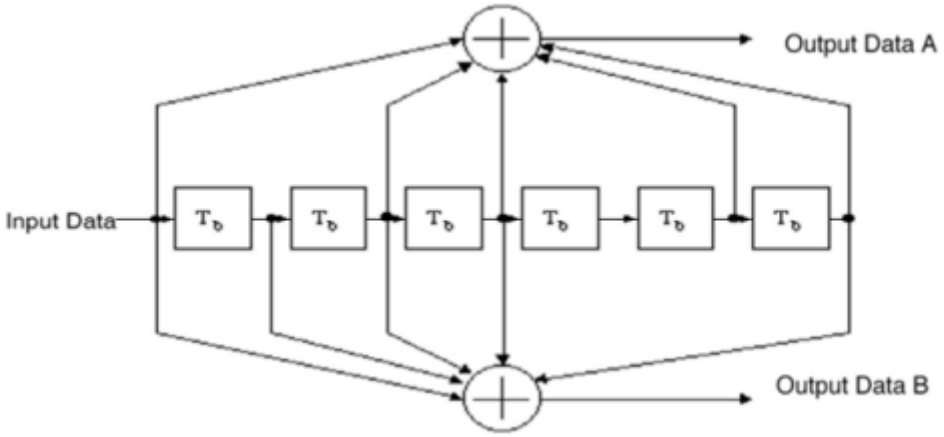


Figure 2-6: Block diagram of the convolutional encoder

The decoding process is intricately tied to the encoder type in use. For convolutional encoding, a Viterbi decoder is employed. The Viterbi decoder operates on a maximum likelihood estimation of the given codeword, mapping it to the most likely transmitted message based on error analysis. This decoder proves effective in correcting random errors. Even when punctuating (omitting) certain bits from the codeword and replacing them with zeros, the Viterbi decoder can still provide a reliable estimate of the message (details on puncturing will be discussed later). In the presence of random errors, the Viterbi decoder can consistently recover the correct data stream.

However, if the system encounters deep fades resulting in a burst of errors bit by bit, the Viterbi decoder's performance diminishes. In such cases, an alternative encoding and decoding algorithm, like Reed-Solomon, may be necessary. Reed-Solomon is adept at handling burst errors, but it operates as a block-type encoding algorithm, introducing potential overhead to our system due to its non-instantaneous (bit by bit) nature, unlike the convolutional encoder. To address this challenge, the introduction of an interleaver is required. The convolutional encoder serves to rectify random errors induced by channel effects, particularly signal fading in the time domain, including both random and burst fading. While the convolutional encoder and decoder effectively manage random errors, they are less suited to handling burst errors, a gap filled by linear block codes. This limitation underscores the necessity of an interleaver to minimize the impact of burst errors, a topic that will be explored further in subsequent sections.

2.1.1.4 Puncturing

The standard offers a specific rate for each modulation scheme, the rate is defined in the Code Rate column in the table we mentioned before. After the convolutional encoder, we have a coding rate of $\frac{1}{2}$ the standard states that we must have a $\frac{2}{3}$ coding rate for a 64-QAM for example, so we will need to remove some redundancy bits from the codeword randomly (some will be data bits too) which will increase our coding rate, the following visualization explains more what we want to do:

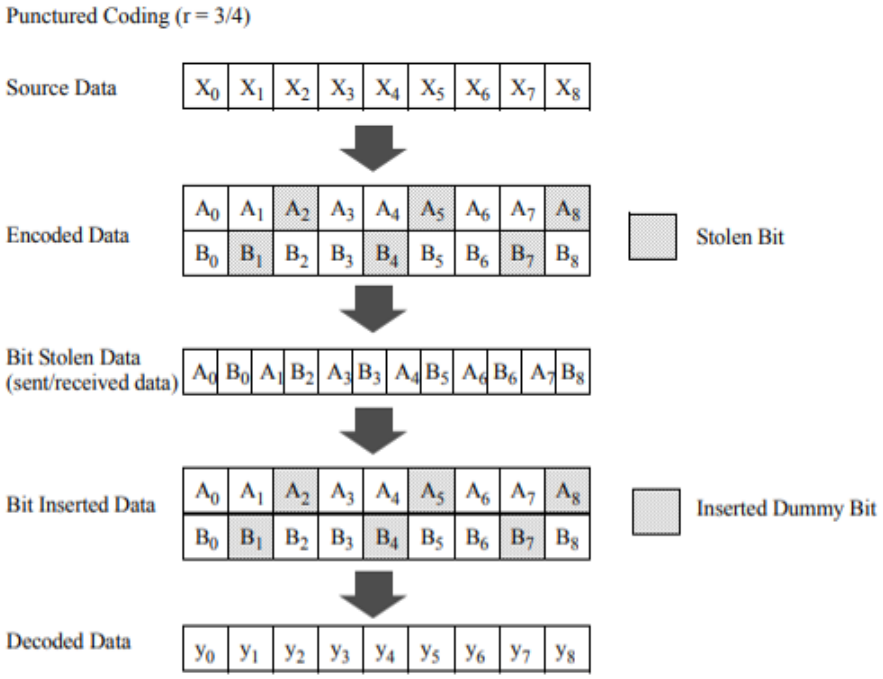


Figure 2-7: Puncturing

The following example explains how the coding rate is increased:

Table 2-2: Modulation specific parameters

Modulation	Coding rate (R)	Coded bits per subcarrier (N_{BPSC})	Coded bits per OFDM symbol (N_{CBPS})	Data bits per OFDM symbol (N_{DBPS})	Data rate (Mb/s) (20 MHz channel spacing)	Data rate (Mb/s) (10 MHz channel spacing)	Data rate (Mb/s) (5 MHz channel spacing)
64-QAM	3/4	6	288	216	54	27	13.5

In a typical OFDM system using 802.11 we have 48 subcarriers carrying the data so in total the coded bit per symbol is: $48 * 6 = 288$

After the encoder we have half of these bits representing the DATA bits, which will be 144 bits

After puncturing a whole data stream, we remove some redundancy bits and some data then we take again 288 bits and map them to an OFDM symbol, this on average gets us to 216 Data bits per OFDM symbol which is an increase to the data rate and the code rate too:

$$r_{old} = \frac{144}{288} = \frac{1}{2} \qquad R_{b,old} = \frac{144}{4us} = 36\ Mbps$$

$$r_{new} = \frac{216}{288} = \frac{3}{4} \qquad R_{b,new} = \frac{216}{4us} = 54Mbps$$

Where $4us$ is the time of the OFDM symbol

2.1.1.5 Pilot Insertion

We need not to place the same pilots for all the symbols, instead we want to send a variety of pilots to allow for a better estimation, we can achieve this by using the scrambler and a predefined pilot sequence. The pilots are real valued signals inserted to the OFDM symbol, we have 4 pilots

The contribution of the pilot subcarriers for the n^{th} OFDM symbol is produced by inverse Fourier transform of sequence P, given by

$$P_{-26,\,26} = \{0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0,$$

$$0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, -1, 0, 0, 0, 0, 0\}$$

Initiating the scrambling with a seed of $(1111111)_2$ gets us the pseudorandom sequence for the pilot arrangement with $0s$ and $1s$, replacing $1s$ with -1 and $0s$ with 1 we get the following sequence:

$$p_{0..126v} = \{1,1,1,1,\ -1,-1,-1,1,\ -1,-1,-1,-1,\ 1,1,-1,1,-1,-1,1,1,\ -1,1,1,-1,\ 1,1,1,1,\ 1,1,-1,1,$$
$$1,1,-1,1,\ 1,-1,-1,1,\ 1,1,-1,1,\ -1,-1,-1,1,\ -1,1,-1,-1,\ 1,-1,-1,1,\ 1,1,1,1,\ -1,-1,1,1,$$
$$-1,-1,1,-1,\ 1,-1,1,1,\ -1,-1,-1,1,\ 1,-1,-1,-1,\ -1,1,-1,-1,\ 1,-1,1,1,\ 1,1,-1,1,\ -1,1,-1,1,$$
$$-1,-1,-1,-1,\ -1,1,-1,1,\ 1,-1,1,-1,\ 1,1,1,-1,\ -1,1,-1,-1,\ -1,1,1,1,\ -1,-1,-1,-1,\ -1,-1,-1\}$$

Where the element P_0 multiplies the SIGNAL symbol, P_1 multiplied the first DATA symbol and so on, the preamble symbols do not need pilots as they act as pilots themselves and are used for CFO and STO

2.1.1.6 Modulation Mapping

Now we are left with a stream of bits, we need to allocate the bits to symbols and each symbol to be transmitted by subcarriers, depending on the modulation scheme used, we group the bits and form constellations (6 bits for 64-QAM, 4 bits for QPSK etc.), the constellations are them multiplied by normalizations factors to ensure the same average power to all mappings; maintain a uniform power level across different symbols to ensure efficiency of the comm channel. This normalization makes sure that each signal contributes equally to the total power of the transmitted signal:

Modulation	K_{MOD}
BPSK	1
QPSK	$1/\sqrt{2}$
16-QAM	$1/\sqrt{10}$
64-QAM	$1/\sqrt{42}$

Table 2-3: Normalization factors

We will be using the MODEM described in section 2.8.1.7 and 2.8.1.8 to perform the modulation and demodulation

2.1.1.7 Modulation

We used 3 modulation schemes: M-PAM, M-PSK and M-QAM for each we needed to make a constellation grid to map each received symbol to the respective I-Q components

2.1.1.7.1 M-PSK

The In-phase component of the M-PSK

$$I = \frac{1}{\sqrt{2}} cos\left(2\pi \frac{m}{M}\right) \quad (2.1\text{-}7)$$

The Quadrature component of the M-PSK

$$Q = \frac{1}{\sqrt{2}} sin\left(2\pi \frac{m}{M}\right) \quad (2.1\text{-}8)$$

m is symbol value and M is the modulation order

2.1.1.7.2 M-PAM

The In-phase component of the M-PAM

$$I = 2m - M + 1 \quad (2.1\text{-}9)$$

m is symbol value and M is the modulation order

2.1.1.7.3 M-QAM

The In-phase component of the M-QAM

$$I = \frac{1}{\sqrt{M}} cos\left(2\pi \frac{m}{M}\right) \quad (2.1\text{-}10)$$

The Quadrature component of the M-QAM

$$Q = \frac{1}{\sqrt{M}} sin\left(2\pi \frac{m}{M}\right) \quad (2.1\text{-}11)$$

m is symbol value and M is the modulation order

2.1.1.8 Demodulation

Demodulator is done using an IQ detector, the detector computes the pair-wise Euclidean distance of each point in the received vector against every point in the reference constellation. It then returns the symbols from the reference constellation that provide the minimum Euclidean distance.

The equation to calculate the Euclidean distance is

$$\sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2} \quad (2.1\text{-}12)$$

2.1.1.9 Frequency Domain to Time Domain (FFT)

The 6-stage radix-2 64-point FFT (Fast Fourier Transform) is a sophisticated algorithm designed to efficiently compute the discrete Fourier transform of a sequence with 64 data points. Employing a radix-2 algorithm means that the transformation process is recursively divided into stages, with each stage handling a factor of 2 points. In the context of a 64-point FFT, six stages are involved.

At each stage, the algorithm performs various operations, including butterfly operations and twiddle factor multiplications. The butterflies, which involve complex number additions and subtractions, are fundamental to the FFT algorithm and contribute to its efficiency. Twiddle factors introduce phase shifts and scale the

intermediate results. The radix-2 64-point FFT is particularly advantageous for power-of-two-sized data sets, offering a balance between computational complexity and efficiency. The algorithm leverages the inherent symmetry and periodicity properties of the FFT to optimize computations and reduce redundancy.

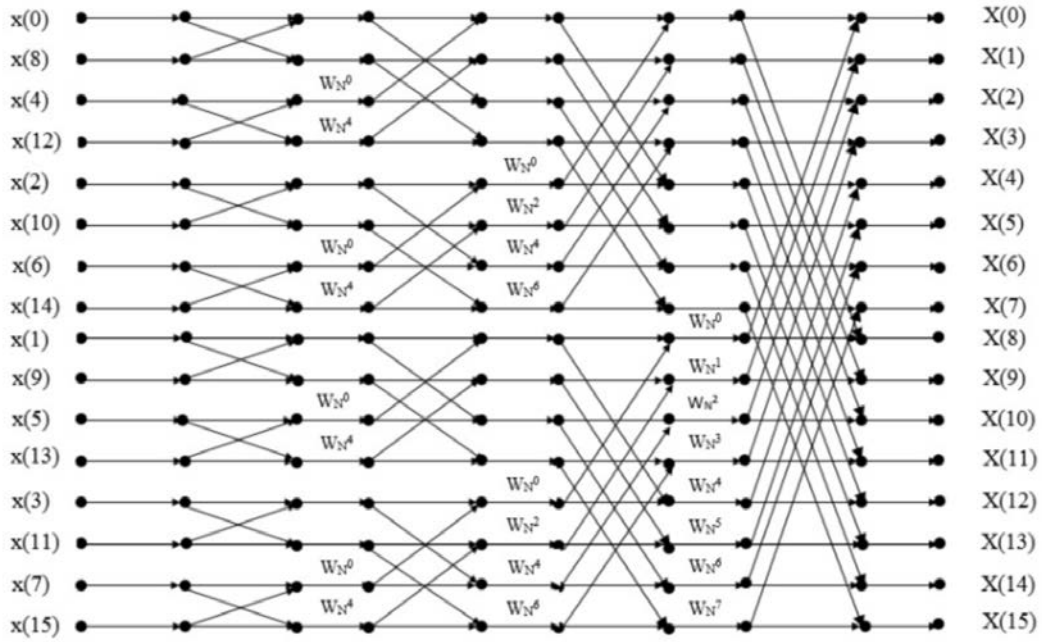


Figure 2-8 FFT radix 2 butterfly diagrams

2.1.1.10 Windowing

The time domain signal can be seen as an inverse Fourier transform multiplied by a window function in the same period, the windowing function is used to smooth the transition order to reduce the spectral sidelobes of the transmitted waveform.

After the parallel to serial conversion, we concatenate all the time domain signals after each other to form an OFDM burst, this burst is then raised on a carrier and sent through the antenna, the sudden shift in phase between each time domain signal causes spectral leakage, this can be encountered by windowing in time domain, where we insert a cyclic suffix of an OFDM symbol, and an overlap it took from the following OFDM signal cyclic prefix multiply each by windowing functions and the corresponding indices, below is a visual representation:

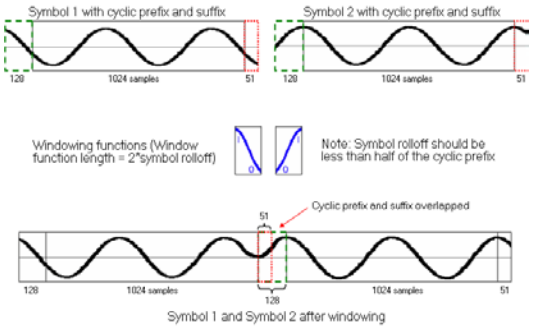


Figure 2-9 Windowing functions

The windowing function used is the Hann Windowing function:

$$w(n) = 0.5\left(1 + cos\left(\frac{\pi n}{N-1}\right)\right) \qquad (2.1\text{-}13)$$

2.2 HARWARE IMPLEMENTATION OF OFDM

This chapter is adapted from [7]

2.2.1 Fixed-point and Floating-point precision

One of the key strengths of hardware designing the system is that it is fast, power efficient and requires less area to operate if compared to using a general-purpose computer to apply the computations needed for OFDM. Fixed point representation of numbers allows the hardware to be as powerful as it is, because the size of the number or we can say bits representing the number is always determined, unlike the floating-point arithmetic where the number of bits to represent the number is undetermined and often left as maximum and the computer fills the bits to the required number

2.2.2 Floating-point arithmetic

Floating points are what we use in our daily computers. The numbers, be it decimal or integers, are assigned a space in the memory of 32 or 64 bits and the number is allowed to fit inside those numbers of bits, the number being represented is often much smaller than the assigned size so more memory is used than needed. The number being represented in those number of bits, computations are also being done of all those bits at a time, making the computations take more time and draw more power than needed. The only advantage in using a floating-point representation is that the numbers can allow a higher dynamic range, in decimals specially where in some applications like DSPs, we need this accuracy to perform operations like FFT and equalization, but as we said, using floating points are much less efficient in dealing with computations

2.2.3 Fixed-point arithmetic

In fixed points, we define a bit size for every block in the system according to its need to reduce the overall memory usage, an example of that can be the modulator, where a 4 QAM modulator outputs 4 numbers (0, 1, 2, 3) we only need 2 bits to represent all these numbers.

The process of defining each bit size for every block is called Quantization, from its name it suggests giving levels to each sample of data, another example which is more commonly used is the FFT, the transform needs I Q samples to function, the samples are often decimals and are of a high dynamic range as of the twiddle factors inside the transform, the twiddle factors are simply exponentials, so to quantize them we need to fit them inside a number of bits, which will reduce their precision which we will come back to that later. The transform as we said contains I Q samples, so we will split them into 2 separate arrays and deal with them as separate numbers. A Quantizing function will be needed to fit the numbers inside the specified range for example:

The number 5.6589 is a floating pint number if we are to fit them into 3 bits it will be 5.625, 6 bits will be 5.65625 and 10 bits will be 5.658203125, so the number eventually converges to the original as we increase the number of bits, but we need infinite number of bits to represent a number to its full accuracy

This precision difference is what we call quantization noise, and we calculate the deviation from the original number by using SQNR (Signal to Quantization Noise Ratio), this metric is used to measure the overall accuracy of a certain block, where if this metric tends to infinity if the number of the fixed point matches the floating point.

Let’s take the FFT as an example to calculate the SQNR of it as we change the number of decimal points to represent the number, the FFT used here is 64 point FFT, so it consists of 6 stages, we can quantize each stage individually to have a specific number of bits, and we can quantize the input and twiddle factors as well, but in the example we quantize the stages and input only, the following matrix shows the stages on the X axis and the number of bits used to represent each stage of the Y axis, we go from 0 bits as decimal to 20 bits for decimal while keeping the integer part as is

	0	1	2	3	4	5
0	-4.83271	-4.8085	-5.54944	-4.94866	-5.00222	-5.38292
1	0.507093	1.75944	0.863521	1.27873	1.42532	1.1769
2	7.077	7.71285	8.50511	8.32478	8.5612	7.95176
3	12.0438	12.9378	12.4204	12.4636	12.7992	12.5019
4	17.1853	17.3224	17.5034	17.3733	17.5059	17.4189
5	23.6895	23.817	23.9901	23.9391	24.4145	24.0474
6	28.3621	28.021	28.7312	28.9079	29.195	29.0325
7	37.0523	38.1167	37.8316	37.7758	38.0422	37.8899
8	41.4862	42.7202	42.377	42.3815	42.6491	42.3674
9	46.9188	47.0822	47.4835	47.1762	47.5399	47.4593
10	53.1452	53.5441	53.8825	54.0731	54.0859	53.9219
11	60.1122	60.5907	60.6618	60.3078	60.6427	60.3503
12	65.3129	65.1343	65.8928	65.7489	65.9216	65.4545
13	73.4462	73.5053	74.5267	74.6354	75.1257	74.5343
14	79.3032	80.3856	80.2956	80.5127	80.8942	80.5283
15	85.4105	87.7382	87.3445	88.3182	88.5914	87.7163
16	90.3308	91.7987	92.3921	93.132	93.292	93.0273
17	96.1189	97.3193	97.4211	97.9487	97.6707	97.3616
18	103.171	104.33	103.87	103.911	104.426	103.943
19	109.112	108.905	109.693	109.71	110.135	109.553

Figure 2-10 FFT SQNR per stage

As we can see here, we tend to get a higher SQNR as we increase the number of bits per stage, this contributes to the overall accuracy of the system, but more bits means more computation, the optimum SQNR for any system is around 45 dB so we must have at least 9 bits to represent the numbers inside the FFT to get this precision, we can select different bits to quantize each stage but we must then align the points between each stage and the other, so for simplicity we take all stages to have the same number of bits so the point is fixed

The fixed point will reduce the accuracy and dynamic range of the system, but we can tolerate some noise to have the overall computational efficiency and speed of the hardware design.

2.2.3.1 Synchronization

Packet detection, carrier frequency offset estimation, carrier frequency offset correction and symbol timing detection are essential parts in the receiver that acts as an alarm for the rest of the receiver chain to start processing the received samples.

Briefly, this part of the receiver is responsible for:

- Detecting the existence of a packet.
- Determining the CFO between the transmitter and the receiver (which is crucial to OFDM systems as CFO existence causes inter-carrier interference).
- Correcting the CFO.
- Detecting the start of the OFDM symbol within an acceptable range.
- Removing the cyclic prefix and providing the IQ samples to the receiver chain with correct timing.

The RTL representation of synchronization module mainly consists of 3 parts:

1- Packet detection and CFO estimation.

2- CFO correction, Symbol timing detection and removing CP.
3- A reset module responsible for resetting the previous modules after providing the chosen number of OFDM frames in a packet (in our case it is 4 frames per packet, but it can be adjusted to be any number).

2.2.3.1.1 Algorithm used in packet detection

The algorithm used in the packet detection is the Schmidl & Cox synchronization technique because it is considered a preamble based, autocorrelation algorithm with low complexity and ability to estimate the CFO, also it has good performance in the multipath fading channels because it uses correlation among the samples of received signal itself and minimizes the effect of the white gaussian noise by the correlation.

We used an adjusted version of the algorithm that compares the uses the magnitude instead of magnitude squared in the comparison, also the division is replaced with multiplication and a comparator with threshold set to 0.75 to be implemented by shift registers.

Generic block diagram of Schmidle & Cox algorithm (which will be applied to the STF of 802.11a preamble):

Figure 2-11Cross-Correlation in Detection shows the generic block diagram.

$$c_n = \sum_{k=0}^{L-1} r_{n+k}\, r^*_{n+k+D}$$

$$p_n = \sum_{k=0}^{L-1} r_{n+k+D}\, r^*_{n+k+D}$$

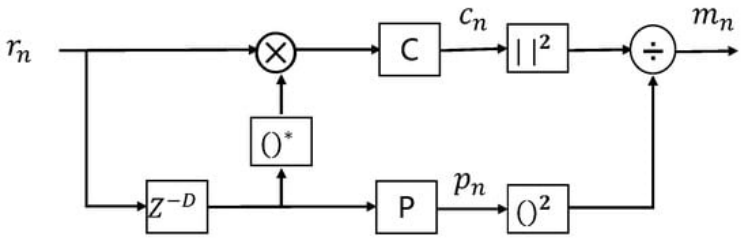


Figure 2-11Cross-Correlation in Detection

Coarse CFO can be estimated here by finding the angle of Cn parameter of packet detection.

A 64 point of Cn is used in a moving average filter in order to make the coarse CFO estimation as accurate as possible.

2.2.3.1.2 Algorithms used in Finding angles of Complex samples

It is clear that it is needed to calculate the angle of complex vector, in hardware there 3 methods to find the angle of a vector, using tan inverse:

1- Using lookup table and a divider.
2- CORDIC algorithm in rotation mode.
3- Using power series in case of existence of multipliers.

Comparing these algorithms.

2.2.3.1.2.1 Using Lookup Table and a Divider

Advantages:

- **Speed:** Once the lookup table is loaded, retrieving values and performing computations can be very fast.
- **Simplicity:** This method can be simpler to implement if the hardware supports fast division and lookup operations.

Disadvantages:

- **Memory Usage:** Requires memory to store the lookup table, which might be significant depending on the resolution needed.
- **Limited Accuracy:** The accuracy is limited by the size and the resolution of the lookup table. Higher accuracy requires a larger table, consuming more memory.

2.2.3.1.2.2 CORDIC Algorithm in Rotation Mode

Advantages:

- **Flexibility:** CORDIC is very flexible and can calculate a variety of functions (trigonometric, hyperbolic, logarithmic, etc.) beyond just the arctangent.
- **Accuracy:** It provides good accuracy that can be controlled by the number of iterations (more iterations mean higher accuracy).
- **Hardware Efficiency:** Does not require multipliers, which can simplify hardware design and reduce power consumption.

Disadvantages:

- **Convergence Speed:** The convergence speed depends on the number of iterations; more iterations slow down the computation.
- **Complexity:** More complex to implement than a simple lookup due to the iterative nature and the need for precise control over the iterations.

2.2.3.1.2.3 Using Power Series (with Multipliers)

Advantages:

- **High Accuracy:** Potentially very high accuracy if the power series is computed to many terms.
- **Scalability:** Can be scaled for different levels of precision as needed.

Disadvantages:

- **Resource Intensive:** Requires multipliers, which consume more hardware resources and power.
- **Complexity:** Implementation is more complex due to the need to manage multiple multiplication and addition operations accurately.

Summary

Table 2-4 Comparison between Different angle calculation algorithms

	LUT with Divider	CORDIC	Power Series
Advantages	• High speed • Simple	• Various functions • Accuracy • Hardware Efficient	• High Accuracy • Scalable with Accuracy
Disadvantages	• High memory usage compared to accuracy provided	• Convergence speed	• Resource intensive • Complex implementation

CORDIC is chosen in our design because of it's ability to be used in different modes and doing different functions, it will be used later on in the CFO correction by configuring it in Vectoring mode.

2.2.3.1.2.4 How does CORDIC Work?

This part is adapted from [8]

CORDIC Overview

- **CORDIC** is a collection of iterative algorithms using shifts and adds in lieu of multiplications in order to compute a number of vector rotations.
- At each step, rotate vector by a predetermined angle so as to eventually achieve a target angle.
- Larger number of steps yields a better precision, but at the cost of higher area or latency in HW.

CORDIC Cosine and Sine Calculation
The idea of finding Sine and Cosine of an angle using CORDIC is rotating a unit vector by that angle, and the resulting vector (which is still a unit vector) represents the cosine and sine of the angle represented in the X and Y components respectively.
- Applying the rotation transformation matrix on the unit vector results in the cosine and sine of the rotation angle:

$$x' = x\cos(\theta) - y\sin(\theta)$$
$$y' = y\cos(\theta) + x\sin(\theta)$$

- How can we eliminate all multiplications?

$$x' = \cos(\theta)\,(x - y\tan(\theta))$$
$$y' = \cos(\theta)\,(y + x\tan(\theta))$$

An iterative form will be:

$$x_{n+1} = \cos(\theta)\,(x_n - y_n\tan(\theta))$$
$$y_{n+1} = \cos(\theta)\,(y_n + x_n\tan(\theta))$$

We remove multiplications with tan:

$$\theta = \tan^{-1}(2^{-n})$$

So the iterative equations are now:

$$x_{n+1} = \cos(\theta)\left(x_n - y_n(2^{-n})\right)$$
$$y_{n+1} = \cos(\theta)\left(y_n + x_n(2^{-n})\right)$$

- The cosine term over a fixed number of steps is predetermined.

$$x_0 = \frac{1}{\cos(\theta_0) \cdot \cos(\theta_1) \cdots \cos(\theta_n)} \approx 0.60725 \ldots$$
$$y_0 = 0$$

- Therefore, the magnitude of the initial vector can be scaled down to compensate for the gain.

$$x_{n+1} = (x_n - y_n\, tan(\theta))$$
$$y_{n+1} = (y_n + x_n \tan(\theta))$$

- Keep track of angle to determine if a rotation should be clockwise or counterclockwise.

$$x_{n+1} = x_n - y_n(2^{-n})\text{sign}(\text{angle}_n)$$

$$y_{n+1} = y_n + x_n(2^{-n})\text{sign}(\text{angle}_n)$$

$$\text{angle}_{n+1} = \text{angle}_n - \theta_n\text{sign}(\text{angle}_n)$$

Block diagram of a single stage of number n:

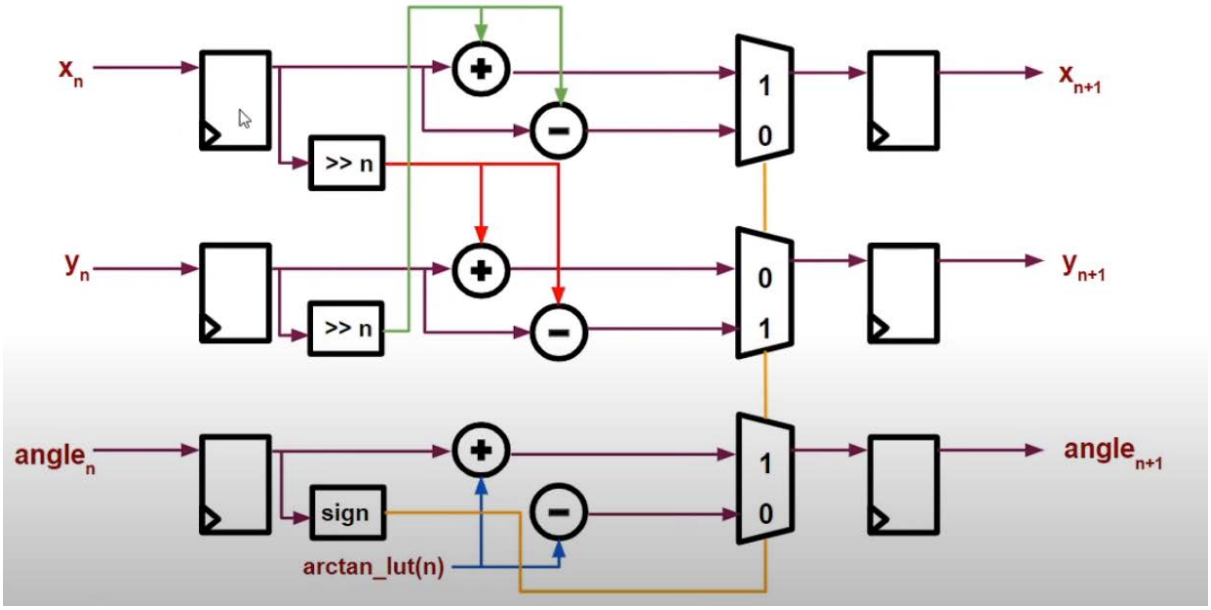


Figure 2-12 Block diagram of CORDIC single stage

The algorithm covers angles range between -pi/2 and pi/2, so any angle outside this range must be mapped into this range and then change sign according to the following:

If it is in second quadrant, map it to first quadrant, subtract pi/2 from the angle and flip sign of Cosine component.

If it is in third quadrant, map it to fourth quadrant, subtract the angle from 2*pi and flip sign of Sine component.

We used a time shared approach for the CORDIC instead of stages, this reduces the redundant hardware and replacing it with a controller.

2.2.3.1.3 Algorithm used in estimating the magnitude of a complex number

The CORDIC core can be adjusted to calculate the magnitude, but a simpler algorithm that is considered an estimation is used, it called "alpha max plus beta min".

$$M \approx \alpha \cdot \max(|I|, |Q|) + \beta \cdot \min(|I|, |Q|)$$

Optimally, alpha should be equal to 0.9604, and beta equal to 0.398 giving a maximum error of 3.96%.
These values in hardware can be approximated to 1 and 0.25 for alpha and beta respectively in hardware in order to replace the multipliers with shift registers, giving a max error of 11.61% and 3.2% mean error.

Maximum error is found around 45, 22.5 and 67.5 degrees of the vector.

Table 2-5 Performance of alpha min beta max algorithm

Alpha	Beta	Largest error (%)	Mean error (%)
1	1/2	11.80	8.68
1	1/4	11.61	3.20
1	3/8	6.80	4.25
7/8	7/16	12.50	4.91
15/16	15/32	6.25	3.08
Optimum (0.9604)	Optimum (0.398)	3.96	2.41

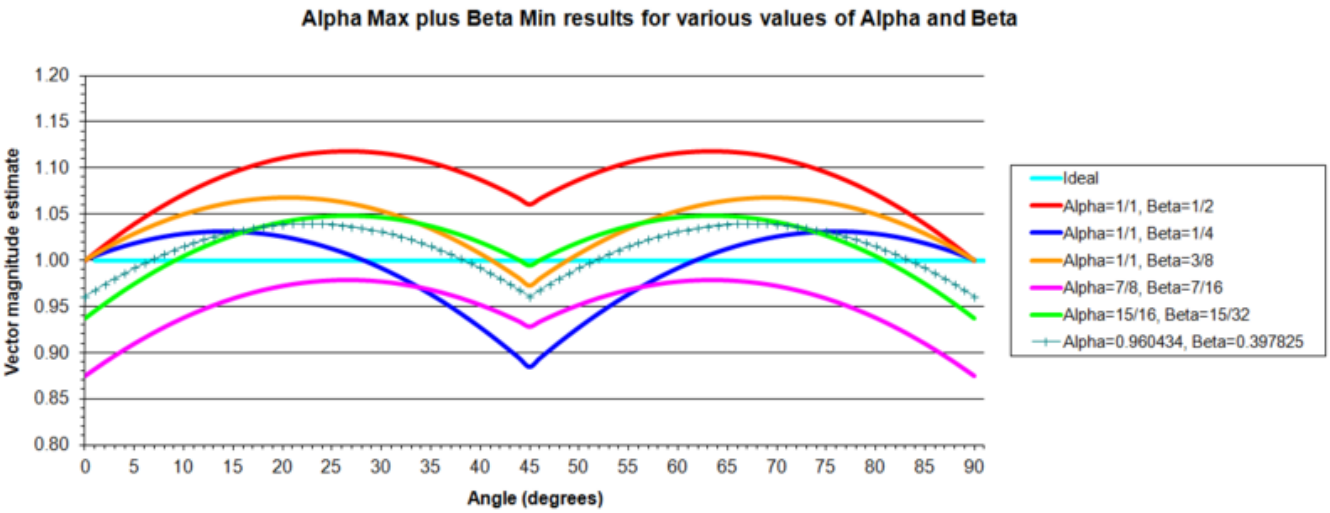


Figure 2-13 Phase of vector and corresponding error in estimation

2.2.3.1.4 Algorithm used in Symbol timing offset estimation

The algorithm used in the symbol timing is the cross correlation technique with a maximum search.

$$c_n = \sum_{k=0}^{15} r_{n+k}\, T_k^*$$

$$p_n = \sum_{k=0}^{L-1} T_{n+k}\, T_{n+k}^*$$

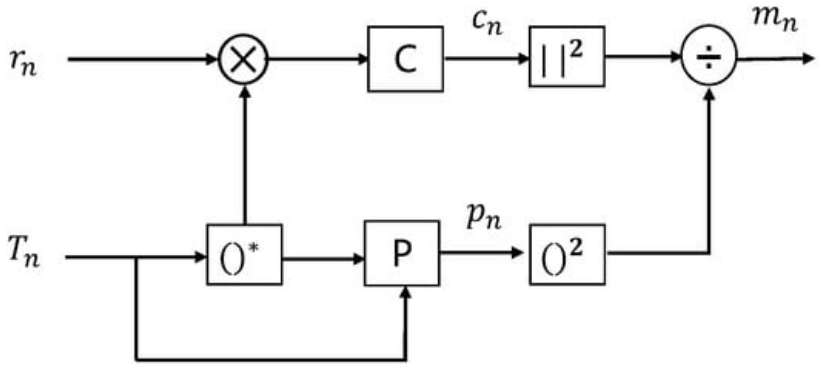


Figure 2-14 Cross-Correlation Block Diagram

2.2.3.2 Receiver processor chain

From the synchronization we get the data valid signal which states that the data entering the receiver is ready to be demodulated, the receiver chain can be described in more detail by the following block diagram:

Insert Block Diagram Picture below

In the coming sections we will go in brief detail about how to implement each block in the block diagram and which implementation we chose that fits our system

2.2.3.2.1 Data flow in the receiver

From the synchronization we get 2 kinds of data for each OFDM frame, a long preamble and the data symbols. The long preamble is used for channel estimation and equalization, while the data symbols contain the data that needs to be demodulated. The data, be it preamble or data, both enter the FFT and are transformed back into frequency domain, but the preamble is stored in memory to equalize the rest of the OFDM symbols.

As stated before, in the WIFI 802.11 standard, we have 48 data subcarriers, 4 pilot subcarriers and 12 nulls to help with the spectral efficiency and orthogonality, the data provided by the synchronization is from all the 64 Subcarriers, so we need to filter out the data subcarriers and the pilots for phase offset estimation and remove the nulls from the frame in order to deal efficiently with the data in the frame.

So, the data flow is as follows:

- Transform the data using FFT
- Filter out the nulls from the OFDM symbol
- Average out the pilots from the OFDM symbols to perform phase offset estimation and store the avg in memory
- If the data is LTS store the filtered frame in the memory
- After each OFDM symbol is filtered out, perform channel estimation on all the data
- Equalize the data again using the averaged-out pilots for phase estimation
- Demodulate the symbols and retrieve the data bits

2.2.3.2.2 Pipelined Fast Fourier Transform

This chapter is adapted from [9]

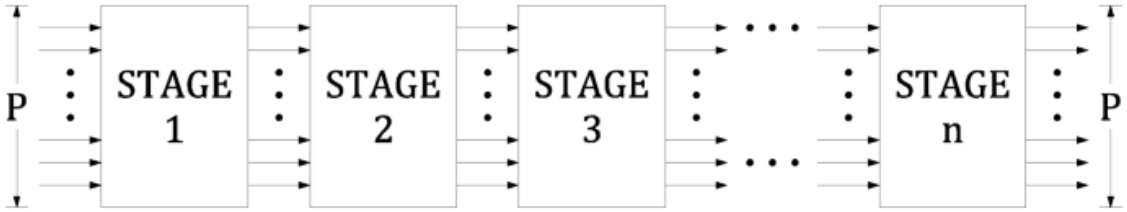


Figure 2-15 Pipelined FFT Architecture

Figure shows the general structure of a pipelined FFT. It consists of *n* stages connected in series where data flows from stage 1 towards stage *n*, and each stage *s* of the architecture calculates all the computations of one stage of the FFT algorithm. Each stage has *P* inputs and *P* outputs, and data flow in continuous flow at a rate of *P* data per clock cycle.

There are 2 types of FFT architectures

- Serial Pipelined FFT: Where only 1 input is entered at a time into the system, a delay is issued until all the suitable inputs are aligned to be processed, examples of this type is:
  - SDF: Single-path Delay Feedback

  This type of FFT has a throughput of 1 as only 1 input is output at a time and is suitable for slow systems and require less area to operate

- Parallel Pipelined FFT: Where multiple inputs are input into the system and processed at the same time, no delay is issued at the beginning as the inputs are already aligned, but delay can be issued later in the stages to align the different inputs from different stages examples of this type is:
  - MDC: Multipath Delay Commutator

  This type of FFT has a throughput >1 it depends on how many inputs are input at the time and for the same reason the same number of outputs is transformed

2.2.3.2.2.1 Singel-path Delay Feedback (SDF)

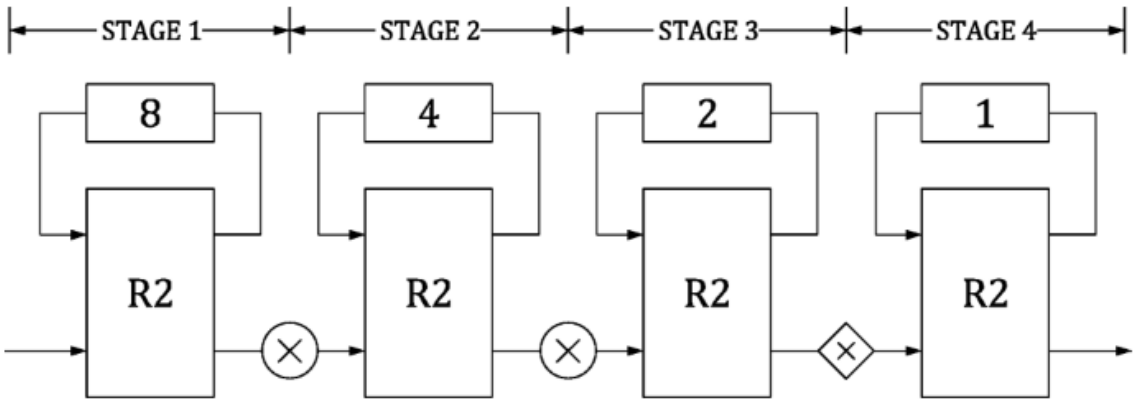


Figure 2-16 16-point radix-2 SDF FFT

Figure shows a 16-point radix-2 SDF FFT architecture. Each stage includes a radix-2 butterfly (R2), a rotator (⊗) which is a full complex multiplier or trivial rotator (diamond-shaped) that multiplies by [10 -j -j], and a buffer to align the different inputs that are input serially to be processed in the butterfly

To have a clearer sight of how the architecture is displayed we can extend this to a 4-point FFT, what we expect to happen is the butterfly to be a R4 butterfly and to have more delay elements stacked over each other to align the different inputs to the FFT and the stages to reduce by half for the same number of FFT points

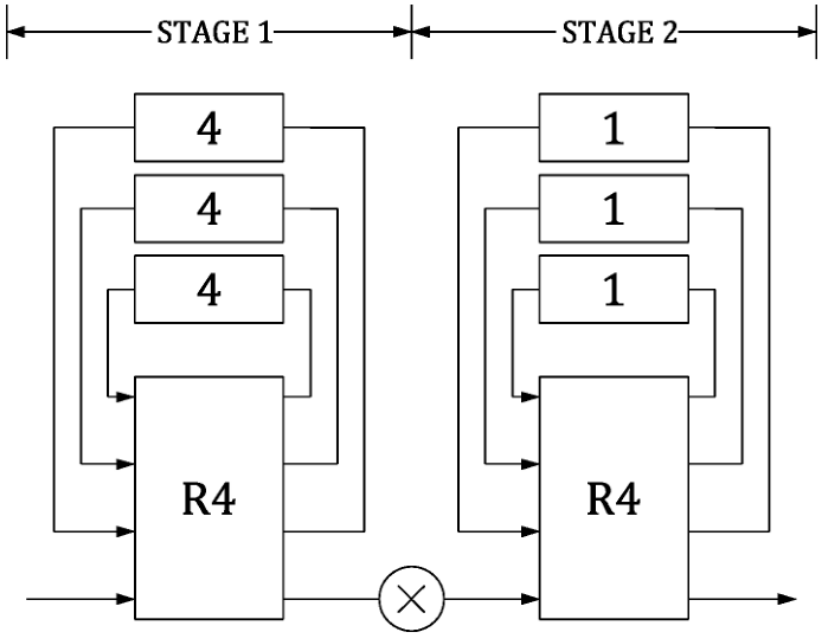


Figure 2-17 16-point radix-4 SDF FFT architecture

What we observed from the following argument is that the Radix 2 FFT butterflies are less complex than the radix 4 butterflies in the number of delay elements but the radix 4 have less number of complex multipliers, which greatly improve the synchronization between stages as we do not need now to calculate the twiddle factors to go from each stage to another, instead it is done by trivial multiplications or less twiddles are needed.

An algorithm can be made to take the best of both worlds. This algorithm is the Common Factor Algorithm (CFA). The Radix-22 Fast Fourier Transform (FFT) algorithm is an efficient method for computing the discrete Fourier transform (DFT) by leveraging a hybrid approach combining radix-4 and radix-2 techniques. This algorithm breaks down a problem of size N into smaller sub-problems through a series of stages: initially decomposing the sequence into smaller parts processed using radix-4 FFTs and then refining the results using a final radix-2 step. This method is particularly efficient for data sizes that are powers of four, such as $N = 4^k$. The Radix-22 FFT minimizes computational complexity by optimizing both the number of arithmetic operations and memory accesses. It applies a blend of four-point butterfly operations typical of radix-4 and two-point operations from radix-2, which together facilitate an effective balance between performance and simplicity.

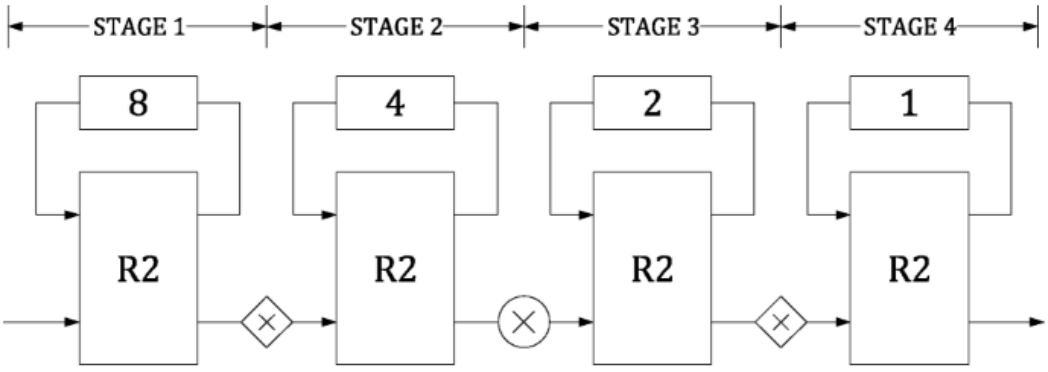


Figure 2-18 16-point radix-22 SDF FFT architecture

This figure shows an example about the Radix-22 FFT for 16 points, and the number of complex multipliers is reduced by 1 which significantly reduces the area if we use a higher point FFT.

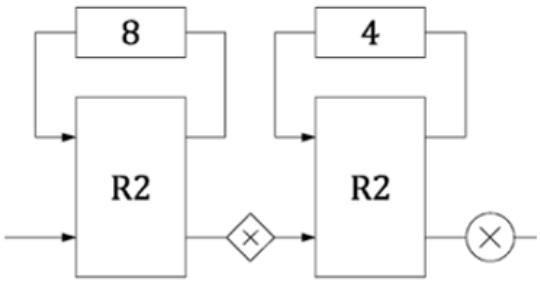


Figure 2-19 A Single SDF Unit

For having naming simplicity, we will refer to this as an SDF unit, this concept applies to higher point FFT as well, the SDF unit consists of 2 butterflies, delay buffers, a trivial and a complex multiplier, and for the SDF unit of the last stage there is no complex multiplier for the SDF unit, we only have the butterflies, delays and trivial multipliers.

Now for our system we have a 64-Point FFT, we will review the butterfly diagrams regarding Radix-2 and Radix-22 FFT:

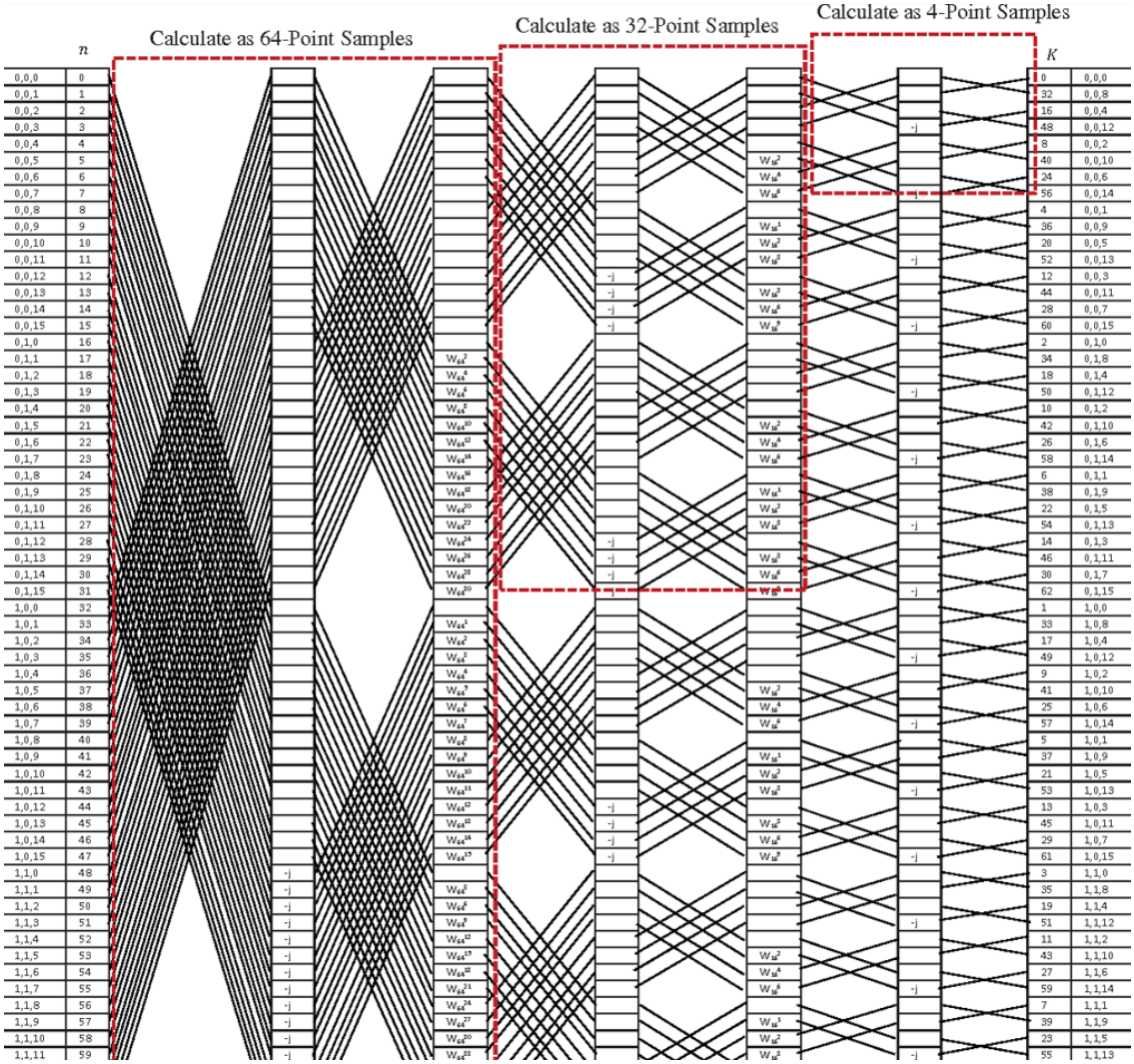


Figure 2-20 Radix-22 64-Point FFT

Using the Radix-22 Algorithm significantly reduced the complexity of the design by having 2 complex multipliers instead of 5, this reduces the area of the design, speed and addressing of the twiddle factors.

2.2.3.2.2.2 MULTI-PATH DELAY COMMUTATOR [11]

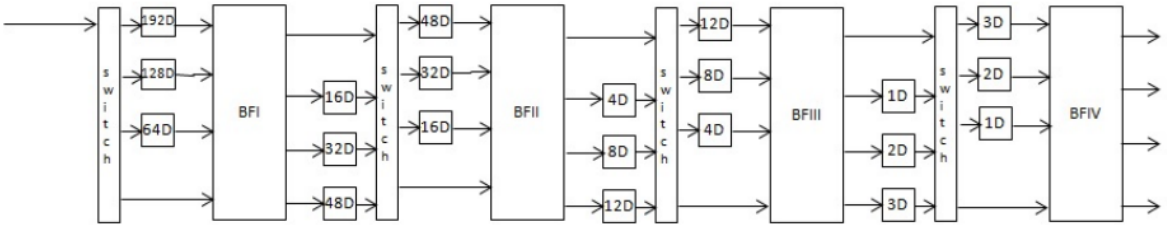


Figure 2-21 256-point 4-parallel radix-4 MDC FFT architecture

The multi-path architecture's main purpose is to provide a higher throughput than the single path, as we input more than one data sample at a time, the block diagram consists of Delay elements, butterfly and a switch, which is often referred to as a reordering element or a commutator, the commutator is used to align the outputs for the next butterfly to process. And the butterfly contains fully complex multipliers for the twiddle factor multiplications for transferring from one stage to another.

The commutator is simply a multiplexer that acts upon every clock cycle of its inputs, it reorders the inputs based on the clock cycle it is in, where every clock cycle has a reordering pattern [12]

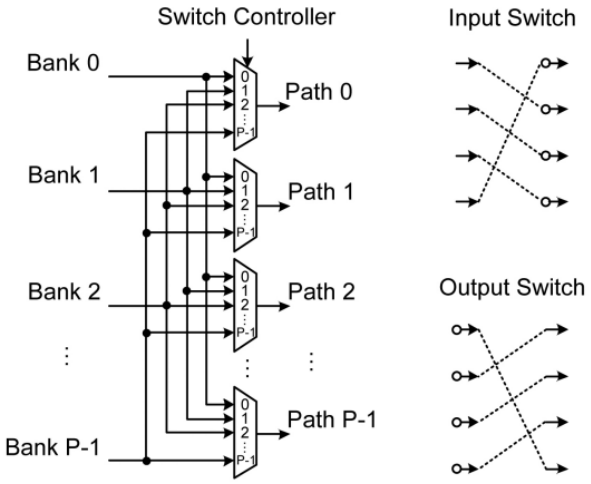


Figure 2-22 Commutator architecture between the memory banks and butterfly.

A more detailed description of the commutator is demonstrated by the diagram below [13]

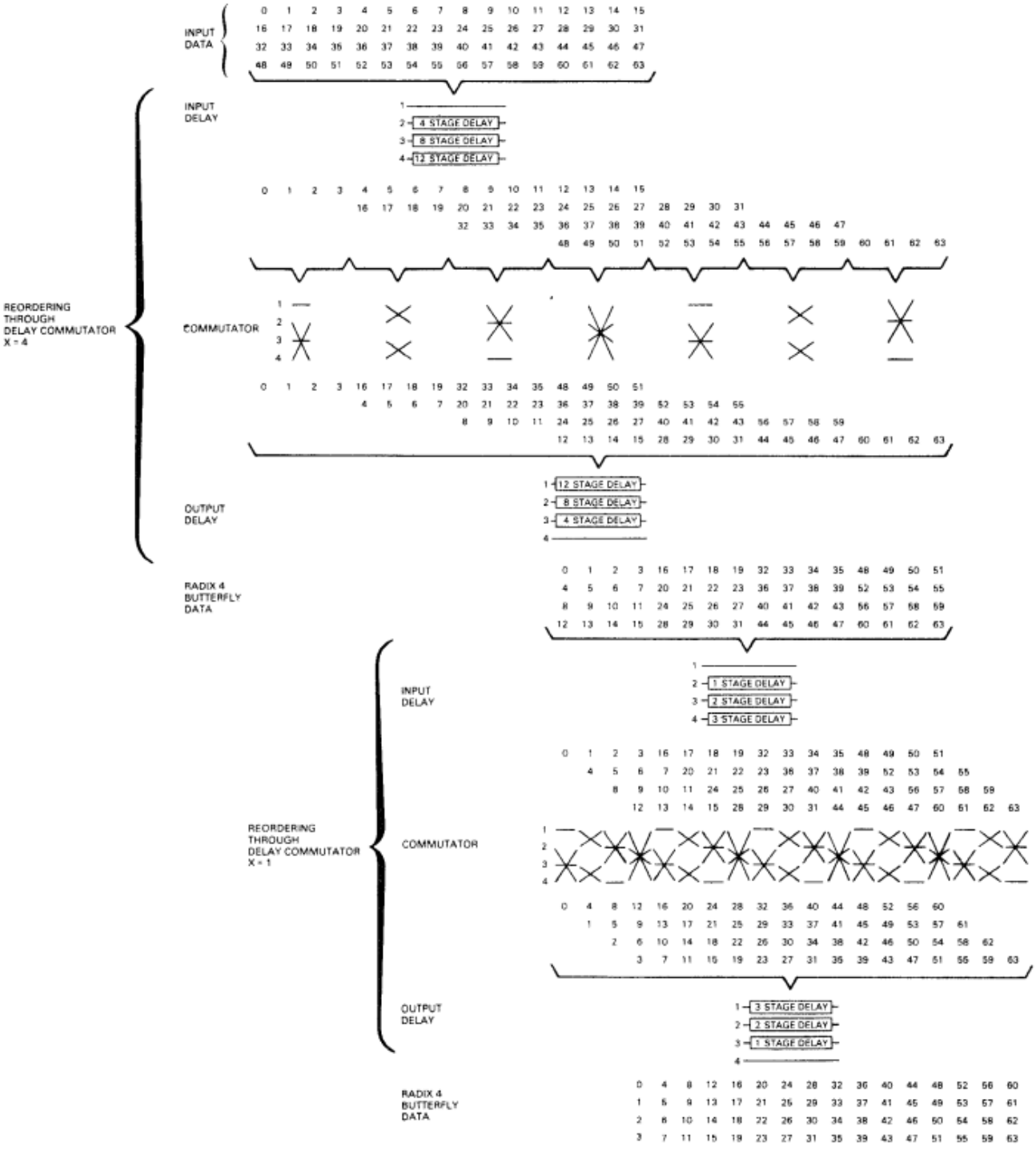


Figure 2-23 Data patterns through two delay commutators for a 64 point

2.2.3.2.3 Channel estimation techniques

This section is adapted from [14]

2.2.3.2.3.1 Least Squares (LS)

The least-square (LS) channel estimation method finds the channel estimate in such a way that the following cost function is minimized:

$$\begin{aligned} J(\hat{\mathbf{H}}) &= \left\|\mathbf{Y}-\mathbf{X}\hat{\mathbf{H}}\right\|^2 \\ &= (\mathbf{Y}-\mathbf{X}\hat{\mathbf{H}})^H(\mathbf{Y}-\mathbf{X}\hat{\mathbf{H}}) \\ &= \mathbf{Y}^H\mathbf{Y}-\mathbf{Y}^H\mathbf{X}\hat{\mathbf{H}}-\hat{\mathbf{H}}^H\mathbf{X}^H\mathbf{Y}+\hat{\mathbf{H}}^H\mathbf{X}^H\mathbf{X}\hat{\mathbf{H}} \end{aligned} \qquad (2.2\text{-}1)$$

The Y term represents the received signal, X is the sent data an H is the channel response term that we wish to find from the channel estimate.

By setting the derivative of the function with respect to H to zero:

$$\frac{\partial J(\hat{H})}{\partial \hat{H}} = -2X^{\mathsf{T}}Y + 2X^{\mathsf{T}}X\hat{H} = 0 \qquad (2.2\text{-}2)$$

$$\widehat{H_{LS}} = (X^{\mathsf{T}}X)^{-1}X^{\mathsf{T}}Y = X^{-1}Y \qquad (2.2\text{-}3)$$

So, the least squares estimate is dividing the received signal by the transmitted signal, here we will have pilots at the receiver with predetermined values, we will divide it by the received pilot signals, here we will get the deviation between the reference and received signal, then we use interpolation techniques are employed to infer values between the known pilot locations. This ensures a smooth and continuous estimation of the channel characteristics across the entire frequency spectrum.

2.2.3.2.3.2 Minimum Mean Square Error (MMSE)

The Minimum Mean Square Error (MMSE) method is a powerful technique that seeks to minimize the mean square error between the estimated channel response and the true channel response. To adapt the MMSE estimate to a specific system, a regularization factor, often denoted by W, is introduced.

This regularization factor plays a crucial role in controlling the trade-off between fitting the estimate closely to the observed data and maintaining stability in the presence of noise. The MMSE channel estimation involves taking the Least Squares (LS) estimate and multiplying it by the regularization factor. This regularization term is added to the diagonal elements of the channel covariance matrix, providing a controlled amount of regularization to the estimate.

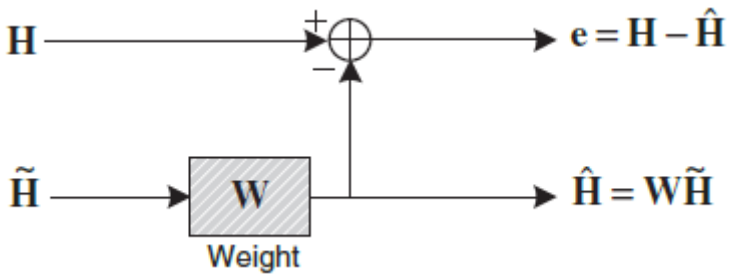


Figure 2-24: Weight factor in MMSE

The equation we want to minimize is:

$$J(\hat{\mathbf{H}}) = E\{||\mathbf{e}||^2\} = E\{||\mathbf{H}-\hat{\mathbf{H}}||^2\} \tag{2.2-4}$$

Thus, the MMSE channel estimation method finds a better (linear) estimate in terms of W in such a way that the MSE in Equation is minimized. The orthogonality principle states. that the estimation error vector is orthogonal to such that:

$$\begin{aligned} E\{\mathbf{e}\tilde{\mathbf{H}}^H\} &= E\{(\mathbf{H}-\hat{\mathbf{H}})\tilde{\mathbf{H}}^H\} \\ &= E\{(\mathbf{H}-\mathbf{W}\tilde{\mathbf{H}})\tilde{\mathbf{H}}^H\} \\ &= E\{\mathbf{H}\tilde{\mathbf{H}}^H\}-\mathbf{W}E\{\tilde{\mathbf{H}}\tilde{\mathbf{H}}^H\} \\ &= \mathbf{R}_{\mathbf{H}\tilde{\mathbf{H}}}-\mathbf{W}\mathbf{R}_{\tilde{\mathbf{H}}\tilde{\mathbf{H}}} = \mathbf{0} \end{aligned} \tag{2.2-5}$$

Solving for W we get:

$$\mathbf{W} = \mathbf{R}_{\mathbf{H}\tilde{\mathbf{H}}}\mathbf{R}_{\tilde{\mathbf{H}}\tilde{\mathbf{H}}}^{-1} \tag{2.2-6}$$

Solving for autocorrelation matrix we get:

$$\begin{aligned} \mathbf{R}_{\tilde{\mathbf{H}}\tilde{\mathbf{H}}} &= E\{\tilde{\mathbf{H}}\tilde{\mathbf{H}}^H\} \\ &= E\{\mathbf{X}^{-1}\mathbf{Y}(\mathbf{X}^{-1}\mathbf{Y})^H\} \\ &= E\{(\mathbf{H}+\mathbf{X}^{-1}\mathbf{Z})(\mathbf{H}+\mathbf{X}^{-1}\mathbf{Z})^H\} \\ &= E\{\mathbf{H}\mathbf{H}^H+\mathbf{X}^{-1}\mathbf{Z}\mathbf{H}^H+\mathbf{H}\mathbf{Z}^H(\mathbf{X}^{-1})^H+\mathbf{X}^{-1}\mathbf{Z}\mathbf{Z}^H(\mathbf{X}^{-1})^H)\} \\ &= E\{\mathbf{H}\mathbf{H}^H\}+E\{\mathbf{X}^{-1}\mathbf{Z}\mathbf{Z}^H(\mathbf{X}^{-1})^H\} \\ &= E\{\mathbf{H}\mathbf{H}^H\}+\frac{\sigma_z^2}{\sigma_x^2}\mathbf{I} \end{aligned} \tag{2.2-7}$$

Now the MMSE estimate is:

$$\begin{aligned} \hat{\mathbf{H}} = \mathbf{W}\tilde{\mathbf{H}} &= \mathbf{R}_{\mathbf{H}\tilde{\mathbf{H}}}\mathbf{R}_{\tilde{\mathbf{H}}\tilde{\mathbf{H}}}^{-1}\tilde{\mathbf{H}} \\ &= \mathbf{R}_{\mathbf{H}\tilde{\mathbf{H}}}\left(\mathbf{R}_{\mathbf{H}\mathbf{H}}+\frac{\sigma_z^2}{\sigma_x^2}\mathbf{I}\right)^{-1}\tilde{\mathbf{H}} \end{aligned} \tag{2.2-8}$$

The auto and cross correlation terms are:

$$E\left\{h_{k,l}\tilde{h}_{k',l'}^*\right\} = E\left\{h_{k,l}h_{k',l'}^*\right\} = r_f[k-k']r_t[l-l'] \tag{2.2-9}$$

The frequency domain correlation:

$$r_f[k] = \frac{1}{1 + j2\pi\tau_{rms}k\Delta f} \tag{2.2-10}$$

The time domain correlation:

$$r_t[l] = J_0\left(2\pi f_{\max} l T_{sym}\right) \tag{2.2-11}$$

The time domain correlation yields unity as it is a Bessel function as it an autocorrelation function between the symbol index and itself, it can be simplified from this graph:

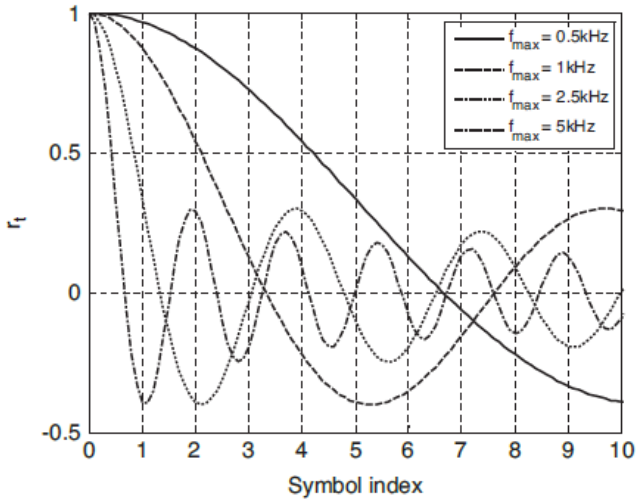


Figure 2-25: Time domain correlation characteristic

2.2.3.2.3.3 DFT BASED ESTIMATION

The DFT-based channel estimation method has been developed to enhance the efficacy of both Least Squares (LS) and Minimum Mean Square Error (MMSE) channel estimation by mitigating the influence of noise beyond the maximum channel delay. Leveraging the sparsity inherent in the channel response, the DFT transforms this response into the frequency domain, where only frequencies corresponding to substantial delays exhibit noteworthy energy.

This transformation by the DFT serves to segregate the frequency components associated with significant channel delays from those attributable to noise and interference. While noise outside the maximum channel delay in the time domain may disperse across multiple frequencies, the frequency domain allows for a clearer distinction and isolation of the pertinent components.

The DFT-based approach facilitates the suppression of noise components unrelated to significant channel delays. By concentrating on the pertinent frequency components, the impact of noise beyond the maximum channel delay is mitigated, resulting in a more precise and accurate channel estimation.

2.3 PEAK TO AVERAGE POWER RATIO (PAPR)

2.3.1 Introduction

One of the major problems of OFDM is that the peak amplitude of the emitted signal can be considerably higher than the average amplitude, this causes degradation in the efficiency of the power amplifier in the transmitter. High peak to average power causes nonlinear distortion in the transmitted OFDM signal when it is passed through a nonlinear HPA[15].
This nonlinear distortion causes serious in-band distortion as well as adjacent channel interference due to spectrum regrowth in the transmitted signa, also it causes reduction in the efficiency of the OFDM system which means that the receiver will not be able to receive the signal properly (increasing in bit error rates in the receiver). Mainly this problem originates from the fact that an OFDM signal is the superposition of N sinusoidal signals on independently different subcarriers and each of different phase value.
And since all subcarriers are added together in time domain via IFFT this may result in adding all subcarriers constructively at a certain instance causing high peak ,so that the amplitude of the signal will be proportional to N, and the power will be proportional to N^2 so increasing number of subcarriers will result in high PAPR[15].
The following figure illustrates the effect of N on PAPR:

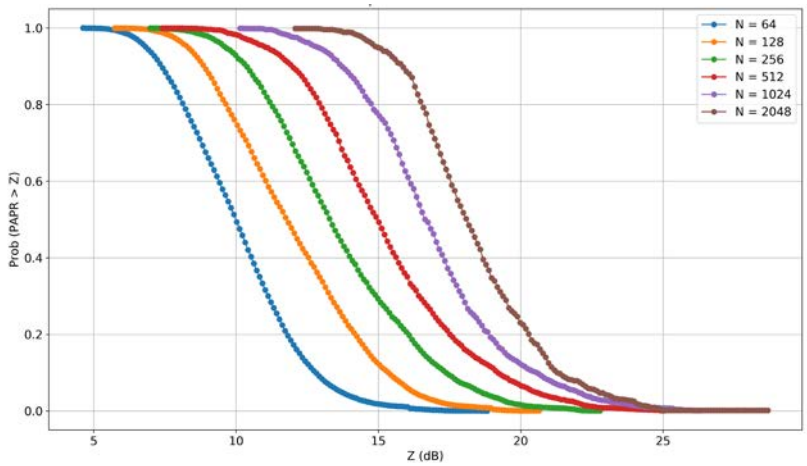


Figure 2-26 N vs PAPR

Generally, Peak to average power ratio is the ratio between the maximum power and the average power of the complex passband signal and it is usually represented in dB, the PAPR for the time domain OFDM signal can be defined as:

$$PAPR[s(n)] = \frac{P_{PEAK}}{P_{AVERAGE}} = 10\log_{10}\frac{\max\limits_{0\le n\le N-1}|S(n)|^2}{E[|S(n)|^2]}$$

Statistically it is possible to characterize the PAPR using Complementary Cumulative Distribution Function (CCDF). CCDF is the most common way to evaluate the PAPR by estimating the probability of PAPR when it exceeds a certain level. The CCDF expression of the PAPR of OFDM signals with relatively small subcarriers can be written as[16]:

$$CCDF = P(PAPR > PAPR_0)$$

This equation can be interpreted as the probability that the PAPR of a symbol block exceeds some threshold level $PAPR_0$.

2.3.2 Drawbacks of PAPR

It decreases SQNR (Signal-to-Quantization Noise Ratio) that reflects the relationship between the maximum nominal signal strength and the quantization error (also known as quantization noise) of ADC (Analog-to-Digital converter) and DAC (Digital-to-Analog Converter).

The main drawback of PAPR is degrading the efficiency of the high-power amplifier in the transmitter.
Most radio systems employ the HPA in the transmitter to obtain sufficient transmission power, Practical power amplifiers are linear only over a finite range of input amplitudes, so if input becomes much larger than its nominal value, output will be distorted due to saturation characteristics of power amplifiers. Nonlinearity in power amplifier response leads to nonlinear amplification of OFDM signal.

The following figure shows the input-output characteristics of high-power amplifier (HPA) in terms of the input power Pin and the output power Pout. Due to the saturation characteristic of the amplifier, the maximum possible output is limited by $Pout_{max}$when the corresponding input power is given by Pin_{max}. Therefore, the nonlinear region can be described by IBO (Input Back-Off) or OBO (Output Back-Off).

$$IBO = 10\log_{10}\left(\frac{p_{max_{in}}}{P_{in}}\right), \quad OBO = 10\log_{10}\left(\frac{p_{max_{out}}}{P_{out}}\right)$$

To prevent saturation and clipping of the OFDM signal peaks, the amplifiers must be operated with sufficient back-OF. However, increasing back-off will reduce the efficiency of the power amplifier[15].

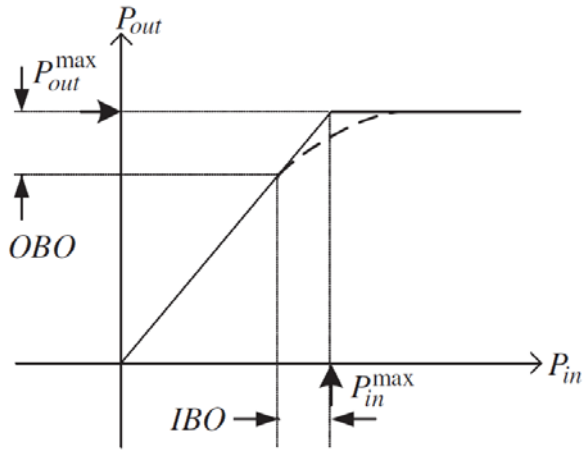


Figure 2-27 input-output characteristic of an HPA

2.3.3 PAPR Reduction Techniques

The paper explores various techniques to address the issue of high peak-to-average power ratio (PAPR) in OFDM systems. These techniques include clipping and filtering, coding, partial transmit sequence (PTS), selective mapping (SLM), and interleaving.

2.3.3.1 Coding Technique

Coding techniques offer a different approach to PAPR reduction in OFDM systems. Unlike clipping, which modifies the signal itself, coding techniques scramble the data before transmission. This scrambling is done in a way that allows the receiver to easily unscramble it and recover the original data.

There are several advantages to using coding for PAPR reduction. First, coding techniques are distortionless, meaning they don't introduce any noise or errors into the signal. Second, they can be combined with error correction capabilities, making the system more robust against transmission errors[17].
However, coding also has some drawbacks. The process of finding the optimal code for a particular signal can be complex, especially for systems with a large number of carriers. This complexity can increase the overall system overhead.

Researchers have explored various coding schemes for PAPR reduction. Some methods focus on simply selecting codewords that inherently have lower PAPR. Others combine PAPR reduction with error correction capabilities by using sophisticated codes. The following figure illustrates the effectiveness of different coding rates when using convolutional coding technique:

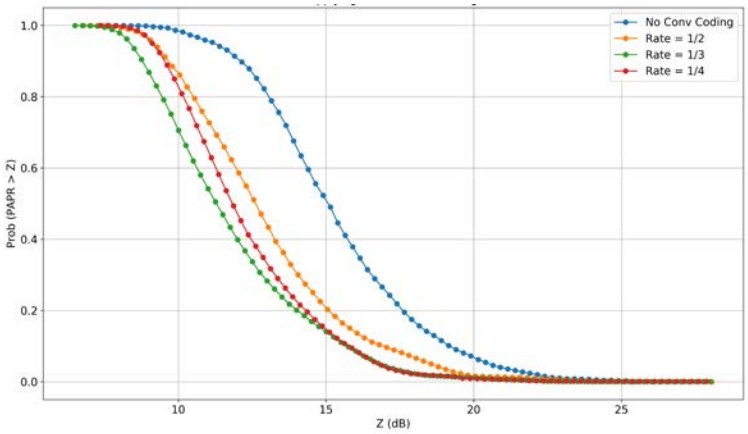


Figure 2-28 Coding Rate vs PAPR

2.3.3.2 Partial transmit sequence (PTS)

Partial transmit sequence (PTS) is another technique for reducing PAPR in OFDM systems. It works by dividing the data stream into smaller sub-blocks and applying a different phase shift to each sub-block. These phase shifts are carefully chosen to minimize the peak amplitude of the combined signal.

PTS offers several advantages. It can achieve significant PAPR reduction without distorting the signal itself. However, it also comes with some challenges.

One challenge is the need for side information (SI). The receiver needs to know the phase shifts that were applied to each sub-block in order to correctly decode the data. This SI needs to be transmitted along with the data, which can consume bandwidth and reduce the overall data rate.

The amount of SI required depends on the number of sub-blocks and the number of possible phase shifts used. As these numbers increase, so does the amount of SI needed. This can lead to a trade-off between PAPR reduction and bandwidth efficiency[16].

Researchers have proposed various modified PTS schemes that aim to reduce the amount of SI required. However, these schemes often come with their own drawbacks, such as difficulty in recovering the SI for signals with low signal-to-noise ratio (SNR) or increased complexity in the design process.

The OFDM transmitter with conventional PTS technique is shown in Fig. 2. The signal $X=(X_1, X_2 \dots X_N)$ is partitioned into disjoint sub-blocks X^V, of length N/V where N is the number of subcarriers. Every X^V disjoint sub-block is to be multiplied with a phase weighting factor (b^i, i=1, 2… V)[18].

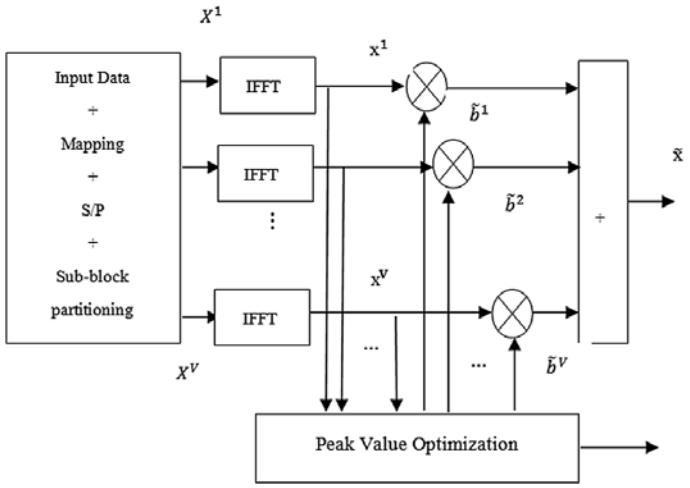


Figure 2-29 Block diagram of partial transmit sequence

So we can write the expression of sequence:

$$X = \sum_{i=1}^{V} b^i X^i , \quad \text{for} \quad i = 1,2,3, \dots , V$$

2.3.3.3 Tone injection

Tone Injection technique can be used to reduce the PAPR without reducing the data rate. It allows the PRTs to be overlapped with data tones. TI is based on mapping of original data that causes large peaks to several new positions this could be done by increasing the constellation size so that each of the points in the original constellation can be mapped into several equivalent points in the expanded constellation where the extra degrees of freedom can be exploited for PAPR reduction[18].
The receiver must know how to map the redundant positions on the original one. TI is distortion less technique and does not exhibit data rate loss. However, transmitter is more complex as it requires additional IFFT operation. TI technique also requires more signal power for transmission of signals to transmit the symbols in the expanded constellation.

2.3.3.4 Clipping and Filtering Technique

Clipping is a simple technique for reducing PAPR in OFDM systems. It works by limiting the peak amplitude of the signal to a certain level. This effectively cuts off the highest peaks of the signal waveform[18].

While clipping is easy to implement, it has some drawbacks. Clipping introduces distortion into the signal, which can lead to noise and reduced performance. This noise can be both in-band (meaning it falls within the desired transmission bandwidth) and out-of-band (meaning it falls outside the desired bandwidth). In-band noise can cause errors in the transmitted data, while out-of-band noise can interfere with other signals.

One way to mitigate the negative effects of clipping is to use filtering after the clipping process. Filtering can help to remove some of the out-of-band noise generated by clipping. However, filtering can also cause the clipped peaks to regrow, which can reintroduce distortion.

Researchers have proposed several methods to improve clipping performance. Some methods use special filters designed to remove clipping noise without affecting the signal too much. Others use techniques to reconstruct the signal after clipping, reducing the impact of the distortion. Still others propose adaptive clipping algorithms that adjust the clipping level based on the signal characteristics. These methods can help to achieve a better balance between PAPR reduction and signal quality[19].

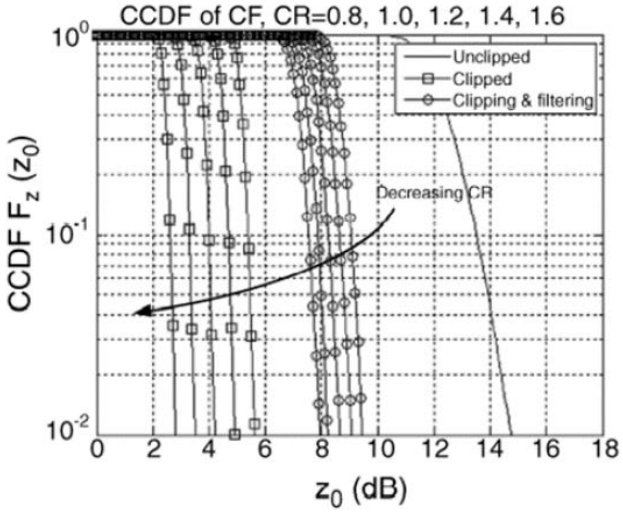


Figure 2-30 PAPR distribution using clipping tech

An important parameter in this technique is the clipping ratio (CR)which is the ratio between the clipping level normalized by the RMS value of OFDM signal

$$CR = \frac{A}{\sigma}$$

It has been known that $\sigma = \sqrt{N}$, $\sigma = \sqrt{\frac{N}{2}}$ in the baseband and passband OFDM signals with N subcarriers, respectively. The previous figure shows the CCDF of PAPR for the clipped and filtered OFDM signals. We notice that the smaller the clipping ratio (CR) is, the greater the PAPR reduction effect.

2.3.3.5 Selected mapping (SLM)

Selected mapping (SLM) is a technique for PAPR reduction in OFDM systems that works by creating multiple versions of the data signal. The transmitter then chooses the version with the lowest PAPR to transmit. This requires sending some side information (SI) to the receiver to indicate which version was chosen.
SLM offers good PAPR reduction, but it also has some drawbacks. The number of versions that can be generated increases with the number of carriers in the signal. This can lead to increased complexity in the transmitter and require more SI to be transmitted, reducing bandwidth efficiency[20].
Researchers have proposed various modified SLM schemes to address these limitations. One approach is to decompose complex signals into real signals, allowing for more versions to be generated without increasing computational complexity.
Another approach is to use codes like Gold or Hadamard codes to optimize the phase sequences used in the different versions, reducing the amount of SI needed.
These modified SLM schemes can achieve good PAPR reduction while keeping complexity and SI overhead low. For example, some studies have shown significant PAPR improvements for different numbers of subcarriers and with varying amounts of phase sequences used.

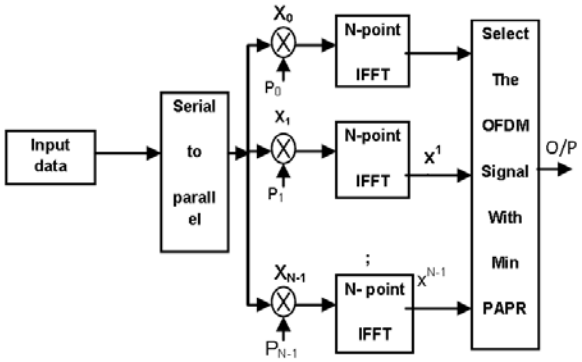


Figure 2-31 Block diagram of Selective Mapping

Here, the input data block X=[x[0],x[1]…,x[N−1]] is multiplied with U different phase sequences p^u=[$p_0{}^u$,$p_1{}^u$,…,$p_{N-1}{}^u$] ^{T} which produce a modified data blocks Xu=[x^u[0],x^u[1],…..,x^u[N−1]] ^{T} which represent the same data,
These are then forwarded into IFFT operation simultaneously. And then the PAPR is calculated for each vector separately. The sequence with the smallest PAPR is selected for final transmission. For the receiver to be able to recover the original data block, the information about the selected phase sequence P^u should be transmitted as a side information. The previous figure illustrates the block diagram of SLM.
Selective Mapping is PAPR reduction techniques to provide good performance without exhibiting any signal distortion. Selective Mapping method is employed and then clipping is performed to achieve further reduction in PAPR. Clipping is performed due to its own advantages such as less complex circuitry and simple computations[21]. Simulation results of a combined scheme of Selective Mapping and Clipping are shown in the following figure.

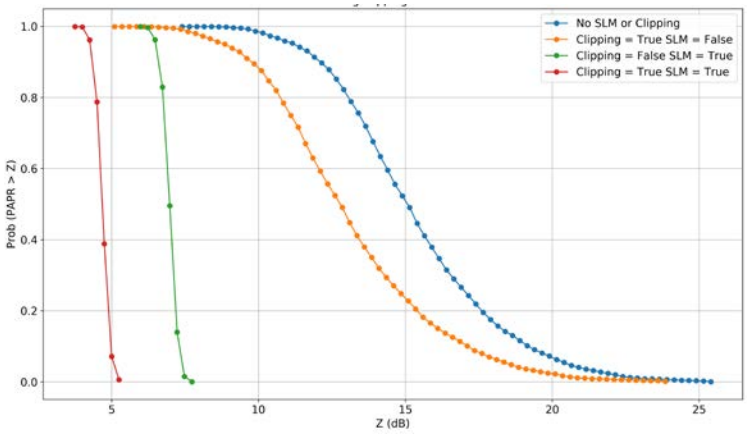


Figure 2-32 SLM-Clipping vs PAPR

2.4 PAPR REDUCTION NETWORK FOR OFDM BASED ON DEEP LEARNING

2.4.1 Motivation

Orthogonal Frequency Division Multiplexing (OFDM) is a widely used modulation technique due to its robustness against channel impairments. However, a major drawback of OFDM is its high Peak-to-Average Power Ratio (PAPR)[22]. This high PAPR can lead to several challenges in practical OFDM systems, including:

- Non-linear distortion: High PAPR signals can cause non-linear distortion in power amplifiers, leading to spectral regrowth and increased out-of-band emissions. This can violate regulatory requirements and interfere with other nearby signals.
- Reduced power efficiency: Power amplifiers operate most efficiently at a certain level. With high PAPR, the average signal power is much lower than the peak power, requiring amplifiers to operate in a less efficient regime.
- Bit Error Rate (BER) degradation: Techniques like clipping, commonly used for PAPR reduction, introduce distortion into the signal, which can degrade the BER performance of the system.

Classical PAPR reduction techniques have been proposed to address these challenges. However, these techniques often suffer from limitations in several key areas:

- Computational Complexity: Some techniques, such as Partial Transmit Sequence (PTS) and Selected Mapping (SLM), involve complex optimization algorithms that increase the computational burden on the transmitter. This can be a significant issue for real-time applications.
- Overhead: Techniques like PTS require transmitting side information (SI) to the receiver to recover the original signal. This SI consumes bandwidth and reduces the overall data rate.
- BER Performance: Clipping, a simple and widely used technique, effectively reduces PAPR but introduces distortion into the signal. This distortion can degrade the BER performance of the system, especially at low Signal-to-Noise Ratio (SNR) levels.

Therefore, there is a need for a new PAPR reduction technique that can overcome the limitations of classical approaches. Ideally, this new technique should be:

- **Computationally efficient:** The technique should have low computational complexity to be suitable for real-time implementation.
- **Low overhead:** The technique should minimize the need for transmitting SI, maximizing bandwidth efficiency.
- **BER-preserving:** The technique should reduce PAPR without introducing significant distortion, ensuring good BER performance even at low SNR levels.

This thesis implements PAPR reduction technique for OFDM systems based on Deep Learning (DL). Deep learning has shown remarkable success in various signal processing tasks, and we believe it has the potential to address the limitations of classical PAPR reduction techniques. We aim to develop a DL-based approach that is computationally efficient, requires minimal overhead, and maintains good BER performance.

2.4.2 Idea

This research explores using deep learning to tackle the high PAPR issue in OFDM systems. We propose a technique that utilizes a specific type of deep neural network (DNN) called an autoencoder. Autoencoders are known for their ability to denoise corrupted data. In our approach, the autoencoder is trained to learn how to map and de-map symbols on each subcarrier in an OFDM system. This learning process aims to achieve two goals: significantly reduce the PAPR of the transmitted signal and minimize any degradation in the bit error rate (BER). Additionally, after the training phase, our proposed method operates with minimal overhead, making it suitable for real-time applications.

2.4.3 The DNN Model

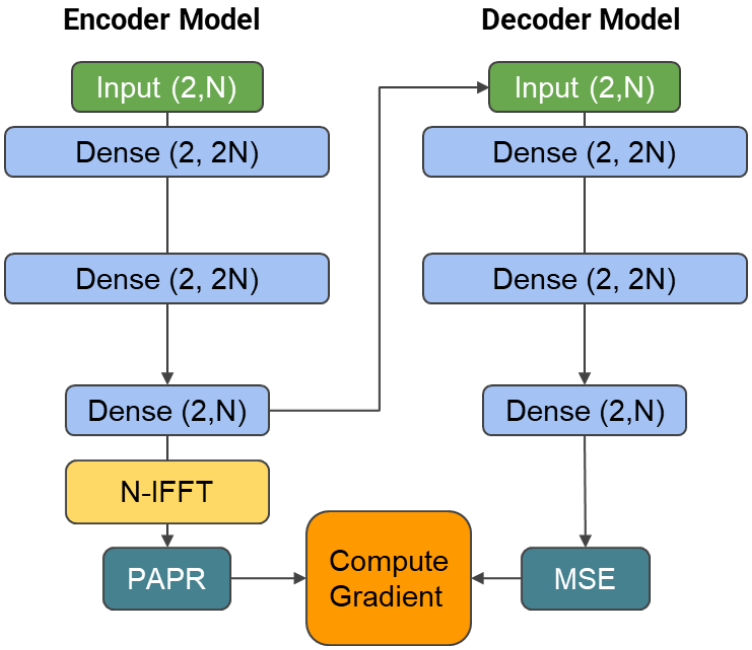


Figure 2-33 Encoder & Decoder Model Diagram

In the proposed system, we adopt a simple and efficient structure for both the encoder and decoder within the PRNet architecture. As depicted in the previous figure, these components share identical sub-blocks (hidden layers) connected in a sequential manner. Each sub-block comprises two key elements: a fully connected (FC) layer and a rectifier linear unit (ReLU) activation function.

The FC layer performs a matrix multiplication of the input data with a weight matrix, followed by the addition of a bias vector. This operation essentially learns a linear transformation of the input. The output of the FC layer is then passed through the ReLU activation function. ReLU introduces non-linearity into the network, allowing it to learn more complex relationships within the data. The ReLU function operates by preserving the input value if it's positive, and setting it to zero otherwise. Finally, the encoded symbols generated by the network undergo an inverse discrete Fourier transform (IFFT) operation to convert them into the time-domain signal for transmission.

2.4.4 Training

The proposed architecture addresses the challenge of PAPR reduction in OFDM systems while simultaneously maintaining good bit error rate (BER) performance. To achieve this, the network is trained with two distinct objectives:

- Signal Reconstruction: The network must learn to effectively reconstruct the transmitted signal from the received signal. This ensures minimal distortion and accurate data recovery at the receiver, thereby minimizing BER degradation.
- Low PAPR Transmission: The network must be trained to generate a transmission signal with a low peak-to-average power ratio (PAPR). This is crucial for mitigating the negative effects of high PAPR in OFDM systems, such as amplifier non-linearities and reduced power efficiency.

By optimizing both of these objectives during the training process, PRNet aims to achieve a balance between low PAPR and good BER performance, leading to a more robust and efficient OFDM communication system.

2.4.4.1 Joint Loss

To achieve the former objective, the encoder of the network is trained to find the proper constellation mapping from the input data to the output and the decoder of the network must be able to decode the received signal. Then, the proper loss function to achieve this objective can be written as the Binary Cross Entropy (log loss):

$$L_1 = -\frac{1}{K}\sum_{K=1}^{K} y_i \, log\big(p(y_i)\big) + (1 - y_i).log\big(1 - p(y_i)\big)$$

Also, the encoder must keep track of decreasing PAPR,

$$L_2 = PAPRIFFT(output_{encoder})$$

So, the model ultimately will try to decrease the joint loss which will be:

$$L_{jiont} = L_1 + L_2$$

2.4.5 Performance Evaluation

The performance evaluation of the network demonstrates its superiority in PAPR reduction compared to existing techniques like SLM-32, SLM-64, SLM-64 with clipping, and normal OFDM. This is evident in the complementary cumulative distribution function (CCDF) graphs, where PRNet exhibits the lowest CCDF curve. This translates to a significantly higher probability of encountering lower PAPR values in the transmitted signal compared to the other methods. This advantage makes PAPRnet a compelling choice for mitigating the negative effects of high PAPR in OFDM systems, leading to improved system efficiency and robustness.

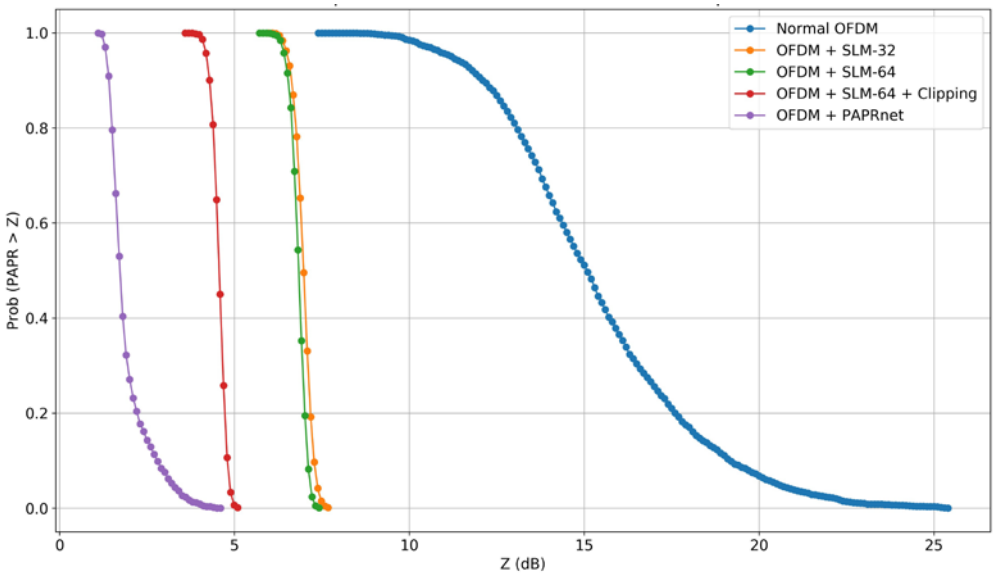


Figure 2-34 PAPR Network Performance

2.5 HARDWARE IMPLEMENTATION OF NEURAL NETWORKS

In the realm of hardware implementation for Deep Neural Networks (DNNs), Reconfigurable Field-Programmable Gate Arrays (FPGAs) have emerged as a powerful and versatile solution. FPGAs offer several key advantages that make them well-suited for DNN applications.
First, they are significantly more cost-effective compared to custom-designed ASICs. Second, FPGAs boast high availability, readily obtainable from various vendors. Finally, and perhaps most importantly, FPGAs possess the unique ability to be reconfigured.

This reconfigurability allows for on-the-fly adjustments to the DNN architecture, enabling adaptation to changing requirements or the exploration of different network designs. These combined attributes – cost-effectiveness, availability, and reconfigurability – make FPGAs a highly attractive choice for hardware-based DNN implementations[23].

Reconfigurable Field-Programmable Gate Arrays (FPGAs) offer significant advantages over traditional custom Very Large-Scale Integration (VLSI) circuits for implementing Deep Neural Networks (DNNs). Unlike VLSI, FPGAs are readily available at a reasonable cost, leading to a reduced hardware development cycle. Additionally, FPGA-based DNNs can be tailored to specific network configurations, eliminating the need for worst-case, fully interconnected designs required in full-custom VLSI implementations. This flexibility allows for more efficient hardware utilization.

However, a key challenge in using FPGAs for DNNs is the high cost associated with implementing the numerous multiplications required for computations. Since each synaptic connection in a DNN essentially involves a multiplication, the number of multipliers scales rapidly with network size. For example, a fully connected 10-neuron DNN necessitates 100 multipliers, while a 100-neuron network would require 10,000. This substantial multiplier requirement necessitates exploring techniques to optimize hardware utilization and mitigate the cost limitations associated with FPGA-based DNN implementations[24].

2.5.1 Implementing Neurons

The efficient implementation of a single neuron is paramount to the successful hardware realization of Deep Neural Networks (DNNs). Field-Programmable Gate Arrays (FPGAs) offer a compelling solution for hardware implementation due to their inherent reconfigurability. This reconfigurability allows for the creation of custom computing architectures specifically tailored for DNN workloads. However, a major challenge arises when scaling DNNs on FPGAs – the sheer number of neurons in large networks. Implementing these networks efficiently on FPGAs requires innovative techniques to overcome limitations and achieve practical performance[25].

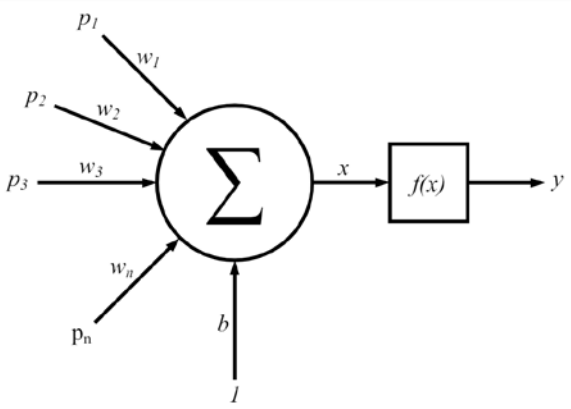


Figure 2-35 Neuron Diagram

The basic structure of a neuron with ‘n’ inputs is shown in the previous figure.

The function of a neuron is described by the following equations.

$$x = \sum_{i=1}^{n} p_i w_i + b$$

where p_i be the i-th inputs of the system w_i is the weight in the i-th connection and 'b' is the bias

$$y = F(x)$$

The function f(x) is the excitation function used in the neuron. Generally Linear, Log-sigmoid and Rectified Linear Unit excitation functions are used. They are defined as:

Linear

$$f(x) = x$$

Sigmoid

$$f(x) = \frac{1}{1 + e^{-x}}$$

ReLU

$$f(x) = \begin{cases} x, & x \geq 0 \\ 0, & x < 0 \end{cases}$$

linear and ReLU functions leverage Verilog's built-in arithmetic operations for efficient hardware implementation.
On the other hand, the sigmoid Due to its complexity, the sigmoid function is typically implemented using Look-Up Tables (LUTs) in Verilog HDL. LUTs are pre-defined memory elements that store pre-computed values of the function for different input ranges. The input (x) is used as an address to access the corresponding pre-computed sigmoid value from the LUT, which becomes the output.
While this approach is less computationally expensive than direct calculations, it introduces a trade-off between accuracy and hardware resource utilization.

The following figure shows a simplified representation of the neuron hardware implementation where the weights and bias are stored in ROMs. Each input is multiplied by its corresponding weight then the result is accumulated. After the last input is processed bias is added and the result is the input to the activation function. The output of the activation function is the output of the neuron which is the passed as the output of the network or as an input to the next layer

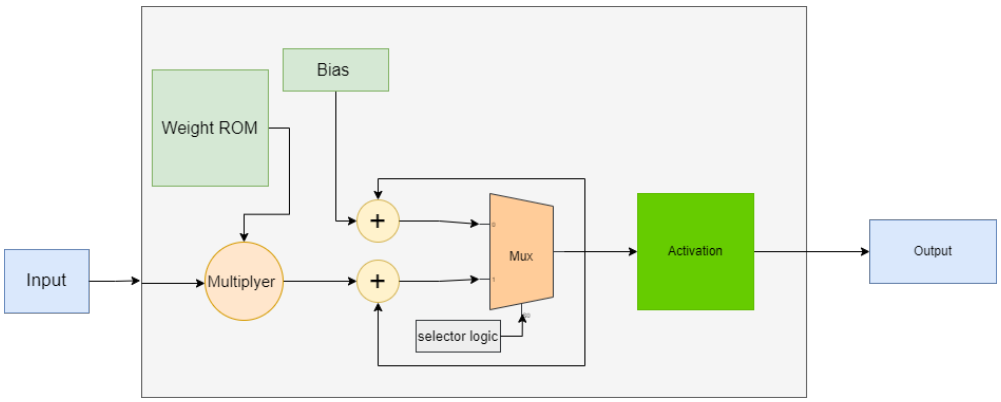


Figure 2-36 Neuron Circuit

2.5.2 Deep Neural Networks

A Deep Neural Network (DNN) is a type of artificial neural network (ANN) that has multiple hidden layers between the input and output layers. The term "deep" refers to the number of layers through which data must pass in a network. Each layer consists of nodes (neurons) that process and transform the input data. DNNs are characterized by their layered architecture, where each hidden layer learns to transform the data in a way that makes it easier for the subsequent layer to perform its task. The following figure shows a sample structure of a DNN.

This hierarchical feature learning enables lower layers to often learn low-level features (like edges in image processing), while higher layers learn more abstract concepts (like objects or faces).

The structure of a DNN includes an input layer, hidden layers, and an output layer. The input layer is the first layer that receives the input data. The hidden layers are where the actual processing occurs through weighted connections. The number of hidden layers can vary, leading to different depths of the network. The output layer is the final layer, which produces the network's output. DNNs use non-linear activation functions (like ReLU, Sigmoid, Tanh) in the hidden layers, enabling them to learn complex patterns and representations.

Training a DNN involves two main processes: forward propagation and backward propagation. In forward propagation, data is passed from the input layer through the hidden layers to the output layer. In backward propagation, the error between the predicted and actual output is calculated and propagated back through the network to update the weights. This process allows the network to learn and improve its performance over time. DNNs have found applications in a wide range of fields, including image and speech recognition, natural language processing, autonomous vehicles, game playing, and time-series forecasting[24].

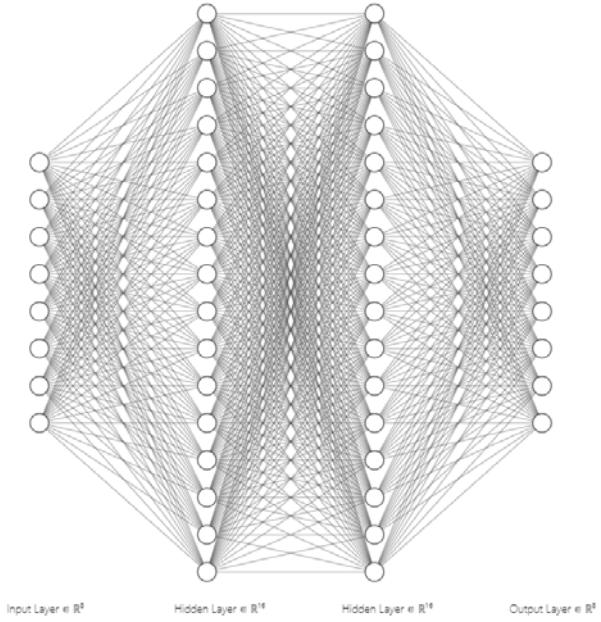


Figure 2-37 Sample DNN Diagram

Pre-trained neural networks are models that have been previously trained on large datasets and saved for future use. These models, often trained on extensive datasets like ImageNet, learn robust feature representations that can be transferred to new tasks. The process involves training a deep neural network (DNN) on a large, diverse dataset, saving the learned weights and biases, and then fine-tuning this pre-trained model on a smaller, task-specific dataset. This approach leverages the knowledge gained during the initial training, making the model more efficient and effective for new tasks.

In the context of hardware implementation, pre-trained neural networks offer several advantages. Firstly, they significantly reduce computational load by eliminating the need for extensive training. This efficiency translates to faster inference times when deploying these models on hardware such as GPUs, TPUs, FPGAs, and ASICs. These specialized devices are optimized for parallel processing, making them ideal for executing pre-trained models quickly and efficiently.

2.5.3 Implementing layers

Implementing a layer in a deep neural network (DNN) on hardware involves mapping the computational processes of the layer's neurons onto a hardware architecture. Each neuron in the layer performs a weighted sum of its inputs, followed by an activation function. This process requires efficient management of memory, data transfer, and computation to optimize performance and power consumption.
Firstly, each neuron in the layer must be instantiated in hardware. This involves creating circuits that can perform the necessary arithmetic operations, such as addition and multiplication, required for the weighted sum. In hardware implementations, these operations can be parallelized to increase throughput. For example, using a field-programmable gate array (FPGA), the neurons can be instantiated as parallel processing units, each handling its specific set of weights and inputs simultaneously. This parallelism is key to achieving high-speed computation.
Memory management is a critical aspect of hardware implementation. Each neuron requires access to its specific set of weights, which must be stored in memory. Efficient memory access patterns are essential to minimize latency and ensure that data is available when needed. Data transfer and interconnects are crucial in a hardware implementation of a DNN layer. The inputs to each neuron and the outputs from each neuron need to be efficiently routed between different layers of the network. This requires a well-designed interconnect architecture that can handle the high bandwidth requirements of a DNN. In an FPGA, this involves configuring the routing fabric to ensure minimal delays.

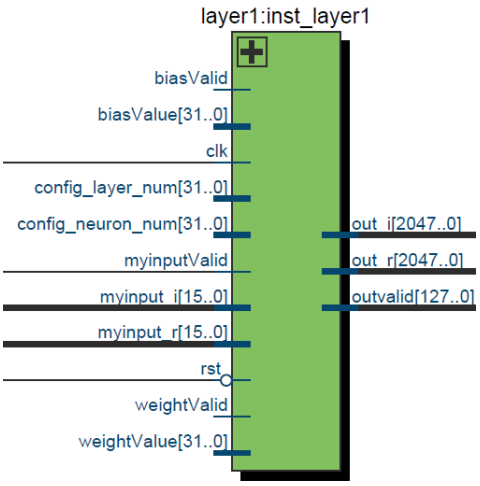


Figure 2-38 Layer Hardware instance

2.5.4 Encoder and Decoder DNN

Implementing an encoder and decoder deep neural network (DNN) on an FPGA involves mapping the computational processes of each network onto the FPGA architecture. Both the encoder and decoder have identical structures but differ in their sets of weights and biases. Each layer in the networks is implemented as described previously, with the output of each layer pipelined to the next, ensuring efficient data flow and high throughput.

The encoder and decoder consist of multiple layers, each implemented with a set of neurons that perform weighted sums followed by activation functions. In the FPGA implementation, these layers are instantiated as parallel processing units. This parallelism leverages the FPGA’s capability to perform multiple computations simultaneously, significantly enhancing the throughput of the network. Each neuron is mapped to a dedicated logic block on the FPGA, which performs the necessary arithmetic operations, including multiplication and addition for the weighted sum and the non-linear activation function.

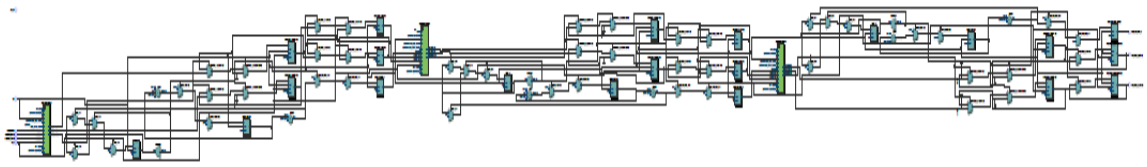


Figure 2-39 Encoder / Decoder Circuit Diagram

In summary, the FPGA implementation of an encoder and decoder DNN involves mapping the neurons of each layer onto the FPGA's parallel processing units, managing memory efficiently, implementing activation functions, designing robust pipelined data transfer mechanisms, and optimizing for power efficiency. By leveraging the FPGA's inherent parallelism and flexibility, this approach enables the efficient execution of encoder and decoder networks, making them suitable for a wide range of applications, from real-time data processing to embedded systems.

Flow Status	Flow Failed - Fri Jun 21 19:49:56 2024
Quartus Prime Version	23.1std.0 Build 991 11/28/2023 SC Lite Edition
Revision Name	DecoderNN
Top-level Entity Name	DecoderNN
Family	Cyclone IV E
Device	EP4CE22F17C6
Timing Models	Final
Total logic elements	276,423 / 22,320 (1238 %)
Total registers	58993
Total pins	69 / 154 (45 %)
Total virtual pins	0
Total memory bits	1,048,576 / 608,256 (172 %)
Embedded Multiplier 9-bit elements	132 / 132 (100 %)
Total PLLs	0 / 4 (0 %)

Figure 2-40 NN-Utilization

2.5.5 Conclusion

Our implementation of the encoder and decoder, which are identical in structure, has resulted in a design whose size exceeds the available resources on the target FPGA. This finding highlights a significant limitation in using FPGAs for such designs, making them impractical for real-world applications. This issue underscores a broader challenge in the commercial adoption of neural networks, particularly related to hardware design constraints. While FPGAs are well-suited for small designs that require very fast operation, their capacity limitations make them less viable for larger, more complex designs.
This conclusion suggests that for substantial neural network implementations, alternative solutions may be necessary to achieve practical and efficient deployment. Our implementation of the encoder and decoder, which are identical in structure, has resulted in a design whose size exceeds the available resources on the target FPGA.

This finding highlights a significant limitation in using FPGAs for such designs, making them impractical for real-world applications. This issue underscores a broader challenge in the commercial adoption of neural networks, particularly related to hardware design constraints. While FPGAs are well-suited for small designs that require very fast operation, their capacity limitations make them less viable for larger, more complex designs. This conclusion suggests that for substantial neural network implementations, alternative solutions may be necessary to achieve practical and efficient deployment.

2.6 MIMO CHANNEL MODELING

This chapter is adapted from [14]

We will begin by providing a summary of a statistical channel model for MIMO systems. Specific methods for implementing the MIMO channel are generally split into two methods: the correlation-based I-METRA channel model and the ray-based 3GPP spatial channel model (SCM), for which we will focus on the former. The correlation-based model uses a spatial correlation matrix for the channel and requires independent generation of temporal correlation using the specified Doppler spectrum. Conversely, the ray-based model utilizes multiple rays distributed across the angular domain based on the Power Azimuth Spectrum (PAS) and does not need a Doppler spectrum or spatial correlation matrix. However, it involves complex computational procedures.

2.6.1 Statistical MIMO Model

Remember that in characterizing the SISO system, delay spread, and Doppler spread are crucial considerations. In the MIMO system, which uses multiple antennas at both the transmitter and receiver, the correlation between the transmit & receive antennas is a significant element of the MIMO channel. This correlation is influenced by the angle-of-arrival (AoA) of each multipath component.
Consider a SIMO channel with a uniform linear array (ULA) in which M antenna elements are equally spaced apart, with an inter-distance of d as shown in Figure 3.1. Let $y_i(t)$ denote a received signal at the ith antenna element with the channel gain α_i, delay τ_i, and angle of arrival (AoA) ϕ_i. As shown in Figure 2-22, the AoA is defined as the azimuth angle of incoming path with respect to the broadside of the antenna element. Note that the received signal of each path consists of the enormous number of unresolvable signals received around the mean of AoA in each antenna element. A vector of the received signals $\mathbf{y}(t) = [y_1(t), y_2(t), \dots, y_M(t)]^T$ in the uniform linear array (ULA) of M elements can be expressed as

$$\mathbf{y}(t) = \sum_{i=1}^{I} \alpha_i \, c(\phi_i)\mathbf{x}(t - \tau_i) + \mathbf{N}(t)$$

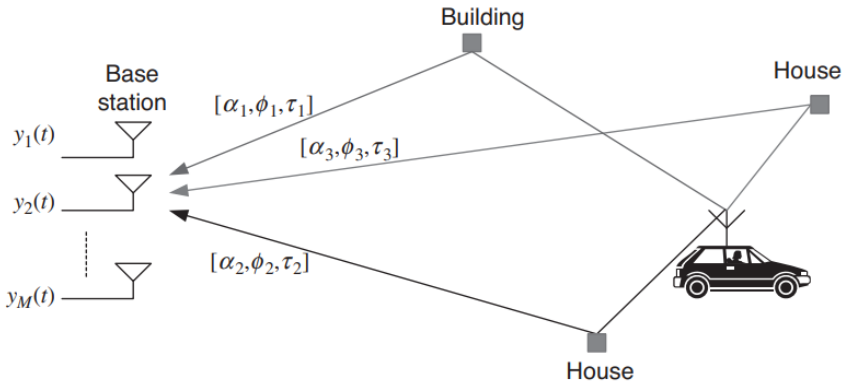


Figure 2-41 SIMO channel environment: an illustration.

where I denotes the number of paths in each antenna element and $\mathbf{c}(\phi)$ is an array steering vector. The array steering vector is defined as

$$\mathbf{c}(\phi) = [c_1(\phi), c_2(\phi), \dots, c_M(\phi)]^T$$

Were

$$c_m(\phi) = f_m(\phi)e^{-j2\pi(m-1)(d/\lambda)\sin\phi}, \quad m = 1,2,\dots,M$$

$f_m(\phi)$ denotes a complex field pattern of the mth array element and λ is the carrier wavelength. The received signal can be expressed in the following integral form:

$$\mathbf{y}(t) = \int\int \mathbf{c}(\phi)\mathbf{h}(\phi,\tau)x(t-\tau)dtd\tau + \mathbf{N}(t)$$

where $\mathbf{h}(\phi,\tau)$ represents a channel as a function of ADS (Azimuth-Delay Spread). The instantaneous power azimuth-delay spectrum (PADS) is given as

$$P_{\text{inst}}(\phi,\tau) = \sum_{i=1}^{I} |\alpha_i|^2\, \delta(\phi-\phi_i, \tau-\tau_i)$$

The average PADS is defined as an expected value of the above, such that

$$P(\phi,\tau) = \mathbb{E}[P_{\text{inst}}(\phi,\tau)]$$

By taking an integral of PADS over delay, PAS (Power Azimuth Spectrum or Power Angular Spectrum) is obtained as

$$P_A(\phi) = \int P(\phi,\tau)d\tau$$

Meanwhile, AS (Azimuth Spread or Angular Spread) is defined by the central moment of PAS, that is

$$\sigma_A = \sqrt{\int (\phi-\phi_0)^2 P_A(\phi)d\phi}$$

where ϕ_0 is the mean AoA (i.e., $\phi_0 = \int \phi P_A(\phi)d\phi$). Similarly, by taking an integral of PADS over AoA, PDS (Power Delay Spectrum) is obtained as

$$P_D(\tau) = \int P(\phi,\tau)d\phi$$

Furthermore, DS (Delay Spread) is defined as the central moment of PDS, that is,

$$\sigma_D = \sqrt{\int (\tau-\tau_0)^2 P_D(\tau)d\tau}$$

where τ_0 is an average delay spread (i.e., $\tau_0 = \int \tau P_D(\tau)d\tau$).

While Clarke's channel model assumes that AoA is uniformly distributed in the mobile station (MS), its distribution is significantly different in the base station (BS). In general, spatial correlation in the MS turns out to be almost zero for the antenna elements that are equally spaced by $\lambda/2$. In order to warrant a low spatial correlation in the BS, however, the antenna elements must be spaced by roughly 10λ to 40λ, even if it depends on AS. Meanwhile, the PDF of delay spread is typically approximated by an exponential function.

Every path comprises M_r indistinguishable sub-paths that are centered around the average Angle of Arrival (AoA). These indistinct sub-paths are distributed normally in both microcell and macrocell environments. Moreover, the distribution of power across the AoA, known as the Power Azimuth Spectrum (PAS), typically adheres to a Laplacian distribution, though this can change depending on the cellular environment. It is important to recognize that the distribution characteristics for AoA and PAS are distinct. Specifically, the AoA distribution does not consider the power associated with each path, whereas the PAS specifically represents how power is distributed in relation to the AoA. Lastly, the power distribution for discernible paths, also referred to as the Power Delay Spectrum (PDS) or Power Delay Profile (PDP), is generally characterized by an exponential distribution.

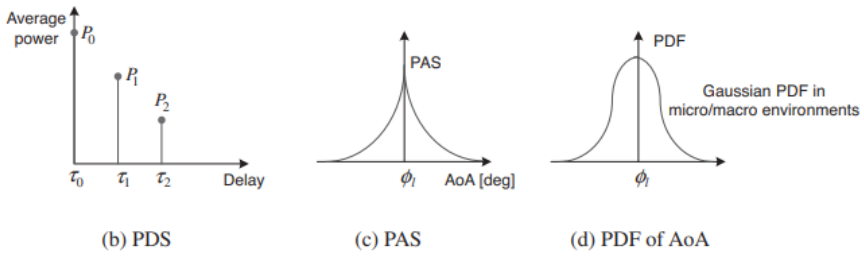


Figure 2-42 MIMO channel model: an illustration.

In general, the received signals for each path of the different antenna elements may be spatially correlated, especially depending on the difference in their distances travelled. Let us investigate the spatial correlation between the received signals in the different antennas.

Consider two omni-directional antennas, a and b, that are spaced apart by d as shown in Figure 2-24. For the baseband received signals with the mean AoA of ϕ_0, the difference in their distance traveled is given by $d\sin\phi_0$ and the corresponding delay becomes $\tau_0 = \frac{d}{c}\sin\phi_0$. Let α and β denote the amplitude and phase of each path, which follow the Rayleigh distribution and uniform distribution over $[0,2\pi)$, respectively.

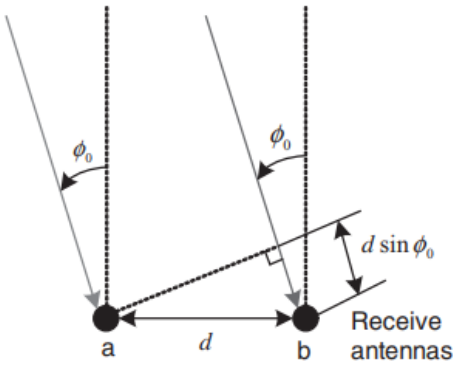


Figure 2-43 Signal models for two omni-directional antennas.

Assuming a narrow-band channel, their impulse responses can be respectively represented as

$$h_a(\phi) = \alpha e^{j\beta}\sqrt{P(\phi)}$$

and

$$h_b(\phi) = \alpha e^{j(\beta+2\pi d\sin(\phi)/\lambda)}\sqrt{P(\phi)}$$

where $P(\phi)$ denotes the PAS defined such that $P(\phi) = P_A(\phi)$.
Let us define a spatial correlation function of the received signals with the mean AoA of ϕ_0 in two antenna elements spaced apart by d as

$$\rho_c(d,\phi_0) = \mathbb{E}\{h_a(\phi)h_b^*(\phi)\}$$

$$= \int_{-\pi}^{\pi} h_a(\phi)h_b^*(\phi)P(\phi-\phi_0)d\phi.$$

Consider an extreme case that has the mean AoA of $\phi_0 = 0°$ and AS of $\sigma_A = 0$, that is, $P(\phi-\phi_0) = \delta(\phi)$, implying that there exists only one sub-ray in a perpendicular direction for each antenna element. In this particular case, AoA does not incur any time difference between h_a and h_b. Therefore, spatial correlation is always equal to 1, that is, $\rho_c(d) = \mathbb{E}\{h_a(\phi)h_b^*(\phi)\} = \mathbb{E}\{|\alpha|^2\} = 1$. However, in the case that both AoA and AS are not equal to

0°, there is a time difference between $h_a(\phi)$ and $h_b(\phi)$ as shown in. This yields the following spatial correlation function:

$$\rho_c(d, \phi_0) = \mathbb{E}\{h_a(\phi) h_b^*(\phi)\}$$

$$= \int_{-\pi}^{\pi} e^{-j\frac{2\pi d \sin(\phi-\phi_0)}{\lambda}} P(\phi - \phi_0) d\phi$$

$$= R_{XX}(d, \phi_0) + j R_{YX}(d, \phi_0)$$

Spatial correlation between antenna elements depends mainly on the mean AoA and PAS as well as antenna spacing d. When AS of PAS is small, most sub-rays that compose each path arrive at each antenna from the same angle. It implies that they are correlated with each other since while the magnitudes of two signals become nearly equal, their phases are different by their AoAs.
Since channel capacity and diversity gain decrease as the correlation between the antenna elements increases, antenna spacing must be set large enough to reduce the correlation.

A pattern of PAS depends mainly on the distribution of the locally scattered components. In general, enormous amounts of locally scattered components are observed by the MS in all different environments. Therefore, its PAS usually follows a uniform distribution. For the BS, however, the different PAS distributions are observed depending on the characteristics of terrain in a cell, which is usually shown to have a small AS. Note that they still show a uniform PAS distribution for the BS in picocells or indoor environments. PAS models for different environments can be found in the following Figure 2-25

		BS	MS
Outdoor	Macrocell	• Truncated Laplacian • *n*-th power of a cosine function • Truncated Gaussian • Uniform	Uniform
	Microcell		
	Picocell	Almost Uniform	
Indoor		Uniform	

Figure 2-44 PAS model for the different environments

2.6.2 I-METRA MIMO Channel Model

2.6.2.1 Statistical Model of Correlated MIMO Fading Channel

Consider a MIMO system with a base station of M antennas and the mobile stations of N antennas as illustrated in Figure 3.10. A narrowband MIMO channel H can be statistically expressed with an $M \times N$ matrix (i.e., $H \in \mathbb{C}^{M \times N}$) as

$$H = \Theta_R^{\frac{1}{2}} A_{\text{iid}} \Theta_T^{\frac{1}{2}}$$

where Θ_R and Θ_T are the correlation matrices for the receive antennas and transmit antennas, respectively, while A_{iid} represents an i.i.d. (independent and identically distributed) Rayleigh fading channel. The basic assumption behind the correlation matrix-based MIMO channel model in the previous equation is that the correlation matrices for the transmitter and receiver can be separated. That particular assumption holds when antenna spacing in the transmitter and receiver is sufficiently smaller than a distance between the transmitter and receiver, which is usually true for most of wireless communication environments

Now, a broadband MIMO channel can be modeled by a tapped delay line (TDL), which is an extension of the narrowband MIMO channel, as follows:

$$H(\tau) = \sum_{l=1}^{L} A_l \, \delta(\tau - \tau_l)$$

where A_l is the complex channel gain matrix for the lth path with delay τ_l. Let $\alpha_{mn}^{(l)}$ be the channel coefficient between the mth BS antenna and the nth MS antenna for the lth path. Assume that $\alpha_{mn}^{(l)}$ is zero-mean complex

Gaussian-distributed, and thus, $\left|\alpha_{mn}^{(l)}\right|$ is Rayleigh-distributed. The complex channel gain matrix A_l in the previous equation is given as

$$A_l = \begin{bmatrix} \alpha_{11}^{(l)} & \alpha_{12}^{(l)} & \cdots & \alpha_{1N}^{(l)} \\ \alpha_{21}^{(l)} & \alpha_{22}^{(l)} & \cdots & \alpha_{2N}^{(l)} \\ \vdots & \vdots & \ddots & \vdots \\ \alpha_{M1}^{(l)} & \alpha_{M2}^{(l)} & \cdots & \alpha_{MN}^{(l)} \end{bmatrix}$$

Let $y_m(t)$ denote the received signal at the mth antenna element in BS. Then, the received signals at the BS antenna are denoted as $y(t) = [y_1(t), y_2(t), \dots, y_M(t)]^T$. Similarly, the transmitted signals at the MS are denoted as $x(t) = [x_1(t), x_2(t), \dots, x_N(t)]^T$ where $x_n(t)$ is the signal transmitted at the nth antenna element. The relation between the MS and BS signals can be expressed as:

$$y(t) = \int H(\tau)s(t-\tau)d\tau$$

The correlation coefficient of a channel gain for two different MS antennas, n_1 and n_2 can be obtained using the Pearson correlation coefficient:

$$\rho_{n_1,n_2}^{\text{MS}} = \left\langle \left|\alpha_{mn_1}^{(l)}\right|^2, \left|\alpha_{mn_2}^{(l)}\right|^2 \right\rangle, \quad m = 1,2,\dots,M$$

We define a symmetric spatial correlation matrix for the MS as:

$$R_{\text{MS}} = \begin{bmatrix} \rho_{11}^{\text{MS}} & \rho_{12}^{\text{MS}} & \cdots & \rho_{1N}^{\text{MS}} \\ \rho_{21}^{\text{MS}} & \rho_{22}^{\text{MS}} & \cdots & \rho_{2N}^{\text{MS}} \\ \vdots & \vdots & \ddots & \vdots \\ \rho_{N1}^{\text{MS}} & \rho_{N2}^{\text{MS}} & \cdots & \rho_{NN}^{\text{MS}} \end{bmatrix}$$

where $\rho_{ij}^{\text{MS}} = \rho_{ji}^{\text{MS}}$, $i,j = 1,2,\dots,N$. Note that a diagonal component of R_{MS} corresponds to the autocorrelation, which is always given by a correlation coefficient of one (i.e., $\rho_{ii}^{\text{MS}} = 1$, $i = 1,2,\dots,N$).

The correlation coefficient of a channel gain for two different BS antennas, m_1 and m_2 can be obtained using the Pearson correlation coefficient:

$$\rho_{m_1m_2}^{\text{BS}} = \left\langle \left|\alpha_{m_1n}^{(l)}\right|^2, \left|\alpha_{m_2n}^{(l)}\right|^2 \right\rangle, \quad n = 1,2,\dots,N$$

We define the spatial correlation matrix for the BS as

$$R_{\text{BS}} = \begin{bmatrix} \rho_{11}^{\text{BS}} & \rho_{12}^{\text{BS}} & \cdots & \rho_{1M}^{\text{BS}} \\ \rho_{21}^{\text{BS}} & \rho_{22}^{\text{BS}} & \cdots & \rho_{2M}^{\text{BS}} \\ \vdots & \vdots & \ddots & \vdots \\ \rho_{M1}^{\text{BS}} & \rho_{M2}^{\text{BS}} & \cdots & \rho_{MM}^{\text{BS}} \end{bmatrix}$$

which is again a symmetric matrix with the unit diagonal components. Note that the correlation coefficients, $\{\rho_{n_1n_2}^{\text{MS}}\}$ and $\{\rho_{m_1m_2}^{\text{BS}}\}$, can be analytically determined by the spatial correlation function for the given PAS model, for example, a Uniform PAS model.

To generate the channel gain matrix A_l, information on the channel correlation between the Tx and Rx antennas is required. In fact, the spatial correlation matrices at BS and MS, R_{BS} and R_{MS}, do not provide sufficient information to generate A_l. In fact, the correlation of coefficients between the pairs of Tx and Rx, $\alpha_{m_1m_2}^{(l)}$, is additionally required, i.e.,

$$\rho_{n_1n_2,m_1m_2}^{\text{H}} = \left\langle \left|\alpha_{m_1n_1}^{(l)}\right|^2, \left|\alpha_{m_1n_2}^{(l)}\right|^2 \right\rangle$$

where $n_1 \neq n_2$ and $m_1 \neq m_2$. In general, there is no known theoretical solution to the previous equation. However, it can be approximated as

$$\rho^{\mathrm{H}}_{n_1 n_2, m_1 m_2} \approx \rho^{\mathrm{MS}}_{n_1 n_2} \rho^{\mathrm{BS}}_{m_1 m_2}$$

under the assumption that the average power of the channel coefficients are the same for all paths.

2.6.2.2 Generation of Correlated MIMO Channel Coefficients

Let the MIMO fading channel for the lth path be represented by an $MN \times 1$ vector

$$\mathbf{a}_l = \left[a_{11}^{(l)}, a_{21}^{(l)}, \ldots, a_{M1}^{(l)}, a_{12}^{(l)}, \ldots, a_{MN}^{(l)}\right]^T$$

which is a vector-form representation of the uncorrelated MIMO channel gain matrix $\mathbf{A_I}$. Here, $a_{ij}^{(l)}$ is a complex Gaussian random variable with a zero mean such that $E\left\{\left|a_x^{(l)}\right|^2\right\} = 1$ and $\left\langle \left|a_{x_1}^{(l_1)}\right|^2, \left|a_{x_2}^{(l_2)}\right|^2 \right\rangle = 0$ for $x_1 \neq x_2$ or $l_1 \neq l_2$ (i.e., $\left\{a_x^{(l)}\right\}$ are the uncorrelated channel coefficients). The correlated MIMO channel coefficients can now be generated by multiplying the uncorrelated MIMO fading channel vector by an $MN \times MN$ matrix C, which is referred to as the following correlation-shaping matrix or symmetric mapping matrix, that is,

$$\widetilde{\mathbf{A}}_l = \sqrt{P_l}\mathbf{C}\mathbf{a}_l$$

where P_l is the average power of the lth path. Note that $\widetilde{\mathbf{A}}_l$ is the correlated $MN \times 1$ MIMO channel vector with the correlated MIMO fading channel coefficients, given as

$$\widetilde{\mathbf{A}} = \left[\alpha_{11}^{(l)}, \alpha_{21}^{(l)}, \ldots, \alpha_{M1}^{(l)}, \alpha_{22}^{(l)}, \ldots, \alpha_{M2}^{(l)}, \alpha_{13}^{(l)}, \ldots, \alpha_{MN}^{(l)}\right]^T$$

In fact, the correlation-shaping matrix C defines the spatial correlation coefficients. A spatial correlation matrix is given as

$$\mathbf{R} = \begin{cases} R_{\mathrm{BS}} \otimes R_{\mathrm{MS}}: & \text{downlink} \\ R_{\mathrm{MS}} \otimes R_{\mathrm{BS}}: & \text{uplink} \end{cases}$$

where $\otimes$ denotes the Kronecker product. Using R, a root-power correlation matrix Γ is given as

$$\Gamma = \begin{cases} \sqrt{\mathbf{R}}, & \text{for field type} \\ \mathbf{R}, & \text{for complex type} \end{cases}$$

where Γ is a non-singular matrix which can be decomposed into a symmetric mapping matrix (or correlation-shaping matrix) with the Cholesky, or square-root decomposition as follows:

$$\Gamma = \mathbf{C}\mathbf{C}^T$$

Note that $\mathbf{C}$ can be obtained by Cholesky decomposition or square-root decomposition, depending on whether R_{BS} and R_{MS} are given as the complex matrices or real matrices, respectively.

2.6.2.3 I-METRA MIMO Channel Model

I-METRA (Intelligent Multi-element Transmit and Receive Antennas) MIMO channel model is based on the stochastic MIMO channel model we discussed, which generates a correlated MIMO fading channel using the spatial correlation derived for ULA (Uniform Linear Array) subject to a single or multiple clusters with the Uniform Truncated Gaussian, or Truncated Laplacian PAS.

The overall procedure for the I-METRA MIMO channel modelling consists of two main steps as shown in Figure 2-26. In the first step, the BS and MS spatial correlation matrices $\mathbf{R}_{\mathrm{BS}}$,$\mathbf{R}_{\mathrm{MS}}$, R and normalization factor are determined for the specified channel configuration, including the number of BS and MS antennas, antenna spacing, the number of clusters, PAS, AS, and AoA. In the second step, a symmetric mapping matrix C is found

and then, the correlated fading MIMO channel is generated by multiplying it by the power per path and the uncorrelated fading signal.

Once the correlated MIMO channel coefficients are generated for each path, the overall MIMO channel is simulated by using a tapped delay line.

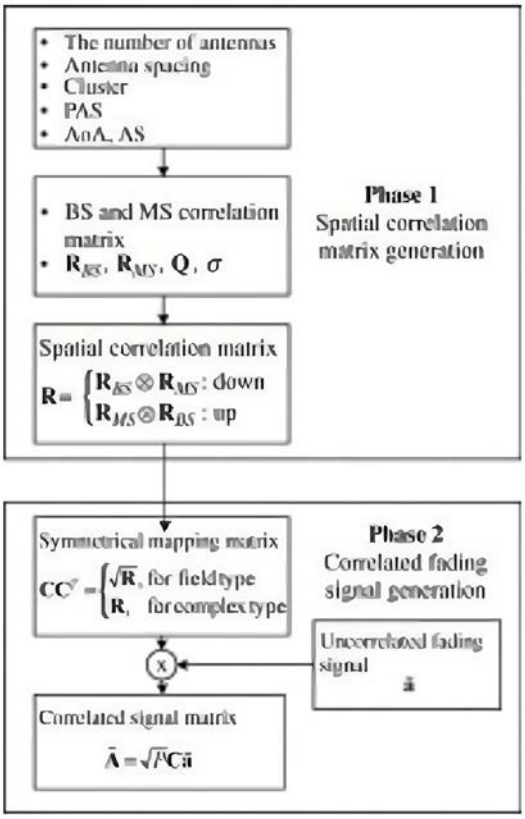


Figure 2-45 I-METRA MIMO channel modeling procedure: an overview.

Figure 2-27 shows a functional block diagram that implements the overall MIMO channel characteristics, taking the delay and power profiles into account. Here, an uncorrelated fading channel is generated with the pre-stored Doppler spectrum. It is multiplied by a spatial correlation mapping matrix to generate a correlated fading channel. The given PDP characteristics are implemented by passing the correlated fading signal through a FIR filter that is designed to satisfy the given average power and delay specification for each path. Furthermore, antenna radio pattern can be adjusted by generating a steering matrix.

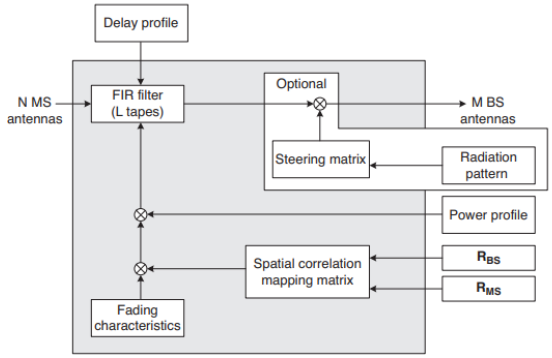


Figure 2-46 Functional block diagram for I-METRA MIMO channel model.

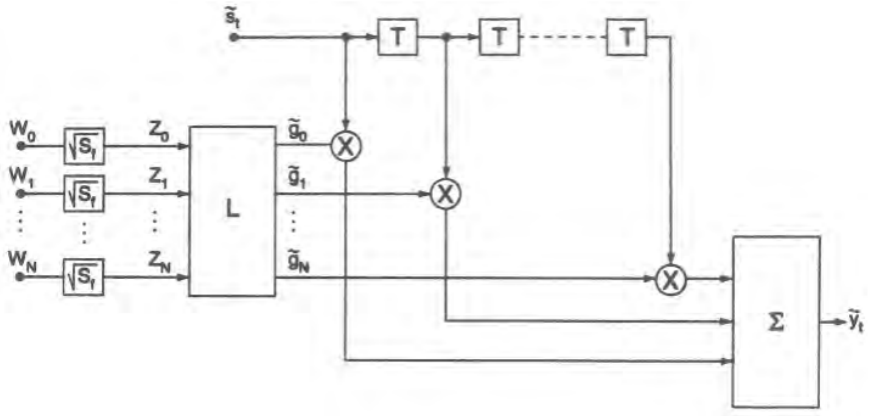


Figure 2-47 FIR filter block diagram for I-METRA MIMO channel model.

2.7 A REVIEW OF GENERATIVE ADVERSARIAL NETWORKS (GANS)

2.7.1 Basic Setup

Generative adversarial network (GAN) has shown great results in many generative tasks to replicate the real-world rich content such as images, human language, and music. It is inspired by game theory: two models, a generator and a critic, are competing while making each other stronger at the same time.

Specifically, Generative Adversarial Networks (GANs) are modeled as a two-player zero-sum minimax game, where the generator and discriminator optimize competing goals, aiming to reach a Nash equilibrium where the generator's ability to create indistinguishable data is countered by the discriminator's ability to classify real versus generated data.

However, it is rather challenging to train a GAN model, as people are facing issues like training instability or failure to converge.

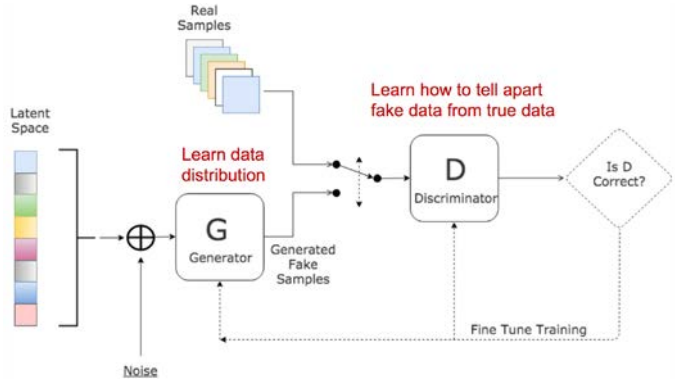


Figure 2-48 Architecture of a generative adversarial network.

On one hand, we want to make sure the discriminator D's decisions over real data are accurate by maximizing $\mathbb{E}_{x\sim p_r(x)}[\log D(x)]$. Meanwhile, given a fake sample $G(z)$, $z \sim p_z(z)$, the discriminator is expected to output a probability, $D(G(z))$, close to zero by maximizing $\mathbb{E}_{z\sim p_z(z)}\left[\log\left(1 - D(G(z))\right)\right]$.

On the other hand, the generator is trained to increase the chances of D producing a high probability for a fake example, thus, to minimize $\mathbb{E}_{z\sim p_z(z)}\left[\log\left(1 - D(G(z))\right)\right]$.

When combining both aspects together, D and G are playing a minimax game in which we should optimize the following loss function:

$$\min_G \max_D L(D,G) = \mathbb{E}_{x\sim p_r(x)}[\log D(x)] + \mathbb{E}_{z\sim p_z(z)}\left[\log\left(1 - D\big(G(z)\big)\right)\right]$$

$$= \mathbb{E}_{x\sim p_r(x)}[\log D(x)] + \mathbb{E}_{x\sim p_g(x)}\big[\log\big(1 - D(x)\big)\big]$$

$\big(\mathbb{E}_{x\sim p_r(x)}[\log D(x)]\big)$ has no impact on G during gradient descent updates.)

2.7.2 Distance Metrics

Before we start examining GANs closely, let us first review two metrics for quantifying the similarity between two probability distributions, as they will be instrumental to understand GANs' loss function.

The Total Variation (TV) distance is:

$$\delta\big(P_r, P_g\big) = \sup_A \big|P_r(A) - P_g(A)\big|$$

The Kullback-Leibler (KL) divergence is:

$$KL\big(P_r \parallel P_g\big) = \int_x \log\left(\frac{P_r(x)}{P_g(x)}\right) P_r(x)\, dx$$

This isn't symmetric. The reverse KL divergence is defined as: $KL\big(P_g \parallel P_r\big)$

The Jenson-Shannon (JS) divergence: Let M be the mixture distribution $M = P_r/2 + P_g/2$ then:

$$JS\big(P_r, P_g\big) = \frac{1}{2}KL(P_r \parallel P_m) + \frac{1}{2}KL\big(P_g \parallel P_m\big)$$

This metric is bounded by $[0,1]$ and symmetric

The Earth Mover (EM) or Wasserstein distance: Let $\Pi\big(P_r, P_g\big)$ be the set of all joint distributions γ whose marginal distributions are P_r and P_g then:

$$W\big(P_r, P_g\big) = \inf_{\gamma\in\Pi\left(P_r,P_g\right)} \mathbb{E}_{(x,y)\sim\gamma}[\parallel x - y \parallel]$$

This metric will be covered in greater detail in an upcoming section.

2.7.3 Proof of Optimality

Now we have a well-defined loss function. Let's first examine what is the best value for D.

$$L(G,D) = \int_x \Big(p_r(x)\log\big(D(x)\big) + p_g(x)\log\big(1 - D(x)\big)\Big)\, dx$$

Since we are interested in what is the best value of $D(x)$ to maximize $L(G,D)$, let us label

$$\hat{x} = D(x), \quad A = p_r(x), \quad B = p_g(x)$$

And then what is inside the integral (we can safely ignore the integral because x is sampled over all the possible values) is:

$$f(\hat{x}) = A\log\hat{x} + B\log(1 - \hat{x})$$

$$\frac{df(\hat{x})}{d\hat{x}} = \frac{A}{\hat{x}\ln 10} - \frac{B}{(1-\hat{x})\ln 10}$$

$$= \frac{1}{\ln 10}\left(\frac{A}{\hat{x}} - \frac{B}{1-\hat{x}}\right)$$

$$= \frac{1}{\ln 10}\frac{A-(A+B)\hat{x}}{\hat{x}(1-\hat{x})}$$

Thus, set $\frac{df(\hat{x})}{d\hat{x}} = 0$, we get the best value of the discriminator:

$$D^*(x) = \hat{x}^* = \frac{A}{A+B} = \frac{p_r(x)}{p_r(x)+p_g(x)} \in [0,1].$$

Once the generator is trained to its optimal, p_g gets very close to p_r. When $p_g = p_r$, $D^*(x)$ becomes $1/2$. When both G and D are at their optimal values, we have $p_g = p_r$ and $D^*(x) = 1/2$ and the loss function becomes:

$$L(G, D^*) = \int_x \left(p_r(x)\log\big(D^*(x)\big) + p_g(x)\log\big(1 - D^*(x)\big)\right) dx$$

$$= \log\frac{1}{2}\int_x p_r(x)dx + \log\frac{1}{2}\int_x p_g(x)dx$$

$$= -2\log 2$$

According to the formula listed in the previous section, JS divergence between p_r and p_g can be computed as:

$$D_{JS}\big(p_r \| p_g\big) = \frac{1}{2}D_{KL}\left(p_r \| \frac{p_r + p_g}{2}\right) + \frac{1}{2}D_{KL}\left(p_g \| \frac{p_r + p_g}{2}\right)$$

$$= \frac{1}{2}\left(\log 2 + \int_x p_r(x)\log\frac{p_r(x)}{p_r(x)+p_g(x)}dx\right) + \frac{1}{2}\left(\log 2 + \int_x p_g(x)\log\frac{p_g(x)}{p_r(x)+p_g(x)}dx\right)$$

$$= \frac{1}{2}\big(\log 4 + L(G, D^*)\big)$$

Thus,

$$L(G, D^*) = 2D_{JS}\big(p_r \| p_g\big) - 2\log 2$$

Essentially the loss function of GAN quantifies the similarity between the generative data distribution p_g and the real sample distribution p_r by JS divergence when the discriminator is optimal. The best G^* that replicates the real data distribution leads to the minimum $L(G^*, D^*) = -2\log 2$ which is aligned with equations above.

2.7.4 Problems in GANs

2.7.4.1 Mode Collapse

During the training, the generator may collapse to a setting where it always produces the same outputs. This is a common failure case for GANs, commonly referred to as Mode Collapse. Even though the generator might be able to trick the corresponding discriminator, it fails to learn to represent the complex real-world data distribution and gets stuck in a small space with extremely low variety.

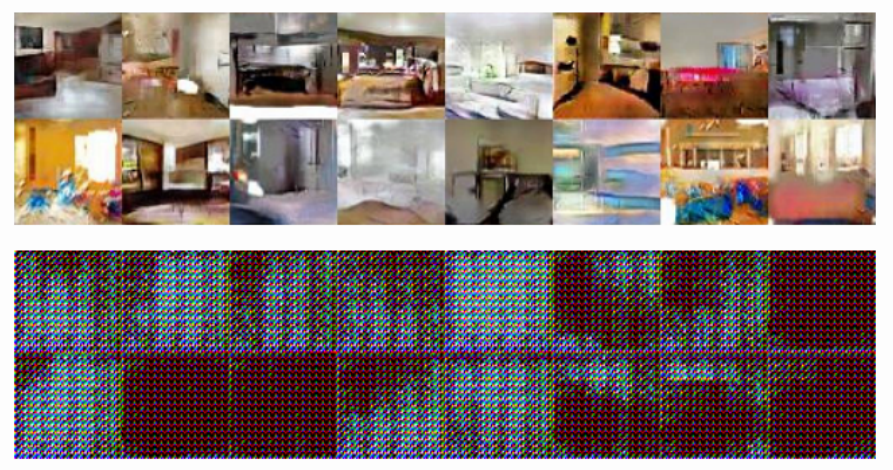


Figure 2-49 Top: proper GAN generated samples. Bottom: GAN generated samples with mode collapse.

2.7.4.2 Vanishing Gradients

When the discriminator is perfect, we are guaranteed with $D(x) = 1, \forall x \in p_r$ and $D(x) = 0, \forall x \in p_g$. Therefore, the loss function L falls to zero and we end up with no gradient to update the loss during learning iterations. Figure 2-31 demonstrates an experiment when the discriminator gets better, the gradient vanishes fast.

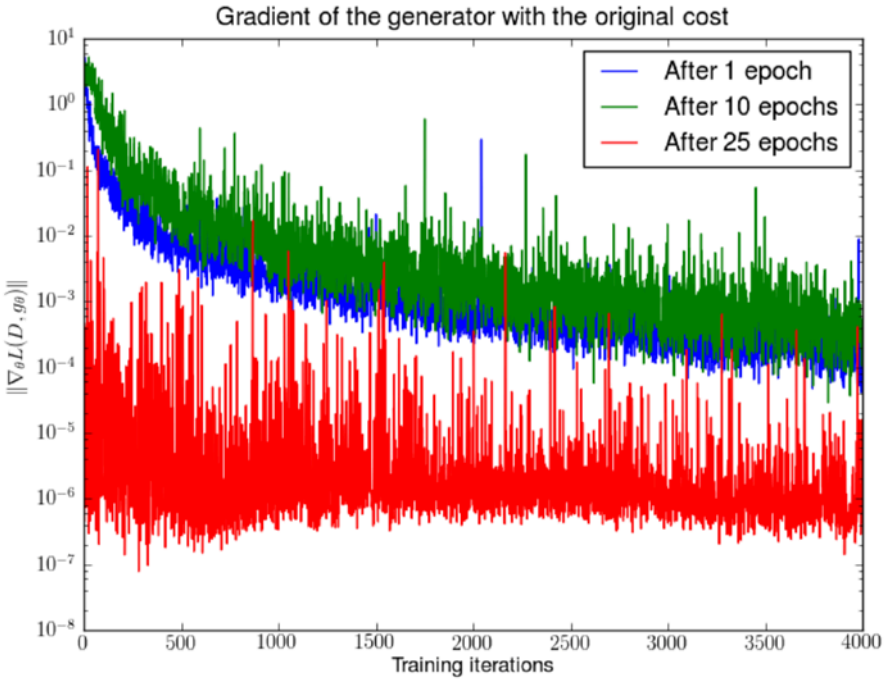


Figure 2-50 Gradient of generator

First, a DCGAN is trained for 1, 10 and 25 epochs. Then, with the generator fixed, a discriminator is trained from scratch and measures the gradients with the original cost function. We see the gradient norms decay quickly (in log scale), in the best case 5 orders of magnitude after 4000 discriminator iterations.

2.7.4.3 Low-dimensional Supports

The dimensions of many real-world datasets, as represented by p_r, only appear to be artificially high. They have been found to concentrate in a lower dimensional manifold. This is actually the fundamental assumption for

Manifold Learning. Thinking of the real-world images for example, once the theme or the contained object is fixed, the images have a lot of restrictions to follow, i.e., a dog should have two ears and a tail, and a skyscraper should have a straight and tall body, etc. These restrictions keep images aways from the possibility of having a high-dimensional free form.

p_g lies in a low dimensional manifold, too. Whenever the generator is asked to a much larger image like 64x64 given a small dimension, such as 100, noise variable input, the distribution of colors over these 4096 pixels has been defined by the small 100-dimension random number vector and can hardly fill up the whole high dimensional space.

Because both p_r and p_g rest in low dimensional manifolds, they are almost certainly gonna be disjoint Figure 2-32. When they have disjointed supports, it has been proven that we are always capable of finding a perfect discriminator that separates real and fake samples.

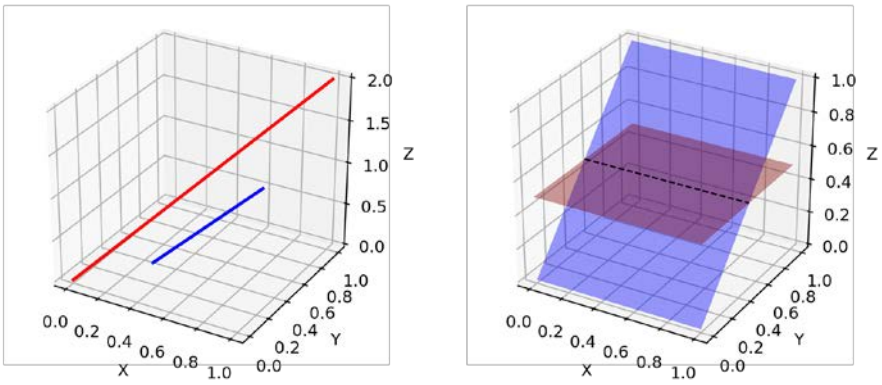


Figure 2-51 Low dimensional manifolds in high dimension space can hardly have overlaps. (Left) Two lines in a three-dimensional space. (Right) Two surfaces in a three-dimension space.

2.7.4.4 Lack of a proper evaluation metric

The JS divergence can lead to unstable training dynamics, including vanishing gradients and mode collapse, due to its inability to provide direct measures of distribution similarity and its assumption of overlapping supports between distributions. This issue presents itself evidently in the typical erratic loss curves of GANs. In addition, This fundamental flaw often results in the discriminator learning too quickly, which can halt the generator's learning process, especially in high-dimensional spaces where disjoint supports are more likely. These challenges underline the need for alternative metrics and approaches, such as the Wasserstein distance, which offers more stable and meaningful gradients for GAN training

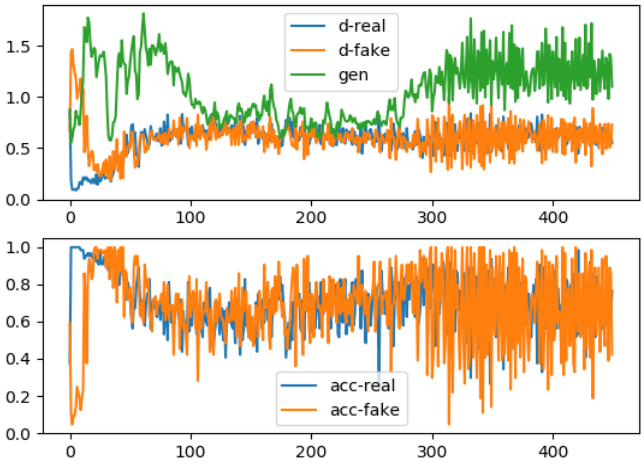


Figure 2-52 Typical GANs loss and accuracy curves.

2.7.5 Introduction to The Wasserstein Distance

For discrete probability distributions, the Wasserstein distance is also descriptively called the earth mover's distance (EMD). If we imagine the distributions as different heaps of a certain amount of earth, then the EMD is the minimal total amount of work it takes to transform one heap into the other. Work is defined as the amount of earth in a chunk times the distance it was moved. Let's call our discrete distributions P_r and P_θ, each with l possible states x or y respectively and take two arbitrary distributions as an example.

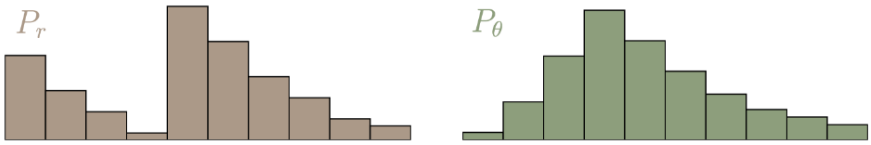


Figure 2-53 Probability distributions Pr and P_theta, each with ten states

Calculating the EMD is in itself an optimization problem: There are infinitely many ways to move the earth around, and we need to find the optimal one. We call the transport plan that we are trying to find $\gamma(x, y)$. It simply states how we distribute the amount of earth from one place x over the domain of y, or vice versa.

To be a valid transport plan, the constraints $\sum_x \gamma(x, y) = P_r(y)$ and $\sum_y \gamma(x, y) = P_\theta(x)$ must of course apply. This ensures that following this plan yields the correct distributions. Equivalently, we can call γ a joined probability distribution and require that $\gamma \in \Pi(P_r, P_\theta)$, where $\Pi(P_r, P_\theta)$ is the set of all distributions whose marginals are P_r or P_θ respectively. To get the EMD, we have to multiply every value of γ with the Euclidian distance between x and y. With that, the definition of the Earth mover's distance is:

$$\mathrm{EMD}(P_r, P_\theta) = \inf_{\gamma \in \Pi} \sum_{x,y} \| x - y \| \gamma(x, y) = \inf_{\gamma \in \Pi} E_{(x,y)\sim\gamma} \| x - y \|$$

The expression inf, it stands for *infimum*, or greatest lower bound. The opposite is sup or *supremum*. We can also set $\Gamma = \gamma(x, y)$ and $D = \| x - y \|$, with $\Gamma, D \in \mathbb{R}^{l \times l}$. Now we can write

$$\mathrm{EMD}(P_r, P_\theta) = \inf_{\gamma \in \Pi} \langle D, \Gamma \rangle_F$$

where $\langle \cdot, \cdot \rangle_F$ is the Frobenius inner product (sum of all the element-wise products).

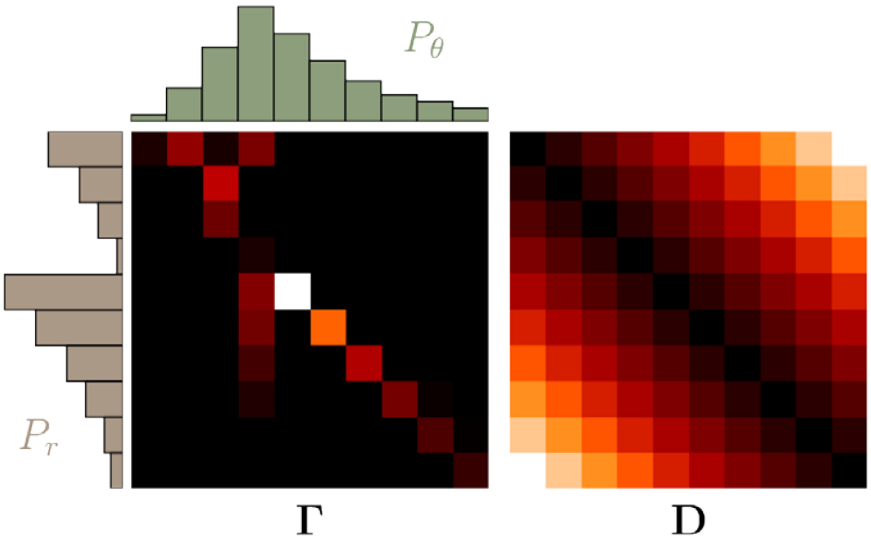


Figure 2-54 Transport plan with Pr and P_theta as marginal probabilities, and distances D

Here is a simple example to argue why we should care about the Earth-Mover distance.
Consider probability distributions defined over $\mathbb{R}^2$. Let the true data distribution be $(0, y)$, with y sampled uniformly from $U[0,1]$. Consider the family of distributions P_θ, where $P_\theta = (\theta, y)$, with y also sampled from $U[0,1]$.

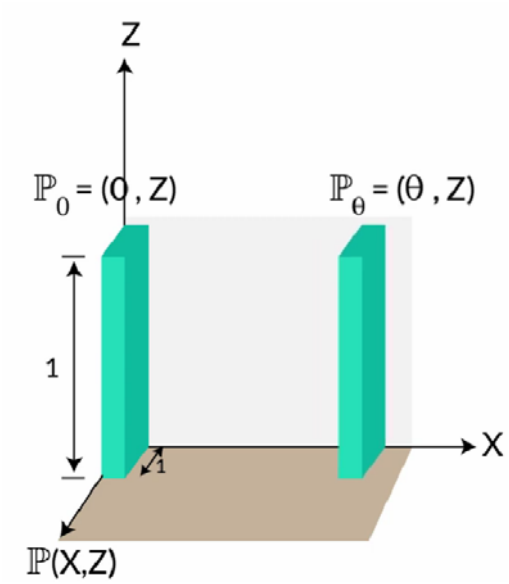


Figure 2-55 Real and Fake Distributions

We'd like our optimization algorithm to learn to move θ to 0, thus as $\theta \to 0$, the distance $d(P_0, P_\theta)$ should decrease. But for many common distance functions, this doesn't happen.

Total variation: For any $\theta \neq 0$, let $A = \{(0, y): y \in [0,1]\}$. This gives

$$\delta(P_0, P_\theta) = \begin{cases} 1 & \text{if } \theta \neq 0, \\ 0 & \text{if } \theta = 0. \end{cases}$$

KL divergence and reverse KL divergence: Recall that the KL divergence $KL(P \parallel Q)$ is $+\infty$ if there is any point (x, y) where $P(x, y) > 0$ and $Q(x, y) = 0$. For $KL(P_0 \parallel P_\theta)$, this is true at $(\theta, 0.5)$. For $KL(P_\theta \parallel P_0)$, this is true at $(0,0.5)$.

$$KL(P_0 \parallel P_\theta) = KL(P_\theta \parallel P_0) = \begin{cases} +\infty & \text{if } \theta \neq 0, \\ 0 & \text{if } \theta = 0, \end{cases}$$

Jenson-Shannon divergence: Consider the mixture $M = P_0/2 + P_\theta/2$, and now look at just one of the KL terms.

$$KL(P_0 \parallel M) = \int_{(x,y)} P_0(x, y) \log \frac{P_0(x, y)}{M(x, y)} \, dy \, dx$$

For any x, y where $P_0(x, y) \neq 0, M(x, y) = \frac{1}{2} P_0(x, y)$, so this integral works out to log 2. The same is true of $KL(P_\theta \parallel M)$, so the JS divergence is

$$JS(P_0, P_\theta) = \begin{cases} \log 2 & \text{if } \theta \neq 0, \\ 0 & \text{if } \theta = 0, \end{cases}$$

Earth Mover distance: Because the two distributions are just translations of one another, the best way transport plan moves mass in a straight line from $(0, y)$ to (θ, y). This gives $W(P_0, P_\theta) = |\theta|$

This example shows that there exist sequences of distributions that don't converge under the JS, KL, reverse KL, or TV divergence, but which do converge under the EM distance.

This example also shows that for the JS, KL, reverse KL, and TV divergence, there are cases where the gradient is always 0. This is especially bad from an optimization perspective - any approach that works by taking the gradient $\nabla_\theta d(P_0, P_\theta)$ will fail in these cases.

Admittedly, this is a contrived example because the supports are disjoint, but when the supports are low dimensional manifolds in high dimensional space, it's very easy for the intersection to be measure zero, which is enough to give similarly bad results.

2.7.6 Wasserstein GANs (WGANs)

2.7.6.1 Wasserstein Distance intractability & Lipschitz Continuity

Unfortunately, computing the Wasserstein distance exactly is intractable. Let's repeat the definition.

$$W(P_r, P_g) = \inf_{\gamma \in \Pi(P_r, P_g)} \mathbb{E}_{(x,y)\sim\gamma}[\| x - y \|]$$

A result from Kantorovich-Rubinstein duality shows W is equivalent to

$$W(P_r, P_g) = \sup_{\|f\|_L \leq 1} \mathbb{E}_{x\sim P_r}[f(x)] - \mathbb{E}_{x\sim P_g}[f(x)]$$

where the supremum is taken over all 1-Lipschitz functions.

Let d_X and d_Y be distance functions on spaces X and Y. A function $f: X \to Y$ is K-Lipschitz if for all $x_1, x_2 \in X$,

$$d_Y(f(x_1), f(x_2)) \leq K d_X(x_1, x_2)$$

Intuitively, the slope of a K-Lipschitz function never exceeds K.

If we replace the supremum over 1-Lipschitz functions with the supremum over K-Lipschitz functions, then the supremum is $K \cdot W(P_r, P_\theta)$ instead. (This is true because every K-Lipschitz function is a 1-Lipschitz function if you divide it by K, and the Wasserstein objective is linear.)

The supremum over K-Lipschitz functions $\{f: \| f \|_L \leq K\}$ is still intractable, but now it's easier to approximate. Suppose we have a parametrized function family $\{f_w\}_{w\in W}$, where w are the weights and W is the set of all possible weights. Further suppose these functions are all K-Lipschitz for some K. Then we have

$$\max_{w\in W} \mathbb{E}_{x\sim P_r}[f_w(x)] - \mathbb{E}_{x\sim P_\theta}[f_w(x)] \leq \sup_{\|f\|_L \leq K} \mathbb{E}_{x\sim P_r}[f(x)] - \mathbb{E}_{x\sim P_\theta}[f(x)] = K \cdot W(P_r, P_\theta)$$

For optimization purposes, we don't even need to know what K is! It's enough to know that it exists, and that it's fixed throughout the training process. Sure, gradients of W will be scaled by an unknown K, but they'll also be scaled by the learning rate α, so K will get absorbed into the hyper Param tuning.

If $\{f_w\}_{w\in W}$ contains the true supremum among K-Lipschitz functions, this gives the distance exactly. This probably won't be true. In that case, the approximation's quality depends on what K-Lipschitz functions are missing from $\{f_w\}_{w\in W}$.

2.7.6.2 WGAN Algorithm

Now, let's loop all this back to generative models. We'd like to train $P_\theta = g_\theta(Z)$ to match P_r. Intuitively, given a fixed g_θ, we can compute the optimal f_w for the Wasserstein distance. We can then backprop through $W(P_r, g_\theta(Z))$ to get the gradient for θ.

$$\nabla_\theta W(P_r, P_\theta) = \nabla_\theta\left(\mathbb{E}_{x\sim P_r}[f_w(x)] - \mathbb{E}_{z\sim Z}\left[f_w\left(g_\theta(z)\right)\right]\right) = -\mathbb{E}_{z\sim Z}\left[\nabla_\theta f_w\left(g_\theta(z)\right)\right]$$

The training process has now broken into three steps.

For a fixed θ, compute an approximation of $W(P_r, P_\theta)$ by training f_w to convergence.

Once we find the optimal f_w, compute the θ gradient $-\mathbb{E}_{z\sim Z}\left[\nabla_\theta f_w\left(g_\theta(z)\right)\right]$ by sampling several $z \sim Z$.
Update θ, and repeat the process.

There's one final detail. This entire derivation only works when the function family $\{f_w\}_{w \in W}$ is K-Lipschitz. To guarantee this is true, we use weight clamping. The weights w are constrained to lie within $[-c, c]$, by clipping w after every update to w.

The full algorithm is below.

Algorithm 1 WGAN, our proposed algorithm. All experiments in the paper used the default values $\alpha = 0.00005$, $c = 0.01$, $m = 64$, $n_{\text{critic}} = 5$.

Require: : α, the learning rate. c, the clipping parameter. m, the batch size. n_{critic}, the number of iterations of the critic per generator iteration.
Require: : w_0, initial critic parameters. θ_0, initial generator's parameters.

while θ has not converged **do**
  for $t = 0, ..., n_{\text{critic}}$ **do**
    Sample $\{x^{(i)}\}_{i=1}^{m} \sim \mathbb{P}_r$ a batch from the real data.
    Sample $\{z^{(i)}\}_{i=1}^{m} \sim p(z)$ a batch of prior samples.
    $g_w \leftarrow \nabla_w \left[\frac{1}{m}\sum_{i=1}^{m} f_w(x^{(i)}) - \frac{1}{m}\sum_{i=1}^{m} f_w(g_\theta(z^{(i)}))\right]$
    $w \leftarrow w + \alpha \cdot \text{RMSProp}(w, g_w)$
    $w \leftarrow \text{clip}(w, -c, c)$
  end for
  Sample $\{z^{(i)}\}_{i=1}^{m} \sim p(z)$ a batch of prior samples.
  $g_\theta \leftarrow -\nabla_\theta \frac{1}{m}\sum_{i=1}^{m} f_w(g_\theta(z^{(i)}))$
  $\theta \leftarrow \theta - \alpha \cdot \text{RMSProp}(\theta, g_\theta)$
end while

Figure 2-56 WGAN Algorithm

2.7.6.3 WGAN-GP Algorithm

The Gradient Penalty (GP) in Wasserstein GANs with Gradient Penalty (WGAN-GP) is a method introduced to improve the training stability and performance of WGANs by enforcing the Lipschitz constraint on the discriminator, also known as the critic. Here's how it works and why it's an improvement overweight clipping:

Enforcing the Lipschitz Constraint

The original WGAN formulation used weight clipping to enforce the Lipschitz constraint on the critic. This constraint is crucial because the theoretical foundation of WGAN relies on the critic being a 1-Lipschitz function. A function is 1-Lipschitz if the absolute value of its gradient does not exceed 1 everywhere.

However, weight clipping has several drawbacks, such as causing the weights to converge towards extreme values, which can lead to either vanishing or exploding gradients and generally poor training behaviour.

To address these issues, WGAN-GP introduces a gradient penalty as an alternative method to enforce the Lipschitz constraint more effectively. Instead of clipping the weights, WGAN-GP adds a penalty term to the critic's loss function. This term penalizes the norm of the gradient of the critic's output with respect to its input when the norm deviates from 1. The penalty term is defined as follows:

$$\text{Penalty} = \lambda(\| \nabla D(\hat{x}) \|_2 - 1)^2$$

where:
D is the critic.
$\hat{x}$ is sampled along straight lines between pairs of points sampled from the real data distribution and the generator distribution.
λ is a coefficient determining the strength of the penalty.
$\| \nabla D(\hat{x}) \|_2$ is the L2 norm of the gradient of the critic with respect to its inputs.

How Gradient Penalty Improves Over Weight Clipping

Training Stability and Performance: The gradient penalty encourages the critic to have gradients of norm close to 1 almost everywhere. This provides a more stable training process as it avoids the pathological behaviors induced by extreme weight values that are typical with weight clipping.

Gradient Behavior: Unlike weight clipping, which crudely forces weights within a fixed range and can lead to non-smooth critic functions, the gradient penalty allows for a more nuanced adjustment of weights. This helps maintain useful gradients throughout training, leading to better convergence properties.

Improved Quality of Generated Samples: By providing a smoother and more stable training process, WGAN-GP often results in higher quality of generated samples compared to traditional WGAN.
The full algorithm is in Figure 2-38

Algorithm 1 WGAN with gradient penalty. We use default values of $\lambda = 10$, $n_{\text{critic}} = 5$, $\alpha = 0.0001$, $\beta_1 = 0$, $\beta_2 = 0.9$.

Require: The gradient penalty coefficient λ, the number of critic iterations per generator iteration n_{critic}, the batch size m, Adam hyperparameters α, β_1, β_2.
Require: initial critic parameters w_0, initial generator parameters θ_0.

while θ has not converged **do**
  for $t = 1, ..., n_{\text{critic}}$ **do**
    for $i = 1, ..., m$ **do**
      Sample real data $\boldsymbol{x} \sim \mathbb{P}_r$, latent variable $\boldsymbol{z} \sim p(\boldsymbol{z})$, a random number $\epsilon \sim U[0, 1]$.
      $\tilde{\boldsymbol{x}} \leftarrow G_\theta(\boldsymbol{z})$
      $\hat{\boldsymbol{x}} \leftarrow \epsilon \boldsymbol{x} + (1 - \epsilon)\tilde{\boldsymbol{x}}$
      $L^{(i)} \leftarrow D_w(\tilde{\boldsymbol{x}}) - D_w(\boldsymbol{x}) + \lambda(\|\nabla_{\hat{\boldsymbol{x}}} D_w(\hat{\boldsymbol{x}})\|_2 - 1)^2$
    end for
    $w \leftarrow \text{Adam}(\nabla_w \frac{1}{m} \sum_{i=1}^{m} L^{(i)}, w, \alpha, \beta_1, \beta_2)$
  end for
  Sample a batch of latent variables $\{\boldsymbol{z}^{(i)}\}_{i=1}^{m} \sim p(\boldsymbol{z})$.
  $\theta \leftarrow \text{Adam}(\nabla_\theta \frac{1}{m} \sum_{i=1}^{m} -D_w(G_\theta(\boldsymbol{z})), \theta, \alpha, \beta_1, \beta_2)$
end while

Figure 2-57 WGAN-GP Algorithm

In summary, the gradient penalty in WGAN-GP provides a more effective and stable way to enforce the Lipschitz constraint on the critic, addressing the limitations associated with weight clipping and generally enhancing the performance and outcome of the generative model.

3 HARDWARE IMPLEMENTATION OF OFDM TRANSCEIVER

3.1 DESIGN METHODOLOGY

This chapter is adapted from [7]

3.1.1 Overview of 802.11 processor chain

The encoding process is composed of many detailed steps, which are described later, as noted below. The following overview intends to facilitate understanding the details of the convergence procedure:

a. Produce the PHY Preamble field, composed of 10 repetitions of a "short training sequence" and two repetitions of a "long training sequence", preceded by a guard interval (GI).

b. Produce the PHY header field from the RATE, LENGTH, and SERVICE fields by filling the appropriate bit fields. The RATE and LENGTH fields of the PHY header are encoded by a convolutional code at a rate of R = 1/2 and are subsequently mapped onto a single BPSK encoded OFDM symbol, denoted as the SIGNAL symbol. To facilitate a reliable and timely detection of the RATE and LENGTH fields, 6 zero tail bits are inserted into the PHY header. The encoding of the SIGNAL field into an OFDM symbol follows the same steps for convolutional encoding, interleaving, BPSK modulation, pilot insertion, Fourier transform, and prepending a GI as described subsequently for data transmission with BPSK-OFDM modulated at coding rate 1/2. The contents of the SIGNAL field are not scrambled.

c. Calculate from RATE field, the number of data bits per OFDM symbol (NDBPS), the coding rate (R), the number of bits in each OFDM subcarrier (NBPSC), and the number of coded bits per OFDM symbol (NCBPS). Depending on the modulation scheme used:

d. Append the PSDU to the SERVICE field. Extend the resulting bit string with zero bits (at least 6 bits) so that the resulting length is a multiple of NDBPS. The resulting bit string constitutes the DATA part of the packet

e. Initiate the scrambler with a pseudorandom nonzero seed and generate a scrambling sequence. A pseudorandom integer constrained such that the first 7 bits of the scrambling sequence are not all 0s; then set the scrambler state to these 7 bits and generate the remainder of the scrambling sequence. XOR the scrambling sequence with the extended string of data bits.

f. Replace the six scrambled zero bits following the data with six non-scrambled zero bits. (Those bits return the convolutional encoder to the zero state and are denoted as tail bits.)

g. Encode the extended, scrambled data string with a convolutional encoder (R = 1/2). Omit (puncture) some of the encoder output string (chosen according to "puncturing pattern") to reach the "coding rate" corresponding to the parameter RATE (Coding Rate R).

h. Divide the encoded bit string into groups of N_{CBPS} bits. Within each group, perform an "interleaving" (reordering) of the bits according to a rule corresponding to the parameter RATE.

i. Divide the resulting coded and interleaved data string into groups of N_{BPSC} bits. For each of the bit groups, convert the bit group into a complex number according to the modulation encoding tables.

j. Divide the complex number string into groups of 48 complex numbers. Each such group is associated with one OFDM symbol. In each group, the complex numbers are numbered 0 to 47 and mapped hereafter

into OFDM subcarriers numbered –26 to –22, –20 to –8, –6 to –1, 1 to 6, 8 to 20, and 22 to 26. The subcarriers –21, –7, 7, and 21 are skipped and, subsequently, used for inserting pilot subcarriers. The 0 subcarrier, associated with centre frequency, is omitted and filled with the value 0.

k. Four subcarriers are inserted as pilots into positions –21, –7, 7, and 21. The total number of the subcarriers is 52 (48 + 4).

l. For each group of subcarriers –26 to 26, convert the subcarriers to time domain using inverse Fourier transform. Prepend to the Fourier-transformed waveform a circular extension of itself thus forming a GI and truncate the resulting periodic waveform to a single OFDM symbol length by applying time domain windowing.

m. Append the OFDM symbols one after another, starting after the SIGNAL symbol describing the RATE and LENGTH fields

n. Upconvert the resulting "complex baseband" waveform to an RF according to the centre frequency of the operating channel and transmit.

3.1.2 Block diagram using LabVIEW and USRP

In this section, our objective is to verify the functionality of our proposed system and continue with the floating-point analysis for the entire signal processing chain. To achieve this, we will develop a behavioural block diagram using LabVIEW. This diagram will model the system's operation in a detailed and structured manner. Once constructed, we will validate the system's functionality by transmitting real-time signals using the Universal Software Radio Peripheral (USRP). This approach allows us to simulate and analyse the system's performance under realistic conditions, ensuring that each component operates correctly and integrates smoothly within the overall system. The use of USRP for real-time signal transmission provides a practical platform to test and refine the system, offering insights into its behaviour and performance in a live environment.

3.1.2.1 Transmitted block diagram

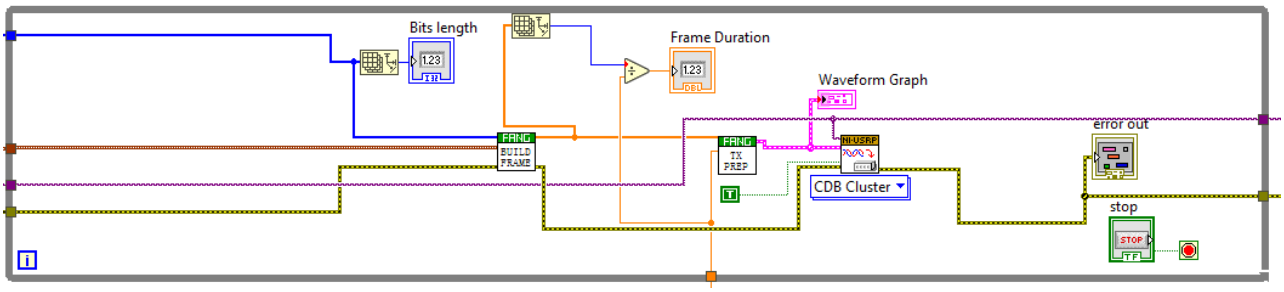


Figure 3-1 LabVIEW transmitter block diagram

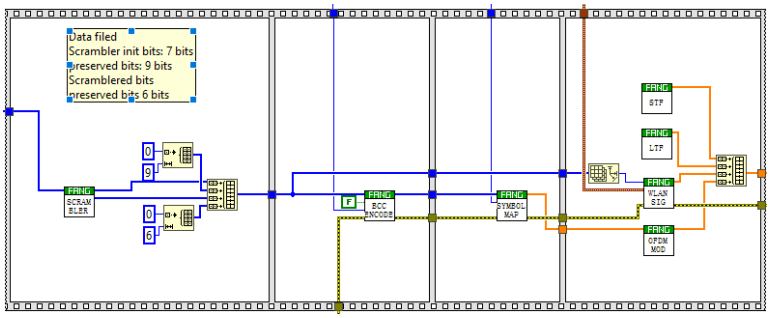


Figure 3-2 Build Frame block detailed view

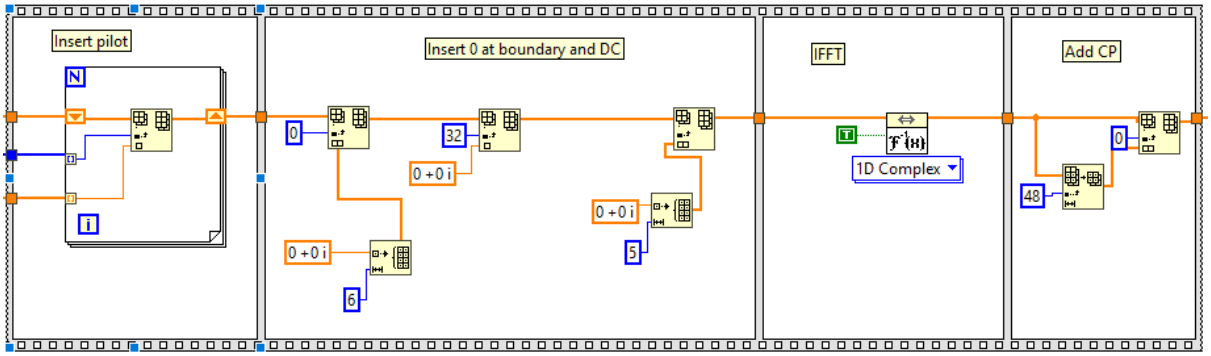


Figure 3-3 OFDM modulator detailed view

The block diagram illustrated in the figures above delineates the signal processing workflow employed in the transmitter to prepare and transmit signals. In Figure 3-1, the key component is the "BUILD FRAME" block, which performs several critical functions to ensure the signal is properly structured for transmission. This block initiates by scrambling the data to randomize the bit sequence, enhancing the robustness of the transmission. It then applies convolutional encoding, which adds redundancy to the data, facilitating error correction at the receiver end. Following this, the component frames the signal, organizing the data into a structured format suitable for transmission.

After framing, the component inserts preambles into the data symbols. Preambles are known sequences that help the receiver synchronize and demodulate the signal correctly. These operations are completed after the data symbols have been transformed into the time domain by the "OFDM MOD" (Orthogonal Frequency-Division Multiplexing Modulator) component. The OFDM MOD component also performs several vital tasks: it inserts four pilot symbols into the signal, which are used for channel estimation and correction at the receiver. Additionally, it places null subcarriers, which serve as guard bands to mitigate interference. The component arranges the samples for the Inverse Fast Fourier Transform (IFFT), converting the frequency-domain data back to the time domain. Lastly, it appends a cyclic prefix to each OFDM symbol to combat inter-symbol interference (ISI), ensuring the integrity of the transmitted signal.

Once these steps are completed, the resulting complex signal frame is sent to the National Instruments Universal Software Radio Peripheral (NI USRP). The NI USRP is responsible for reconstructing the signal and transmitting it over a Radio Frequency (RF) carrier. This process allows the signal to be wirelessly sent to a receiving USRP, where it will undergo a similar but reversed sequence of operations to recover the transmitted data. This detailed process ensures that the transmitted signal is robust, well-structured, and prepared for reliable transmission and reception over RF channels.

3.1.2.2 Receiver Block diagram

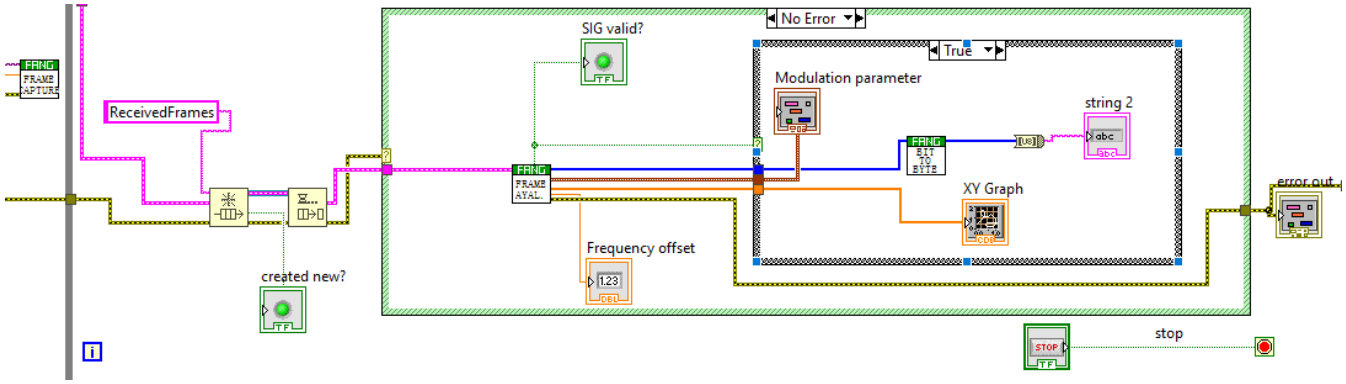


Figure 3-4 Receiver Block diagram

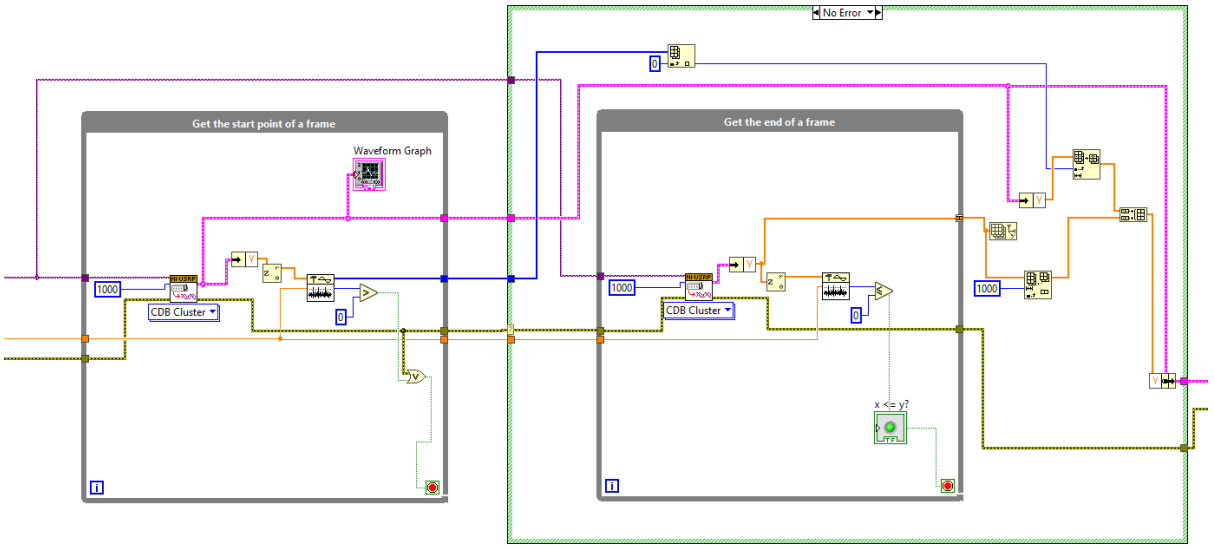


Figure 3-5 Frame capture block detailed view

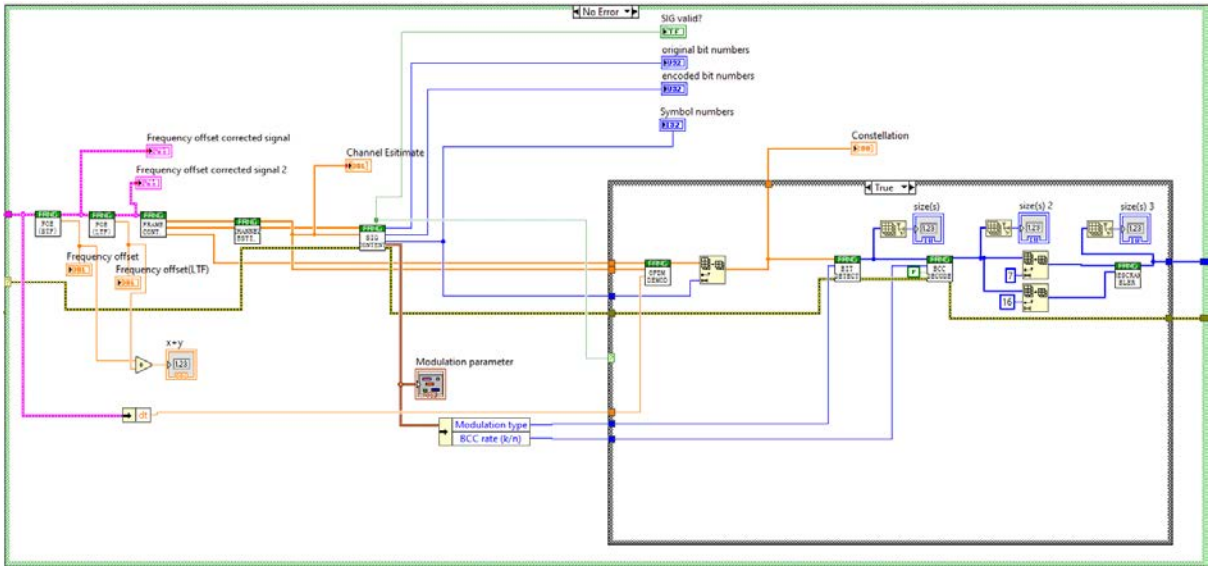


Figure 3-6 Frame analyzer block detailed view

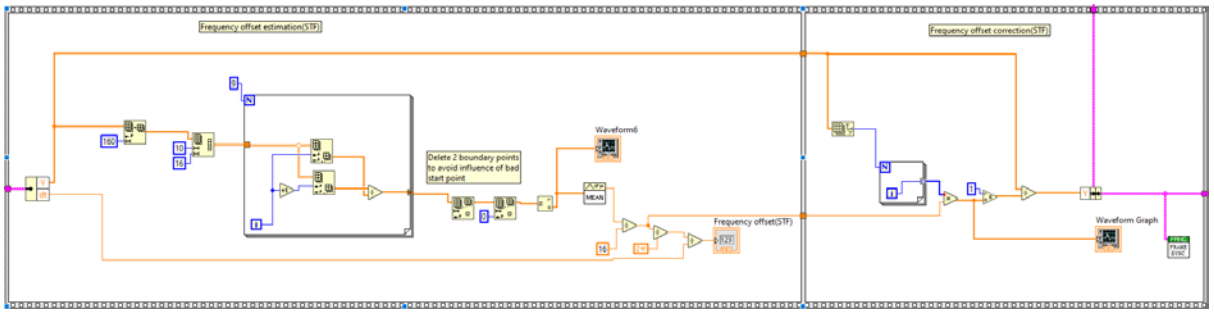


Figure 3-7 Frequency offset estimation detailed view

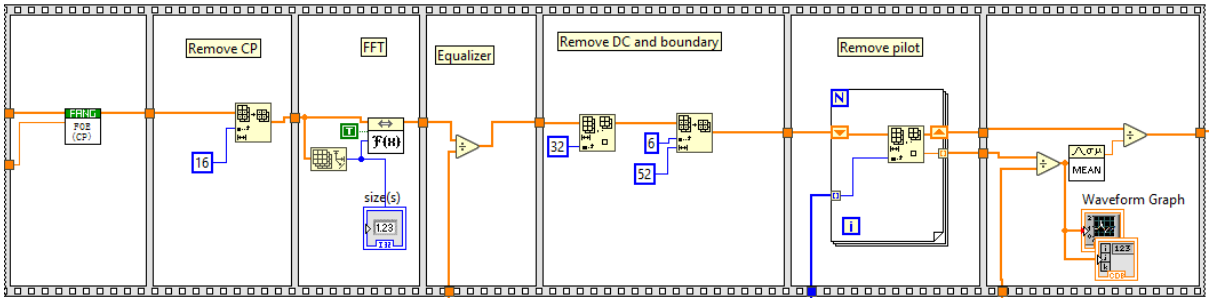


Figure 3-5 OFDM Demod detailed view

The block diagram depicted begins with the signal acquisition from the RF front end of the National Instruments Universal Software Radio Peripheral (NI USRP). This is the starting point where the received signal frames are initially captured.

1. **Frame Capture and Detection**:
   - The process starts in the "FRAME CAPTURE" block. This block's primary function is to monitor the incoming signal for a power level that exceeds a predefined threshold. When the signal power surpasses this trigger level, it indicates the presence of a frame.
   - Upon detecting a frame, the FRAME CAPTURE block initiates the capture process. Since the frame length is predetermined, the samples are buffered accordingly, allowing the system to recognize the frame's end once the expected number of samples is collected. This ensures that the entire frame is accurately captured for subsequent processing.
2. **Frequency Offset Correction**:
   - With the frame successfully captured, the next step involves extracting the short and long training sequences embedded in the frame. These sequences are crucial for correcting the frequency offset, a common issue in wireless communication that can distort the signal.
   - The short training sequence helps in coarse frequency offset correction, while the long training sequence provides fine correction and assists in accurate synchronization of the signal with the receiver's local oscillator.
3. **Channel Estimation and Equalization**:
   - Following the frequency offset correction, the receiver proceeds with channel estimation. Using the long training symbols, the system estimates the channel's characteristics. This is typically done by applying a least squares approach, such as Zero Forcing, to the known training symbols stored in memory.
   - The estimated channel impulse response (CIR) is then used to equalize the received OFDM symbols. Channel equalization compensates for the distortions introduced by the channel, ensuring that the signal is correctly interpreted by the receiver.
4. **Receiver Processing Chain**:
   - After channel equalization, the OFDM symbol processing continues with the removal of the cyclic prefix. This prefix, added during transmission to mitigate inter-symbol interference (ISI), is stripped off to restore the original signal length.
   - The system then applies the Fast Fourier Transform (FFT) to convert the signal from the time domain back to the frequency domain. This step is essential for demodulating the OFDM symbols.
   - Once in the frequency domain, the symbols are further equalized by dividing by the channel's frequency response, which was estimated earlier.
   - The next step involves filtering out the null subcarriers and pilot tones. Nulls serve as guard bands to prevent interference, while pilots are used for channel estimation and synchronization

purposes. Removing these ensures that only the relevant data-bearing subcarriers are processed.

- Finally, the processed symbols are demodulated to retrieve the transmitted data. This demodulation converts the received signal back into the original data symbols, completing the reception process.

By following this detailed processing chain, the receiver can effectively manage the complexities of wireless signal transmission and ensure accurate and reliable data recovery from the transmitted OFDM signal.

3.1.3 Floating-point modelling using python

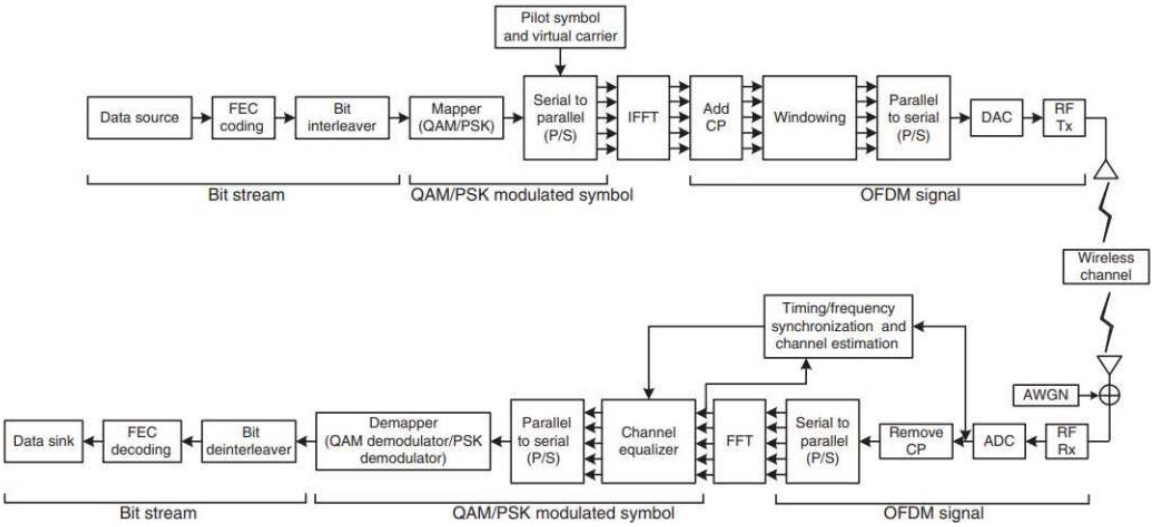


Figure 3-6 Structural floating point block diagram

To develop the floating-point model depicted in the figure above, we selected a specific implementation approach based on the methodologies discussed in Chapter 2: Literature Review. This chapter reviewed various techniques suitable for modeling and simulating digital communication systems. We chose an approach that best fit the requirements of our system, particularly for its precision and compatibility with our simulation tools. Following this, we constructed the floating-point model to replicate the behavior of the LabVIEW block diagrams. Each component, from signal acquisition through channel equalization and demodulation, was modeled using floating-point arithmetic to ensure high precision in representing real-world signal values.

To validate our model, we conducted a Monte Carlo analysis, running numerous simulations to understand the system's performance under varying conditions. This statistical approach enabled us to generate Bit Error Rate (BER) curves for our system, which we compared against the theoretical BER curves for a 4-QAM OFDM receiver. This comparison allowed us to assess the accuracy of our model. The floating-point model followed the detailed structure of the LabVIEW diagrams, ensuring that each block performed correctly and aligned with theoretical expectations. The results of this detailed process, including a thorough performance analysis and insights into system robustness, are discussed in Chapter 4: System Modelling Results. This approach confirmed the functionality of our system, demonstrating its reliability for further development and practical applications

3.1.4 Fixed point modelling using python

The design methodology we adopted progresses towards developing a hardware implementation and a fixed-point model of the system, crucial for ensuring correct functionality in the final, custom-designed hardware. After establishing the floating-point analysis, our next step is to transform this analysis into a fixed-point representation, a vital transition for real-world hardware applications. This process involves assessing the precision deviation between the floating-point model and the fixed-point model. As outlined in Chapter 2: Literature Review, it's essential that the Mean Squared Error (MSE) or Signal-to-Quantization Noise Ratio (SQNR) be on the order of 40 dB or better to ensure the system operates correctly after quantization. Such high precision is necessary to maintain the integrity of the signal processing and ensure that the transition from software simulations to hardware implementations does not introduce significant errors.

Furthermore, the hardware implementation, which we coded in Verilog, must closely match the precision and performance characteristics of the Python-based floating-point model. This bit-matching requirement ensures that the Verilog hardware design will faithfully replicate the behaviour of the floating-point model, allowing for accurate and reliable system performance in practical applications. Through this meticulous approach, we ensure that each stage of development, from floating-point simulation to fixed-point modelling and finally to hardware implementation, maintains high precision and adheres to the stringent performance criteria necessary for the system's success.

3.1.4.1 SQNR

Signal-to-Quantization Noise Ratio (SQNR) serves as a critical measure in digital signal processing to assess the accuracy of fixed-point representations compared to their floating-point counterparts. It quantifies the ratio between the power of the original signal and the power of the noise introduced during the quantization process. This noise arises due to the finite precision of fixed-point arithmetic, where the representation of real numbers is constrained to a limited number of bits. The SQNR calculation involves determining the average power of the original floating-point signal and the average power of the quantization noise, which is the difference between the floating-point signal and its quantized fixed-point version. The formula $\mathrm{SQNR} = 10 \cdot \log_{10}\left(\frac{\mathrm{P_{Float}}}{\mathrm{P_{Noise}}}\right)$ yields a value in decibels (dB), indicating how much stronger the original signal is relative to the noise introduced by quantization. Higher SQNR values, typically around 40 dB or more as recommended in literature, indicate a more accurate fixed-point representation with minimal distortion. By employing SQNR, we can validate and optimize fixed-point implementations to ensure they meet the required precision for robust and efficient signal processing applications in various domains, from telecommunications to digital audio processing.

The following equation can be used to measure SQNR:

$$\mathrm{P_{float}} = \left| \frac{1}{N} \sum_{i=1}^{N} x_i^2 \right|$$

$$\mathrm{P_{Noise}} = \left| \frac{1}{N} \sum_{i=1}^{N} (x_i - y_i)^2 \right|$$

$$\mathrm{SQNR} = 10 \cdot \log_{10}\left(\frac{\mathrm{P_{Float}}}{\mathrm{P_{Noise}}}\right)$$

x is the floating-point signal
y is the fixed-point signal

To measure the SQNR of the system, we must compare the floating-point analysis to that of the fixed-point analysis, use the SQNR equation and check for the deviation, if it is in order of 40 dB then the results are satisfactory, the figure below shows the SQNR of the system taken from the python simulations:

```
In [2]: SQNR(detectedSyms_c, detectedSyms_f_c)
Out[2]: 38.24499573765681
```

Here we created a python function called SQNR, it applies the equations mentioned above, we pass to it the detected symbols before demodulating it, the symbols before the demodulator are complex numbers, the subscript "_f" indicated that of the floating point and without the subscript indicated a fixed point. The SQNR of the system is 38.244 dB which is in the acceptable range of the precision needed

Another SQNR metric we need to prove is the deviation of the fixed-point python model, to the Verilog implementation, the 2 models must be bit matched in for the BER curves to be satisfactory and to have a fair comparison, we will apply the same SQNR function to measure the deviation of the fixed-point model to the Verilog implementation, the result must result to infinity which means the SQNR is very high they are equal, the following function is used to parse the data from the Verilog code and the SQNR function is used to calculate the deviation:

```
def test_sqnr(detectedSyms_c):

    x_r_v = np.loadtxt('out_v_r.txt', dtype=str, skiprows=3)

    x_i_v = np.loadtxt('out_v_i.txt', dtype=str, skiprows=3)

    xv = twos_complement_to_decimal(x_r_v)/2**9 + 1j * twos_complement_to_decimal(x_i_v)/2**9

    return SQNR(xv,detectedSyms_c)
```

The text files are the Verilog output vectors, it is sent to the python model for testing, when the function is run, this is the output:

```
In [11]: test_sqnr(detectedSyms_c)
c:\users\moham\downloads\digital_comm_2\digital_comm_2\fixed_pt.py:9: RuntimeWarning:
divide by zero encountered in scalar divide
  avg = 10 * np.log10(P_Float / P_Noise)
Out[11]: inf
```

In this scenario, the SQNR function encountered a divide-by-zero error because no quantization noise was present between the fixed-point models of the Python and Verilog implementations. This absence of noise is crucial for ensuring the proper functionality and verification of the hardware implementation. It simplifies the debugging process by eliminating potential discrepancies that could arise from quantization errors. This alignment between the Python simulation and Verilog hardware model allows for straightforward comparison and validation of performance metrics, such as Bit Error Rate (BER) curves, which will be thoroughly analysed in Chapter 4: System Modelling Results. These BER curves provide valuable insights into how well the hardware implementation replicates the expected behaviour of the Python model, affirming the accuracy and reliability of the system design across different stages of development.

In the upcoming chapters, we will delve into the intricacies of our hardware implementation, exploring the various algorithms and components integral to its design. This phase of our study will focus on detailing the practical realization of our system, highlighting the specific methodologies employed to ensure a streamlined and error-free design process.

The chapter preceding this has laid the groundwork by outlining the rigorous methodology adopted to guarantee the accuracy and reliability of our system at every stage of development. By discussing the hardware implementation in depth and elucidating the roles of different algorithms and components, we aim to provide a comprehensive understanding of how theoretical concepts are translated into functional hardware solutions, contributing to advancements in digital signal processing and communication technologies

3.1.4.2 Receiver performance metrics

Fixed point Comparison between HDL and floating-point model in MATLAB

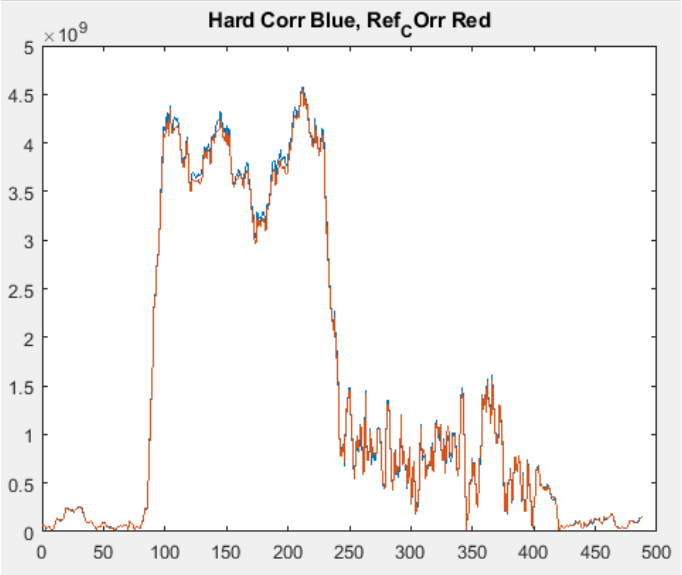


Figure 3-7 Autocorrelation path

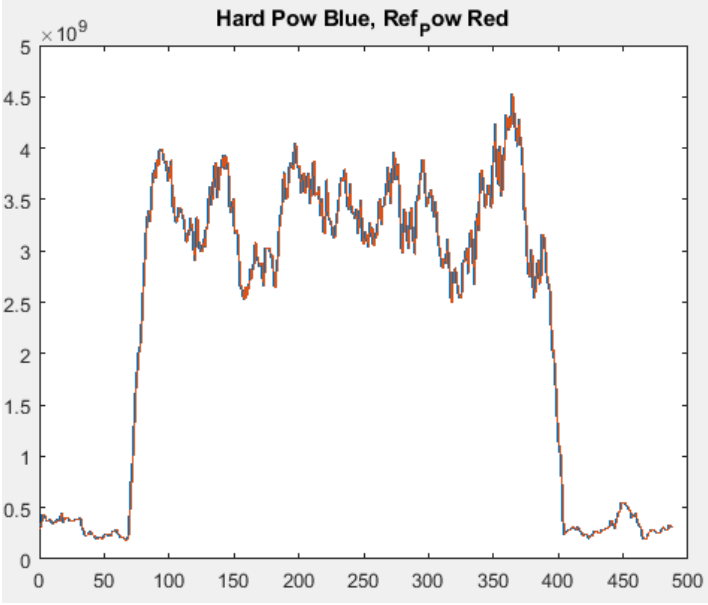


Figure 3-8 Cross correlation path

```
SQNR_Corr =

   35.1134


SQNR_Pow =

  104.7154
```

Figure 3-9 SQNR between fixed and floating points of synchronizer

The SQNR of correlation path is less than the power (or energy) path due to the usage of a magnitude estimator in the first path

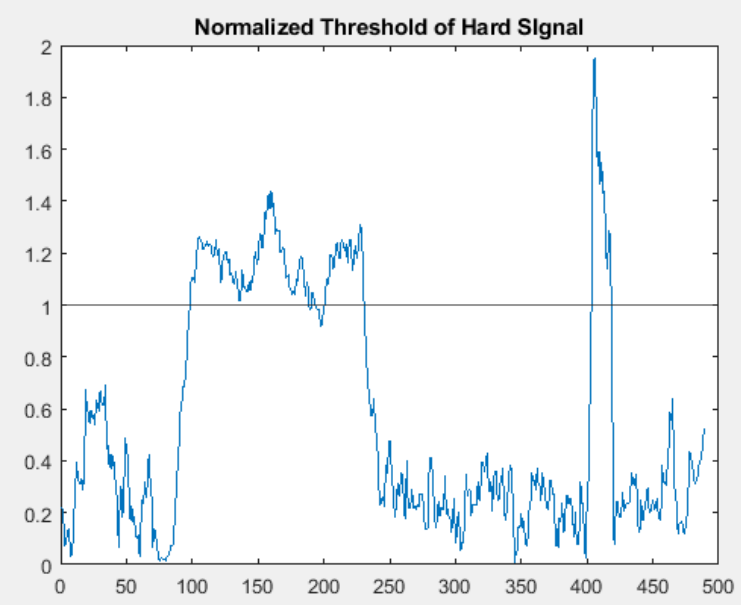


Figure 3-10 The resulting packet detection decision simulated in HDL

Comparison between performance of Correlation on HDL and Floating point

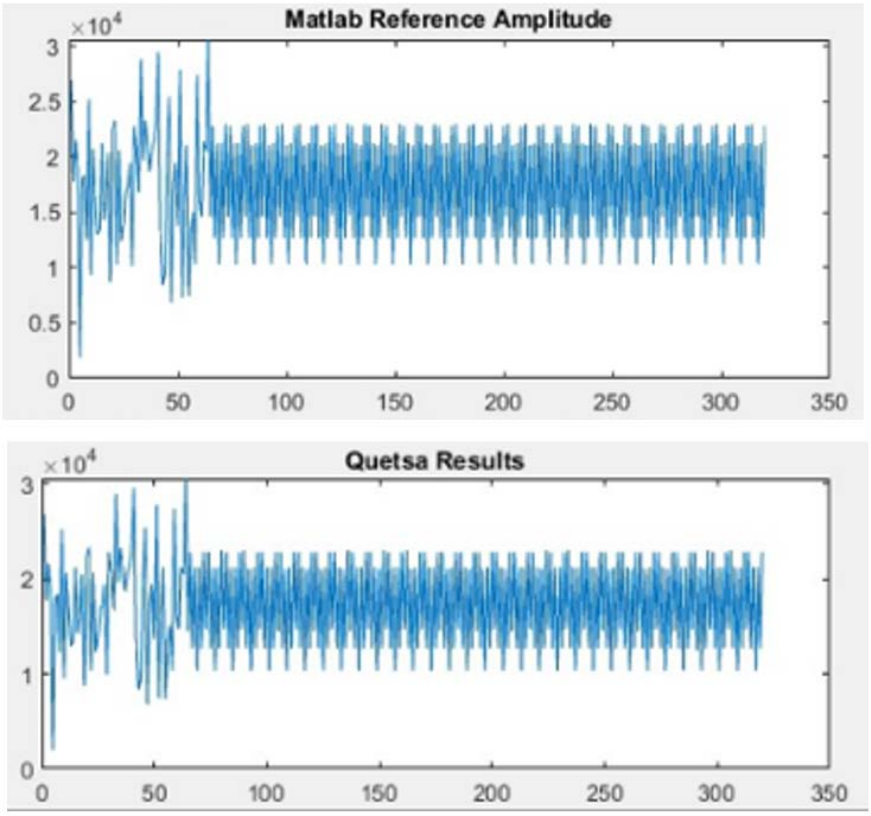


Figure 3-11 Magnitude Correspondence

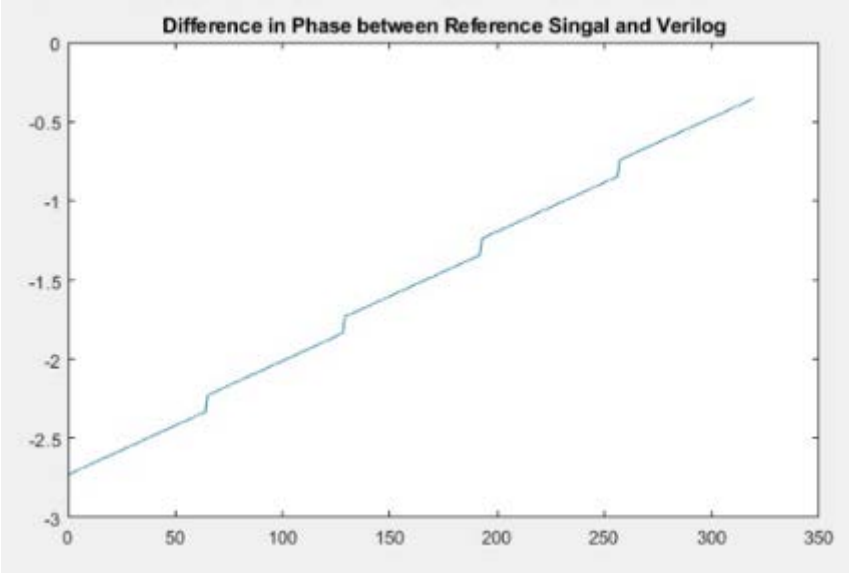


Figure 3-12 Phase difference

3.1.5 Modelling channel effects

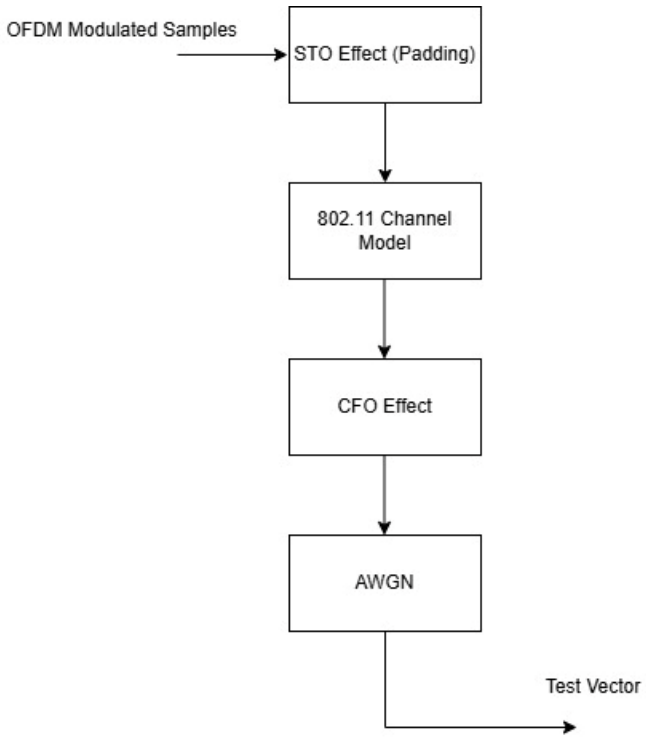


Figure 3-13 Test-vector generation diagram

The diagram depicts the sequential application of various channel effects used to create test vectors for Hardware Description Language (HDL) simulations and FPGA implementations of an OFDM system. Here's the step-by-step process:

1. Start with OFDM Modulated Samples: Initially, we have the OFDM modulated data samples.

2. Apply Symbol Timing Offset (STO): The samples are first subjected to the Symbol Timing Offset (STO) effect. This step includes adding padding to compensate for timing misalignments.

3. Pass through the 802.11 Channel Model: Next, the samples enter the 802.11 channel model, which simulates the multipath fading experienced in a wireless environment under different Root Mean Square (RMS) delay spreads.

4. Introduce Carrier Frequency Offset (CFO): After channel modelling, the samples undergo the Carrier Frequency Offset (CFO) effect. This step addresses the frequency mismatches caused by oscillator instabilities.

5. Add Additive White Gaussian Noise (AWGN): Finally, Additive White Gaussian Noise (AWGN) is added to simulate the ambient noise typically encountered in wireless communication.

The resulting output from this process is a test vector that effectively mimics real-world conditions, providing accurate data for HDL simulation and FPGA testing

3.2 TRANSMITTER PROCESSING CHAIN

This module integrates several key processes to prepare the data for transmission. Initially, the input data stream is mapped to complex symbols using a modulation scheme such as QAM (Quadrature Amplitude Modulation). Pilots and nulls are added and allocated to their appropriate subcarriers. These symbols are then fed into an IFFT (Inverse Fast Fourier Transform) block, which transforms the frequency domain symbols into the time domain. This conversion is essential for generating the OFDM signal, where each symbol occupies a distinct subcarrier. To mitigate inter-symbol interference, a cyclic prefix is added to the beginning of each OFDM symbol.
The transmitter also shapes the OFDM frame and inserts the preamble at the start of each frame.

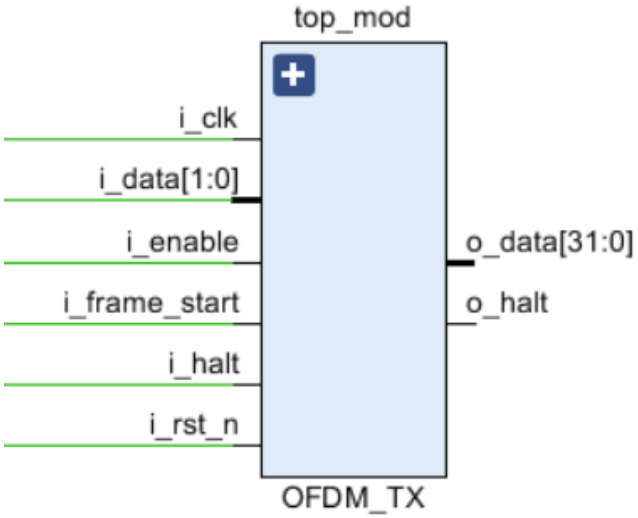


Figure 3-14 Transmitter top module Block Diagram

Signal	Width	Port Type	Description
i_clk	1 bit	Input	System clock signal
i_rst_n	1 bit	Input	Asynchronous reset signal active low
_i_data	2 bits	Input	Input data formatted into the word size required for transmission
i_enable	1 bit	Input	Enable signal for the top module
i_frame_start	1 bit	Input	Flag that indicates the start of a new frame
i_halt	1 bit	Input	Halt signal for the top module
o_halt	1 bit	Output	Halt signal for the top module indicates a pause in operation for the following steps
o_data	32 bits	Output	Output data of the transmitter that is a serial ofdm frame

3.2.1 Mapping

The input serial data stream is formatted into the word size required for transmission, (2 bits/word for QPSK) They are then used to decide the I and Q values of the desired complex symbol for the QPSK gray-coded mapping.

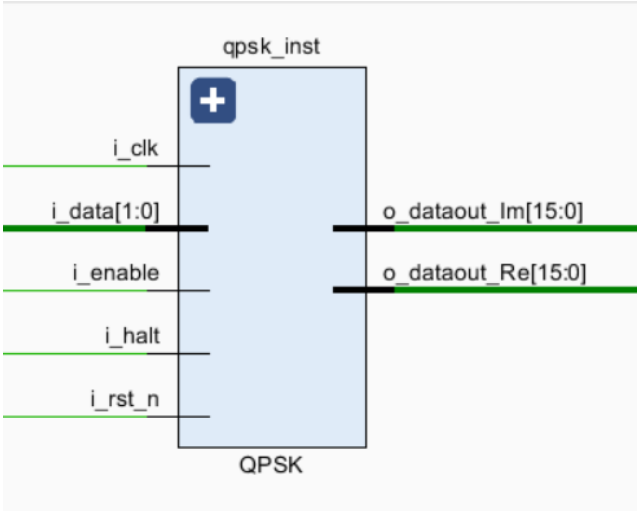


Figure 3-15 Mapping Module Block Diagram

3.2.2 Subcarrier Allocation

There are two purposes for this block the first is to allocate the data and pilot subcarriers, the second is to insert null subcarriers which carries “zero” information.
The pilot information is stored in dual port random-access memories (RAMs) that are then used to select the desired value to be allocated for the predetermined pilot subcarriers

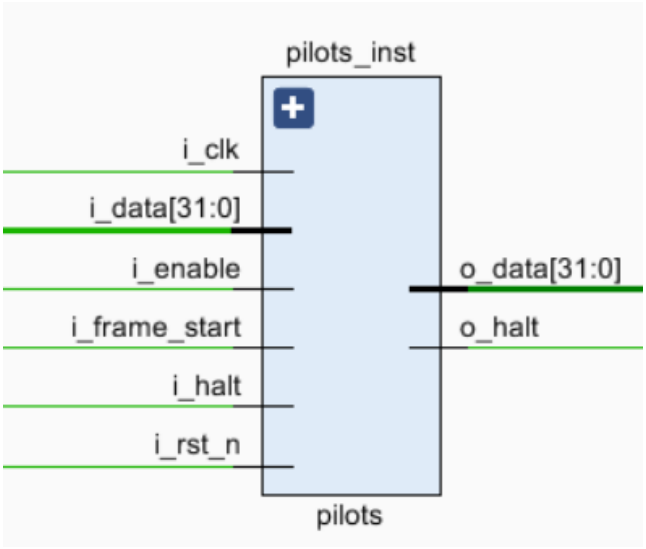


Figure 3-16 Subcarrier allocation Module Block Diagram

3.2.3 IFFT and Add Cyclic Prefix

The IFFT block is imported from Xilinx’s IP core library, named Fast Fourier Transform 9.1 ((xfft_v9.1). The Xilinx FFT v9.1 IP core uses the Radix-2 and Radix-4 algorithms for FFT computation, optimized for streaming

input data. These algorithms are well-suited for high-performance applications, leveraging the inherent parallelism and pipelining capabilities of FPGAs. The inputs and outputs are in fixed point representation.
By defining the “cp_len” in the configuration data, the CP is automatically added during IFFT output process.

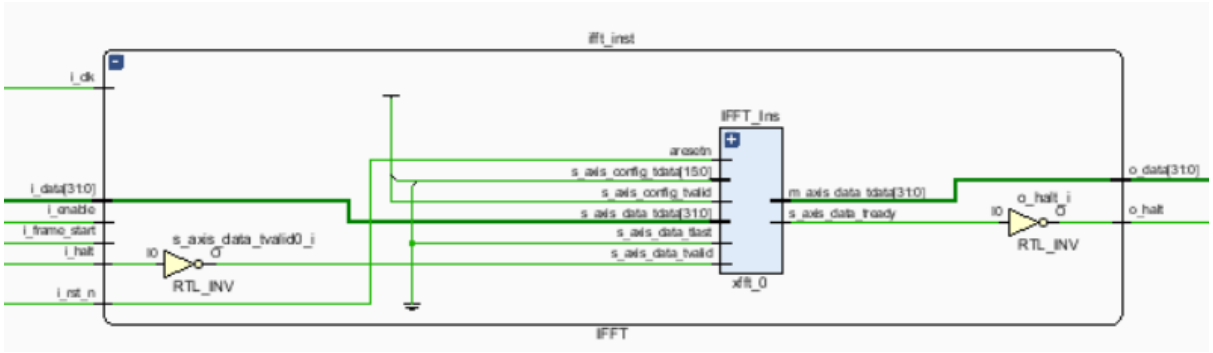


Figure 3-17 IFFT Module Block Diagram

3.2.4 Preamble Insertion

In our implementation both the short preamble sequence and the long preamble sequence are stored in a ROM to save computational resources as they are fixed for every OFDM frame.
The short preamble consists of 10 repeating sequences of 16 I/Q samples, or 160 samples in total.
The long preamble consists of two identical halves, each 80 samples long, resulting in 160 samples.

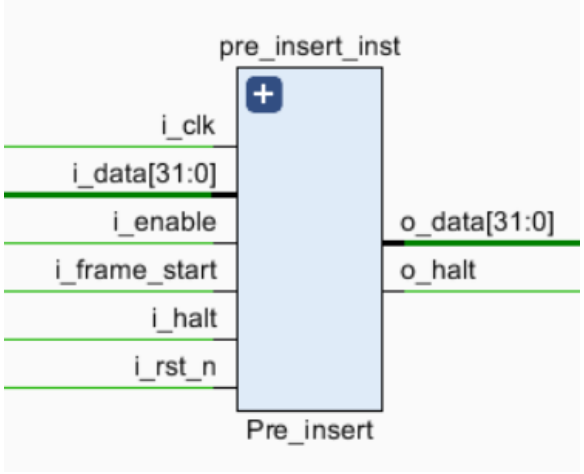


Figure 3-18 preamble Module Block Diagram

3.3 SYNCHRONIZATION

3.3.1 Packet detection module

A simplified block diagram of the synchronizer top module

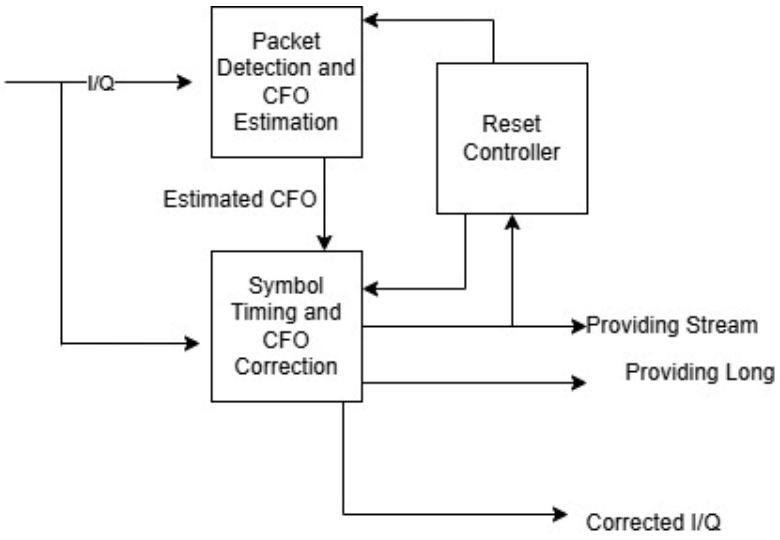


Figure 3-19 Simplified Block diagram of Synchronization

The synchronizer consists of 3 main blocks:

- Packet detection and CFO estimation
- Symbol Timing and CFO correction
- Reset Controller

The flow start first by inserting the I/Q samples to the packet detector to detect if a packet exists and estimates the CFO.
Then the packet existence information and the estimated CFO are passed to the symbol timing and CFO correction.
The Reset module is responsible for flushing the Synchronization blocks every 4 OFDM frames in order to receive new packets, also the main reset signal is passed through it.

3.3.2 Implementation of Packet detection and CFO estimation

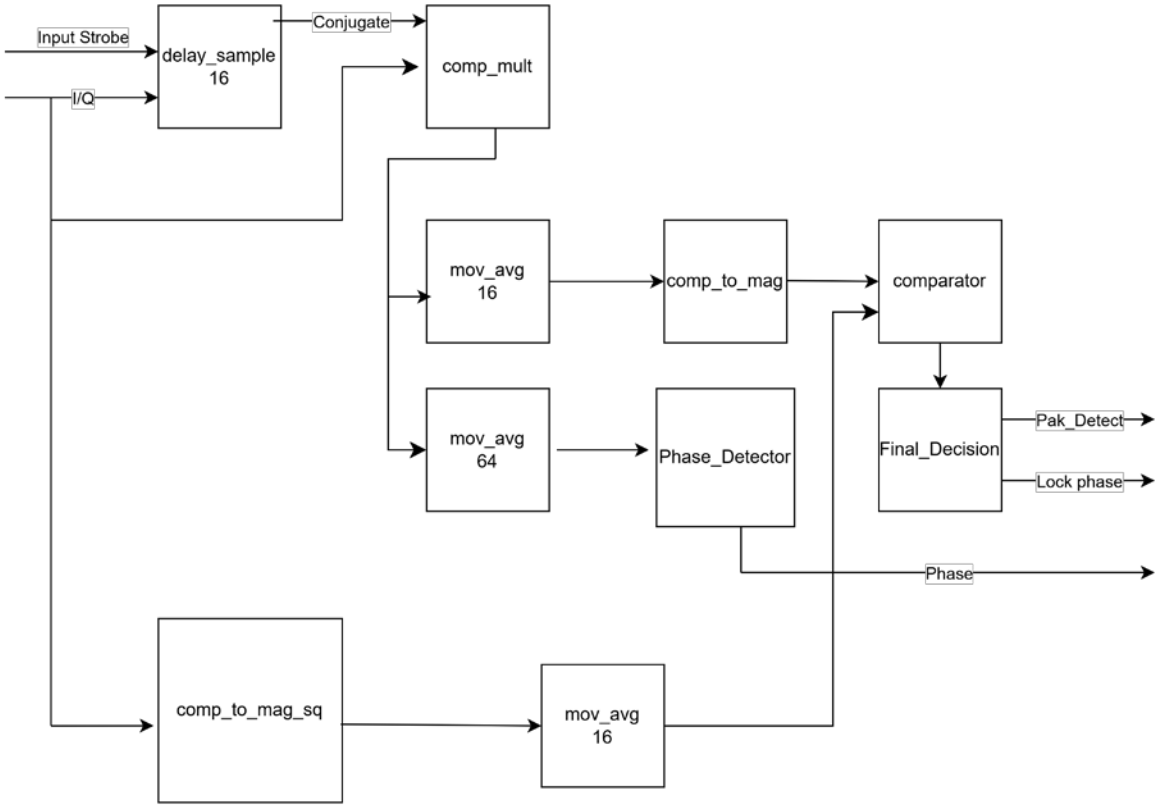


Figure 3-20 Block diagram of Packet detection and CFO estimation module

The module has 2 main paths, a path that correlates 16 point window with a delayed window, and a path that calculates the energy in the current 16 point window.

The first upper path is represented in 16 point delay_sample, complex_multiplier and moving_average of 16 point and a complex_to_magnitude, this finds the correlation between current 16 points window and a delayed version of 16 points.

There is also a derived path that takes average of 64 points in order to find the CFO.

The second path is represented in complex_to_magnitude_squared passed over a moving_average of 16 points, this finds the power of the current 16 points window.

The comparator checks if the value the correlation is greater than 0.75 the value of the energy of the close window.

The final decision block counts 20 points and checks if there are 20 consecutive points that cross the threshold, if this is achieved, the final decision module raises the packet_detected flag and phase_lock to tell the symbol timing and CFO correction module to sample the estimated phase and find the start of OFDM frame.

3.3.2.1 Packet detection top module

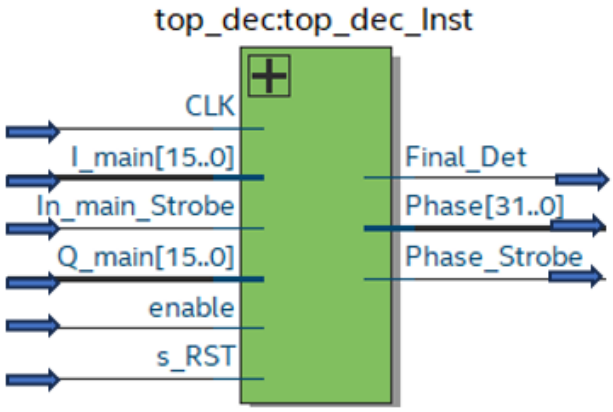


Figure 3-21 Packet detection top module

Table 3-1 Packet detection ports description

Signal	Width	Port Type	Description
CLK	1 bit	Input	System clock signal
S_RST	1 bit	Input	Module reset signal
enable	1 bit	Input	Receivers enable signal
In_main_Strobe	1 bit	Input	Indicates the arrival of new valid data
I_main	16 bits	Input	Real part of input to the receiver
Q_main	16 bits	Input	Imaginary part of input to the receiver
Final_Det	1 bit	Output	Output signal indicates packet detection
Phase	32 bits	Output	CFO phase estimation in radians
Phase_Strobe	1 bit	Output	Indicates the validity of the estimated phase

3.3.2.2 Complex to magnitude estimator using alpha max plus beta min algorithm top module

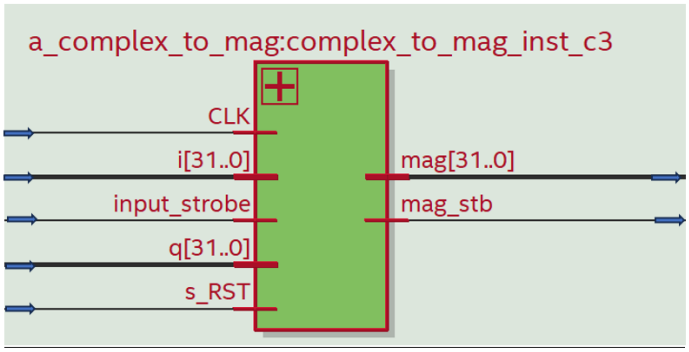


Figure 3-22 complex to mag top module

Table 3-2 Complex to mag ports description

Signal	Width	Port Type	Description
CLK	1 bit	Input	System clock signal
_i	32 bits	Input	Real part of input
input_strobe	1 bit	Input	Indicates the arrival of new valid data
Q_main	16 bits	Input	Imaginary part of input to the receiver
S_RST	1 bit	Input	Module reset signal
mag_stb	1 bit	Output	Indicates the output is valid
mag	32 bits	Output	The estimated magnitude

3.3.2.3 Phase estimator of complex numbers using Vectoring mode top module

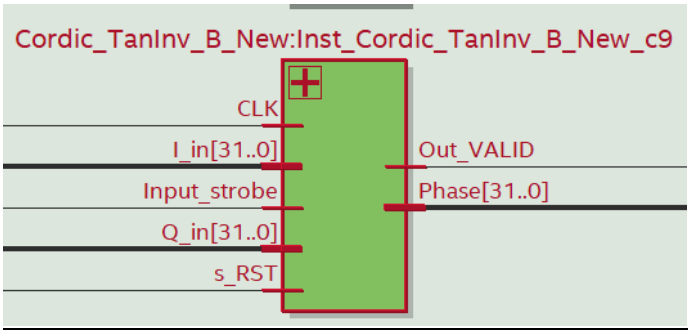


Figure 3-23 Angle Estimator top module

Table 3-3 Angle estimator ports description

Signal	Width	Port Type	Description
CLK	1 bit	Input	System clock signal
_I_in	32 bits	Input	Real part of input
Input_strobe	1 bit	Input	Indicates the arrival of new valid data
Q_in	32 bits	Input	Imaginary part of input to the receiver
S_RST	1 bit	Input	Module reset signal
Out_VALID	1 bit	Output	Indicates the output is valid
Phase	32 bits	Output	The estimated phase

3.3.3 symbol timing offset and CFO module block diagram

This part is adapted from [26]

The algorithm used in the symbol timing is the cross-correlation technique with a maximum search.
A quantized version of the samples is stored in the receiver, the quantization reduces the stored data of each sample from 16 bits to only a sign bit, this significantly reduces the hardware size represented by adders and multipliers required for the cross correlation as the cross-correlation algorithm has high complexity.

CFO correction is made using CORDIC in rotational mode, the phase is sampled after being estimated from the packet detection module.

Coarse CFO is made only, no fine CFO is made due to the residual error caused by the nature of CORDIC, so this effect is compensated by pilot correction in the rest of the receiver chain.

Figure shows the block diagram of the Long_Sync module responsible for estimating the start of OFDM frame and providing the samples to rest of receiver chain with CP removed.

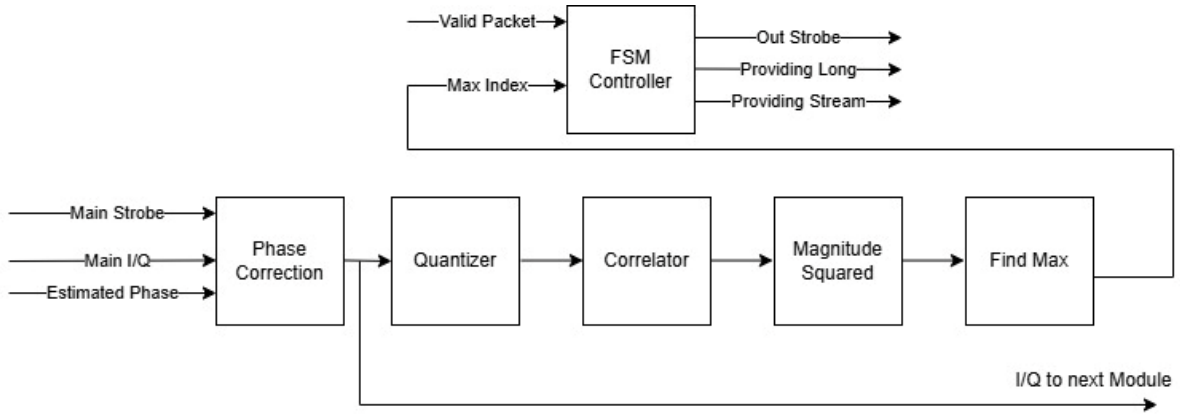


Figure 3-24 Simplified STO and CFO correction Block Diagram

FSM controls the module blocks, but the control signals of FSM are removed from this block diagram for simplicity.

The flow is as follows:

The packet detection module detects the packet existence and provides the info of the packet existence and the estimated CFO through the Valid Packet and Estimated Phase.

The phase corrector (a CORDIC in rotation mode) holds the CFO estimated angle, and adds this angle in an accumulative way on every new I/Q sample in order to correct the phase accumulation caused by CFO, then the sample is corrected.

Quantizer quantizes the samples into a sign bit to make the correlator simple.

The magnitude squared module squares the output of the correlator, this step is made to increase the difference between the maximum and the second maximum, so a more complicated decisions can be made in Find max if needed.

Find max module works on finding the maximum value of the correlation and storing it's corresponding index.

Finally the FSM waits a certain number of samples depending on the found max index.

FSM Providing Long signal indicates that the supplied data is the second Long training symbol which will be used in the channels estimation.
FSM waits the CP length and provides the OFDM frame itself with the Providing Stream signal raised.

After 4 supplied OFDM frames, the reset module resets the whole module.
FSM explanation
The FSM uses a what is called a "General purpose" counter that hold a certain value at each state of the FSM.

1. "IDLE": which means that the FSM is waiting for a valid packet detection.

2. "SAMPLE_PHASE": an intermediate state that tells the phase corrector to sample the angle provided by the packet detection module.

3. "RESTING": The FSM waits for 122 samples, it is estimated that the beginning of CP of first Long training symbol is here.

4. “CORRELATING”: The module is in correlating state which lasts for about 68 samples, and the “General Purpose Counter” is loaded with a certain value according to the found max index.

5. “WAIT_FOR_SECOND_TRAIN”: The FSM waits (Max index – 36), it is estimated that the start of second training symbol is here.

6. “PROVIDING_LONG”: the module is providing the long training sequence now and the FSM counts 64 points.

7. “WAITING_CP”: The FSM waits for 16 points of the CP.

8. “PROVIDING_STREAM”: The module provides the OFDM frame and counts 64 points, then it returns to the WAITING_CP state.

The FSM alternates between these two last states till the reset module resets it after the packet finished, returning it to the IDLE state to start the flow again with a new packet.

3.3.3.1 STO and CFO correction top module

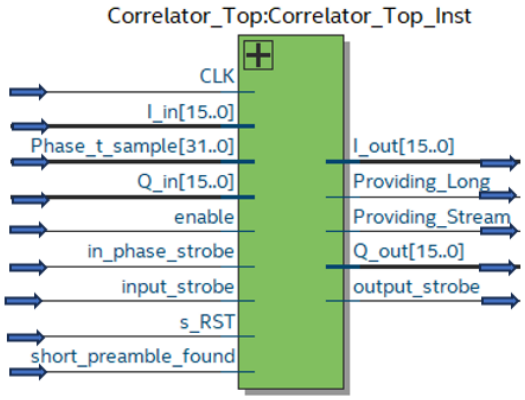


Figure 3-25 STO and CFO correction top module

Table 3-4 STO and CFO estimation top module ports description

Signal	Width	Port Type	Description
CLK	1 bit	Input	System clock signal
I_in	1 bit	Input	Real part of input to the receiver
Phase_t_sample	32 bit	Input	The estimated CFO phase
Q_in	16 bits	Input	Imaginary part of input to the receiver
enable	1 bits	Input	Enable signal to module
in_phase_strobe	1 bit	Input	Indicated the arrival of valid phase
input_strobe	1 bits	Input	Indicates the arrival of valid I/Q
s_RST	1 bit	Input	Reset the module
Short_preamble_found	1 bit	Input	Indicates the existence of a packet
I_out	16 bits	Output	Real CFO corrected sample
Providing_Long	1 bit	Output	Indicated the supply of LTS field
Providing_Stream	1 bit	Output	Indicates the supply of OFDM frame samples
Q_out	16 bits	Output	Imaginary CFO corrected sample
output_strobe	1 bit	Output	Indicated that output is valid

3.3.3.2 Correlator calculator top module

The correlator implementation is a weight stationary systolic architecture.

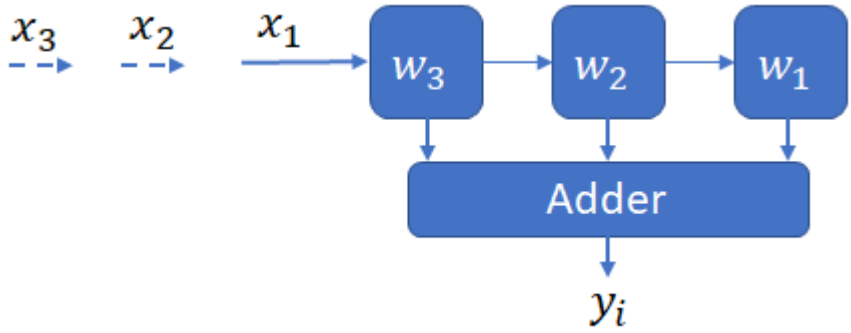


Figure 3-26 Architecture used in correlator

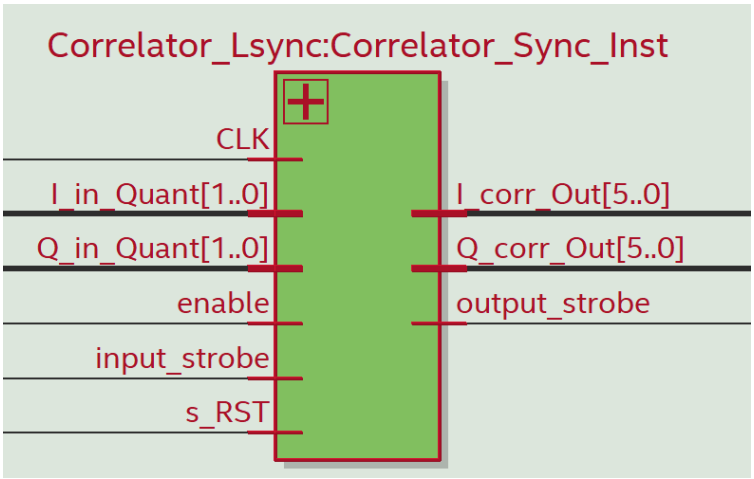


Figure 3-27 Correlator Calculator top module

Table 3-5 Correlator calculator ports description

Signal	Width	Port Type	Description
CLK	1 bit	Input	System clock signal
S_RST	1 bit	Input	Module reset signal
enable	1 bit	Input	Receivers enable signal
input_strobe	1 bit	Input	Indicates the arrival of new valid data
I_in_Quant	2 bits	Input	Real part quantized input
Q_in_Quant	2 bits	Input	Imaginary part of quantized input
I_corr_Out	6 bits	Output	Real output part of correlator
Q_corr_Out	6 bits	Output	Imaginary output part of correlator
Output_strobe	1 bit	Output	Indicates the validity of the output

3.3.3.3 Phase corrector, a CORDIC in Rotational mode and phase accumulator top module

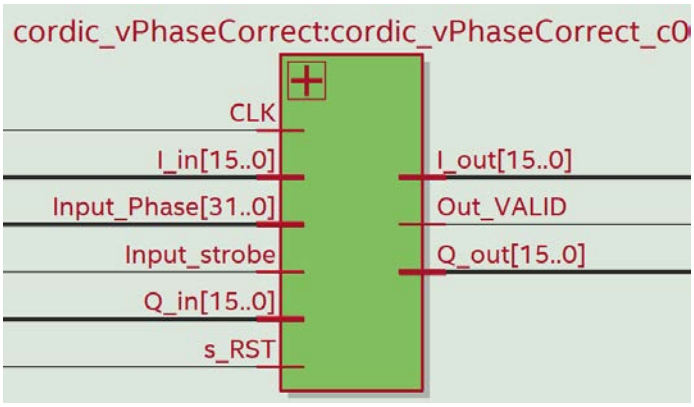


Figure 3-28 Phase correction top module

Table 3-6 Phase correction module ports description

Signal	Width	Port Type	Description
CLK	1 bit	Input	System clock signal
_I_in	16 bits	Input	Real part of input
Input_strobe	1 bit	Input	Indicates the arrival of new valid data
Q_in	16 bits	Input	Imaginary part of input to the receiver
Input_Phase	32 bits	Input	The phase that needs to be rotated
s_RST	1 bit	Input	Module reset signal
Out_VALID	1 bit	Output	Indicates the output is valid
I_out	16 bits	Output	The rotated I
Q_out	16 bits	Output	The rotated Q

3.3.4 Synchronization reset module

The module is responsible for resetting and flushing the synchronization modules in order to be ready for new packet arrival.

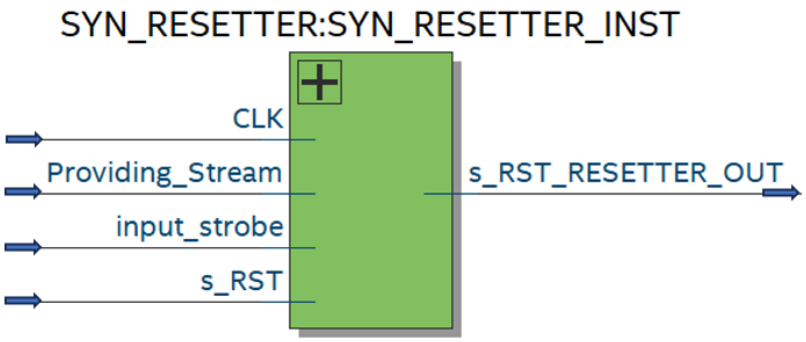


Figure 3-29 Reset controller top module

Table 3-7 Reset controller ports description

Signal	Width	Port Type	Description
CLK	1 bit	Input	System clock signal
Providing_Stream	1 bit	Input	Indicates the supply of OFDM frame samples
input_strobe	1 bits	Input	Indicates the arrival of valid I/Q
s_RST	1 bit	Input	Reset the module
s_RST_RESETTER_OUT	1 bit	Output	Flushes and resets the packet detection and symbol detection modules

3.4 RECIEVER PROCESSING CHAIN

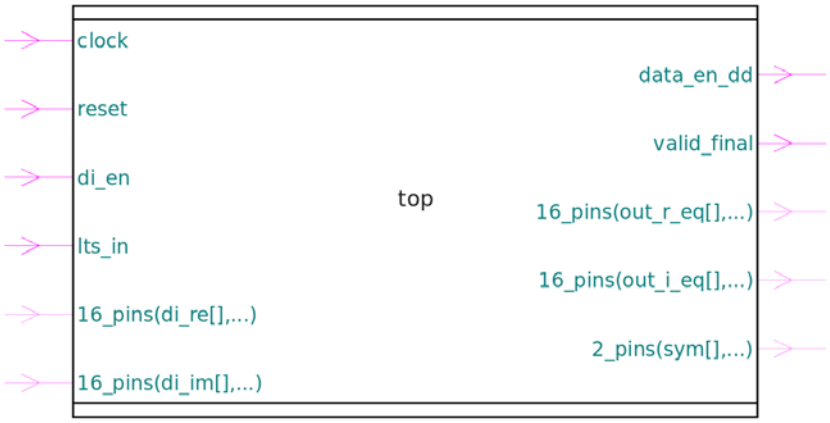


Figure 3-30 Receiver top module

Table 3-8 Receiver ports description

Signal	Width	Port Type	Description
clock	1 bit	Input	System clock signal
reset	1 bit	Input	Module reset signal
di_en	1 bit	Input	Receivers enable signal
LTS_in	1 bit	Input	Indicated that the frame supplied is a preamble
di_re	16 bits	Input	Real part of input to the receiver
di_im	16 bits	Input	Imaginary part of input to the receiver
data_en_dd	1 bit	Output	Output enable signal for SQNR verification
out_r_eq	16 bit	Output	Real part of output before modulator for SQNR verification
out_i_eq	16 bits	Output	Imaginary part of output before modulator for SQNR verification
valid_final	1 bit	Output	Valid modulated output signal for final symbols
sym	2 bits	Output	Final demodulated symbols

3.4.1 Fast Fourier Transform

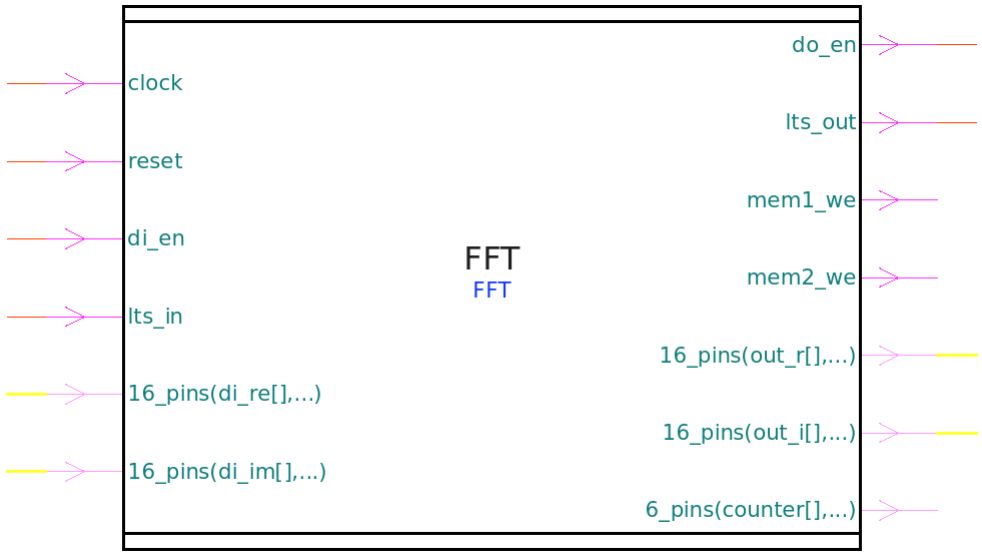


Figure 3-31 FFT top module

Table 3-9 FFT ports description

Signal	Width	Port Type	Description
clock	1 bit	Input	System clock signal
reset	1 bit	Input	Module reset signal
di_en	1 bit	Input	Receivers enable signal
LTS_in	1 bit	Input	Indicated that the frame supplied is a preamble
di_re	16 bits	Input	Real part of input to the receiver
di_im	16 bits	Input	Imaginary part of input to the receiver
do_en	1 bit	Output	Output enable signal for SQNR verification
out_r_eq	16 bits	Output	Real part of transformed output
out_i_eq	16 bits	Output	Imaginary part of transformed output
LTS_out	1 bit	Output	Indicates that output symbol is preamble to other modules for equalization
counter	6 bits	Output	Address counter indicating the index of the transformed output [0 to 63]
mem1_we	1 bit	Output	Memory 1 write enable for verification
mem2_we	1 bit	Output	Memory 2 write enable for verification

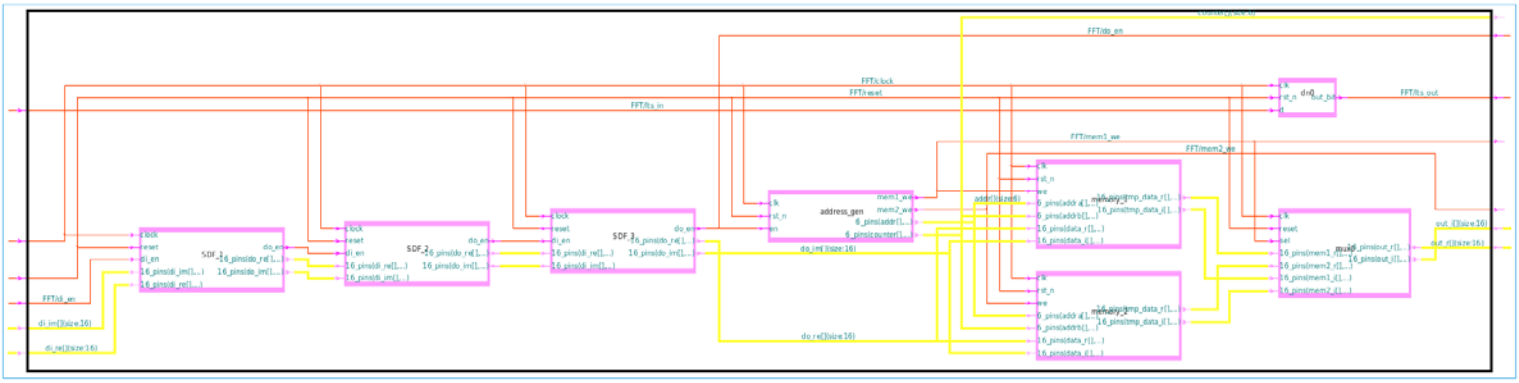


Figure 3-32 Various components inside the FFT

Figure 3-3 shows the data flow inside the FFT showing its various components, each component contributed to transforming the data into frequency domain. The FFT algorithm used is the Radix-22 FFT which uses to CFA which greatly improves the area and the speed of the transform.

The components of the FFT are:

- 3 SDF units
- Address generator
- 2 memory components each of size 64 x 16 bits
- Multiplexer

3.4.1.1 SDF units

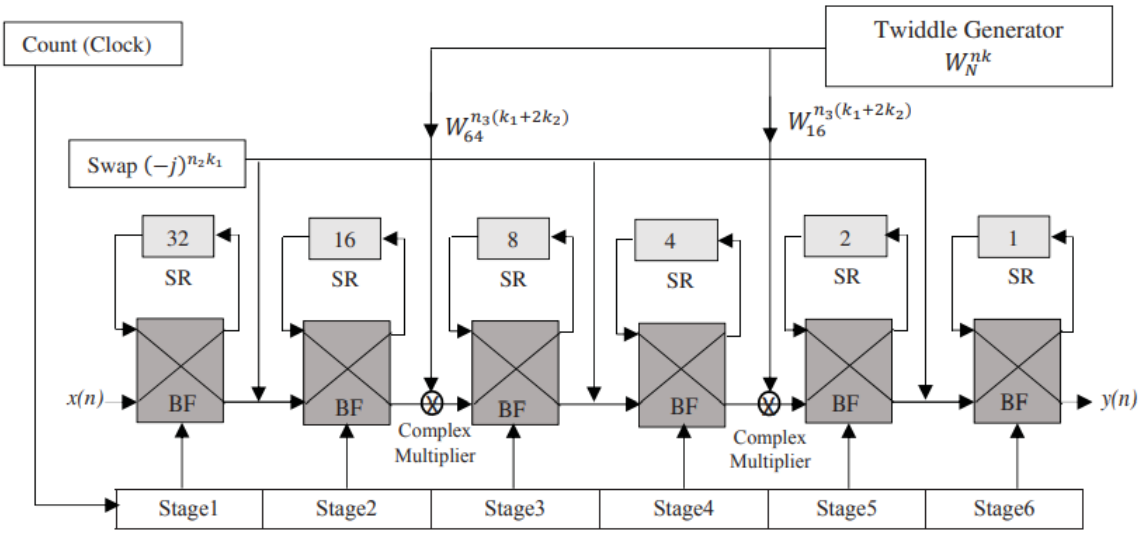


Figure 3-33 Full SDF units of 64-point FFT

As stated in chapter 2.2.4.2.2.1 and [27] the SDF here is the core engine for the FFT, it takes the I and Q inputs from the previous SDF units or the synchronizer with it enable signal and outputs a valid signal once the transform of the aligned symbols is done, the diagram below illustrated the butterfly and delay units, as well as the trivial multipliers present between the SDF unit

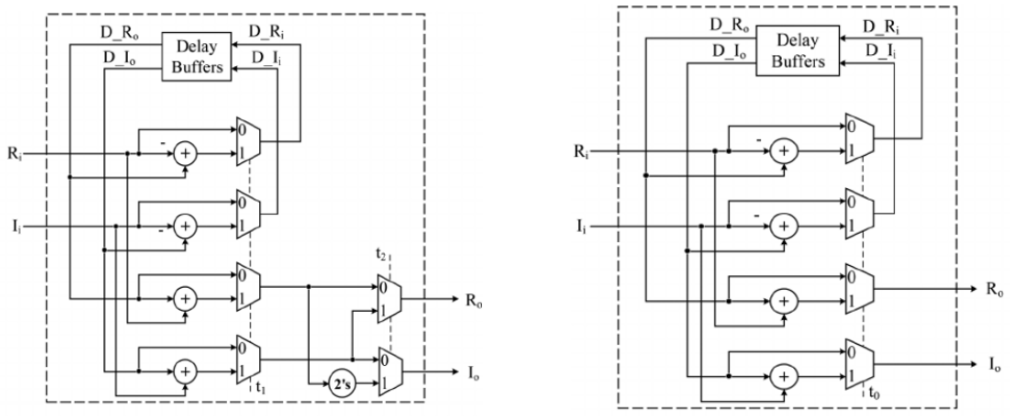


Figure 3-34 Detailed representation of SDF stages and butterfly units BF1 (Left) and BF2 (Right)

As we can see here [28] BF1 is the butterfly with the trivial multipliers, indicated by the 2's complement for the multiplication of a negative number and the select of the mux determines whether the imaginary and real parts are swapped or not for the {j, -j} multiplication

After each SDF unit we need a complex multiplier for the twiddle factors, the complex multiplier can be done using 4 full multipliers a full adder and a full subtractor as shown in the diagram below:

$$(a + jb)(c + jb) = (ac - bd) + j(ad + bc)$$

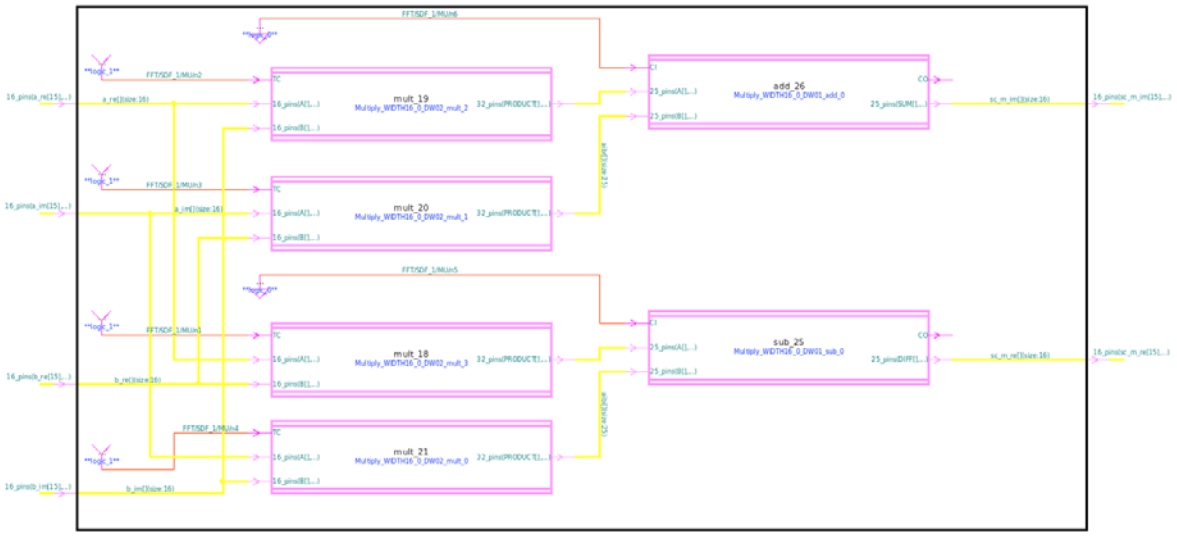


Figure 3-35 Complex Multiplier block diagram

A complex multiplier is present after each SDF unit (2 butterflies), except for the last SDF stage as we multiply by unit twiddle factors, so no need for complex multiplication. We have 3 SDF units connected in series to each other, each help in transforming the inputs, the delay buffers align the inputs between each stage. The output of the FFT is bit reversed so a reordering unit and an address generator are needed to supply the indices of the outputs and a reordering unit to order the input correctly to be passed on to the next stages

3.4.1.2 Address and Reorder units

As mentioned in the previous chapters, the output from the FFT is bit reversed, so they will need ordering, the address generator is just a bit reversed counter that increments by 1 every clock cycles so it supplies the correct index to the memory to order and place the output in the correct place, once the memory is filled, it outputs the contents of the it to the next stages

3.4.1.2.1 Memory contention problem

The reordering memory can only output the symbols once all are reordered and supplied from the FFT, so once the memory output happens, more FFT symbols are output which need reordering as well. To address this problem, we will insert another memory as shown in figure 3.3, as one memory is supplying the symbols, the other memory will receive the symbols and sort them while the other finishes, once it finishes sorting, it starts suppling the symbols, and the memories switch roles, so that one is always receiving and sorting while the other is supplying the sorted symbols.

The memories need control blocks to supply the read and write enable to the memories and that they are always opposite to each other, these signals are mem1_we and mem2_we, they are supplied to the memories to handle which gets written and which outputs the data, while the address and the supplied symbols are not on the memory ports, the behaviour of the memory is determined by the signals mentioned.

A multiplexer is needed at the end of the FFT chain to select from which memory is the output supplied, so at the select of the mux is one of the memories write enable signals and the MUX output is supplied to the next stages.

3.4.2 Pilot Null Detector

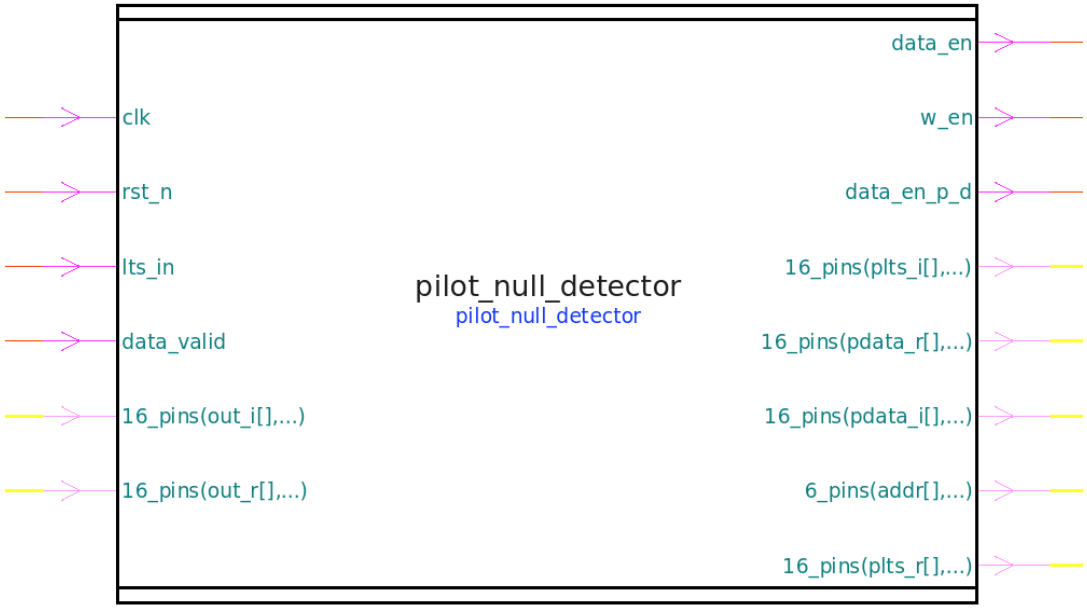


Figure 3-36 Pilot Null detector port description

Table 3-10 Pilot Null detector port description

Signal	Width	Port Type	Description
clock	1 bit	Input	System clock signal
reset	1 bit	Input	Module reset signal
data_valid	1 bit	Input	Module enable/valid signal
LTS_in	1 bit	Input	Indicated that the frame supplied is a preamble
out_r	16 bits	Input	Real part of input to the receiver
out_i	16 bits	Input	Imaginary part of input to the receiver
data_en	1 bit	Output	Output enable signal for channel equalizer to equalize the DATA symbols
w_en	1 bit	Output	Write enable for the memory to write the LTS symbols and store for equalization
data_en_p_d	1 bit	Output	Output enable signal for channel equalizer to equalize the PILOT symbols
addr	6 bits	Output	Deallocated address for the equalizers and the reference symbols memory
plts_r	16 bits	Input	Real part of the LTS pilot symbols
plts_i	16 bits	Input	Imaginary part of the LTS pilot symbols
pdata_r	16 bits	Input	Real part of the DATA pilot symbols
pdata_i	16 bits	Input	Imaginary part of the DATA pilot symbols

The pilot null detector has 2 functions:

- Filtering out the null and pilot subcarriers from the symbol, turning the data symbols from 64 to 48 symbols
- Storing the pilot subcarriers of the LTS and output them along with the data pilot subcarriers for equivalization and averaging them out then equalizing the symbols of the pilot with the averaged-out pilots for phase offset

The module has 2 types of enable signals, and they are used to indirectly filter out the symbols of both the data and pilots without needing an additional memory, the signals are data_en/data_en_p and w_en/w_en_p, the data enable signals are passed on to the equalizers, while the module do not output the filtered out symbols the equalizer has their input I Q symbols directly from the FFT and the data enable signals enable the equalizers to sample the input on the ports and perform equalizers based on the index supplied from the module {addr por. The write enable signals are passed on to the LTS memory and the small 4x16 memory inside the pilot null detector to store the LTS symbols and the LTS pilots.

3.4.3 Phae offset and channel equalizer

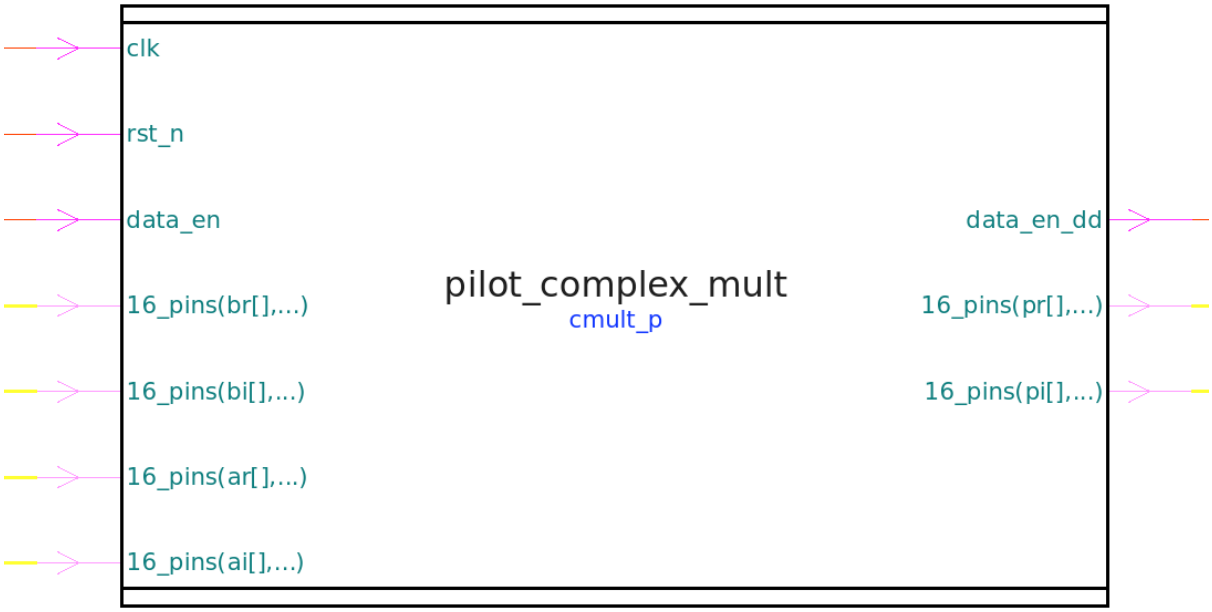


Figure 3-37 Pilot Equalizer block diagram

Table 3-11 Pilot equalizer port description

Signal	Width	Port Type	Description
clock	1 bit	Input	System clock signal
reset	1 bit	Input	Module reset signal
data_en	1 bit	Input	Module enable/valid signal
a_r	16 bits	Input	Real part of input to the receiver
a_i	16 bits	Input	Imaginary part of input to the receiver
b_r	16 bits	Input	Real part of the LTS pilot symbols
b_i	16 bits	Input	Imaginary part of the LTS pilot symbols
pdata_r	16 bits	Output	Real part of the equalized DATA pilot symbols
pdata_i	16 bits	Output	Imaginary part of the equalized DATA pilot symbols

For our equalizer we will apply the least squares equalization method we described in section 2.2.4.2.3.1, we need to divide the received signal by the reference signal, this method can be used to equalize both the phase and magnitude of the signal. In our system we modulated the symbols using 4 QAM, so we will not need to equalize the magnitude of the symbols, but instead we only need to equalize the phase of the symbols. This helps with both phases offset as it causes rotation and channel estimation as we only equalize the phase, the symbol returns to its quadrant and can be demodulated, the following equation states the equalizer used:

$$\hat{\mathbf{S}} = \hat{\mathbf{H}}^{-1}\mathbf{R} = \frac{\mathbf{L_X}}{\mathbf{L_Y}}\mathbf{R} = \frac{\mathbf{L_X}}{|\mathbf{L_Y}|^2}\mathbf{L_Y}^* \cdot \mathbf{R}$$

And if we do not want to equalize the magnitude like in our case, the equation converges to:

$$\hat{\mathbf{S}} = \mathbf{L_X} \cdot \mathbf{L_Y}^* \cdot \mathbf{R}$$

L_x is the stored preamble in the memory
L_y is the received preamble symbol
R is the received data or pilot symbols

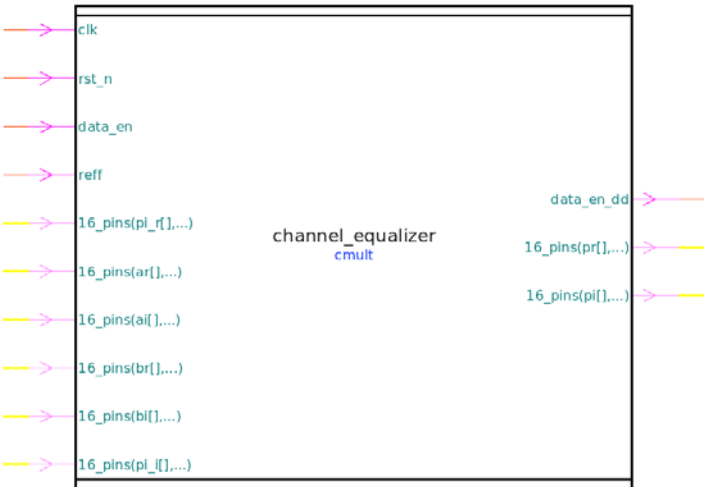


Figure 3-38 Channel Equalizer block diagram

Table 3-12 Channel Equalizer port description

Signal	Width	Port Type	Description
clock	1 bit	Input	System clock signal
reset	1 bit	Input	Module reset signal
data_en	1 bit	Input	Module enable/valid signal
reff	1 bit	Input	Reference symbol from the ROM of the LTS
pi_r	16 bits	Input	Real part of equalized pilots to the equalizer
pi_i	16 bits	Input	Imaginary part of equalized pilots to the equalizer
a_r	16 bits	Input	Real part of input to the receiver
a_i	16 bits	Input	Imaginary part of input to the receiver
b_r	16 bits	Input	Real part of the LTS pilot symbols
b_i	16 bits	Input	Imaginary part of the LTS pilot symbols
data_en_dd	1 bit	Output	Valid Data signal
pr	16 bits	Output	Real part of the equalized DATA pilot symbols
pi	16 bits	Output	Imaginary part of the equalized DATA pilot symbols

To estimate and correct the phase offset we need to perform several steps:

- Extract the pilots in the preamble
- Extract the pilots in the data symbol
- Apply channel equalization to get the deviation caused by channel, from the stored preambles in the ROM
- Take the complex conjugate of the channel equalized pilots in the data symbols and multiply them by their predetermined values at the transmitter (This is known to the receiver, and stored in the memory as {-1, 1, -1, 1})

- Take the complex average of the 4 pilot samples and equalize all the other data samples in this symbol by this pilot average, the pilot average represents the phase offset we are trying to estimate
- The equalization done is a phase equalization, so it follows the same pattern as the channel equalizer, in fact the same channel equalizer is instantiated twice, one for the channel equalization and the other for the phase offset equalization
- This equalization is done per OFDM symbol in the whole frame, so each OFDM symbol is phase equalized by their pilot symbols as the phase offset changes from one OFDM symbol to another.

A clearer representation using equations can be seen below:

$$\theta_n = \angle\left(\sum_{i\in\{-21,-7,7,21\}} \overline{X^{(n)}[i]} \times P^{(n)}[i] \times H[i]\right)$$

H is the channel equalized pilots
P is the predetermined sequence from the transmitter {-1, 1, -1, 1}
X is the pilots in the OFDM symbol

The pilot average module is a cumulative adder, that sums the pilots once every data_en_p is asserted, and once every 4 additions the result of the accumulative adder is shifted by 2 to take the average of the pilot samples. The averaged pilot is then sent to the equalizer to apply the equalization needed for the data samples, so to wrap up, the data samples undergo 2 types of equalizations:

- Channel equalization, where we take the preamble and stored samples in the memory and correct the effect of the channel this happens on both the DATA and PILOT samples in the OFDM frame
- Do the same procedure on the 4 pilot samples, using the predetermined values as reference signals, multiply them by the pilot samples from the received frame, average, take complex conjugate, and multiply the whole OFDM frame by it to correct the phase

3.4.4 Demodulator

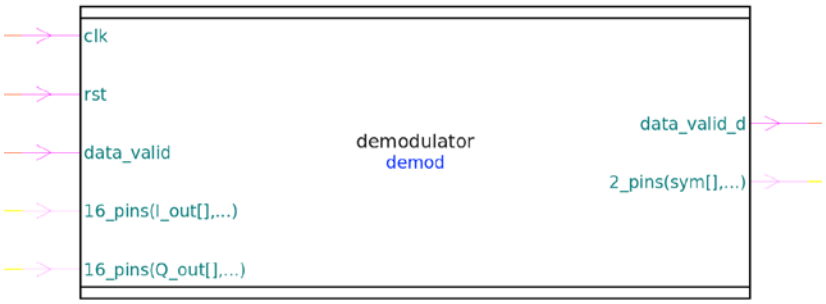


Figure 3-39 Demodulator block diagram

Table 3-13 Demodulator ports description

Signal	Width	Port Type	Description
clock	1 bit	Input	System clock signal
reset	1 bit	Input	Module reset signal
data_valid	1 bit	Input	Module enable/valid signal
I_out	16 bits	Input	Real part of input to the demodulator
Q_out	16 bits	Input	Imaginary part of input to the demodulator
data_valid_d	1 bit	Output	Valid output at the output ports signal
sym	2 bits	Otput	Demodulated symbol

The demodulator used here is a 4 QAM demodulator, the signals are grey encoded in the demodulator (as well as the modulator in the receiver) the decoding method used in a typical demodulator is a maximum likelihood demodulator, where we take the square difference in constellation between the symbol and the neighbouring symbols but that was not used here as we have a 4 QAM signal, the demodulator only checks in which quarter the signal lies in and a corresponding symbol bit is given to the signal and output to retrieve the original symbols.

4 SYSTEM MODELLING RESULTS

4.1 THEORETICAL BER CURVES

The purpose of the system is to produce Bit Error Rate curves, and compare it to the theoretical BER curves of the modulation schemes tested to prove that the overall error rates decrease when we use OFDM based transmission

4.1.1 AWGN channels

Table 4-1: Equations used to generate the BER curves for the AWGN channel[29]

Modulation	Symbol error rate (P_s)
MPAM	$2\left(1-\frac{1}{M}\right)Q\left(\sqrt{\frac{6}{M^2-1}\gamma_s}\right)$
BPSK	$Q\left(\sqrt{2\gamma_b}\right)$
QPSK	$2Q\left(\sqrt{2\gamma_b}\right)-Q^2\left(\sqrt{2\gamma_b}\right)$
MPSK $(M>4)$	$2Q\left[sin\left(\frac{\pi}{M}\right)\sqrt{2\gamma_s}\right]$
MQAM	$1-\left[1-2\left(1-\frac{1}{\sqrt{M}}\right)Q\left(\sqrt{\frac{3\gamma_s}{(M-1)}}\right)\right]^2$

4.1.2 Rayleigh flat-fading channel

Table 4-2: Equations used to generate the BER curves for the Rayleigh channel [13]

$$\mathcal{M}_{\gamma_s}\left(-\frac{g}{sin^2\phi}\right)=\left(1+\frac{g\bar{\gamma}_s}{sin^2\phi}\right)^{-1}$$

Modulation	Parameter g	Avg Prob. of Symbol error $\bar{P}_s$
BPSK	-	$0.5\left(1-\sqrt{\frac{\bar{\gamma}_s}{1+\bar{\gamma}_s}}\right)$
MPSK	$sin^2\left(\frac{g}{M}\right)$	$\frac{1}{\pi}\int_0^{\frac{(M-1)\pi}{M}}\mathcal{M}_{\gamma_s}\left(-\frac{g}{sin^2\phi}\right)d\phi$
MQAM	$\frac{1.5}{(M-1)}$	$\frac{4}{\pi}\left(1-\frac{1}{\sqrt{M}}\right)\int_0^{\pi/2}\mathcal{M}_{\gamma_s}\left(-\frac{g}{sin^2\phi}\right)d\phi-\frac{4}{\pi}\left(1-\frac{1}{\sqrt{M}}\right)^2\int_0^{\pi/4}\mathcal{M}_{\gamma_s}\left(-\frac{g}{sin^2\phi}\right)d\phi$
MPAM	$\frac{3}{(M^2-1)}$	$\frac{2(M-1)}{M\pi}\int_0^{\pi/2}\mathcal{M}_{\gamma_s}\left(-\frac{g}{sin^2\phi}\right)d\phi$

4.2 SIMULATED BER CURVES

To plot the simulated BER and compare it with the theoretical BER we will need to define mainly 3 aspects:

1. Symbol to Noise Ratio
2. Modulation and Demodulation Scheme
3. Channel model

The channel model used is the channel simulator described in Chapter 3, we will use it here to apply noise or fading to the signal with the selected SNR for AGWN channels for example or channel parameters describing LOS and NLOS components for Rayleigh or Ricin channels

4.2.1 Symbol to Noise Ratio

First, we will define an array of the different Signal to Noise Ratios that we want to test the system on, to map the respective Signal to noise to Symbol to noise we need to consider the modulation order used for the test.

Simply we will convert the number of bits used in the modulation order to the Signal to noise ratio we defined to get the Symbol to noise ratio, so for example for a Binary modulation order we add 10*log (1) to the SNR, for a Quad order (we use 2 bits to represent all symbols) we add 10*log (2) and so on [13]

4.2.2 MODEM Results

The results are calculated by dividing the total number of symbols with the correctly received symbols and plotting the SER for all modulation schemes under different channel conditions

4.2.2.1 AWGN channel

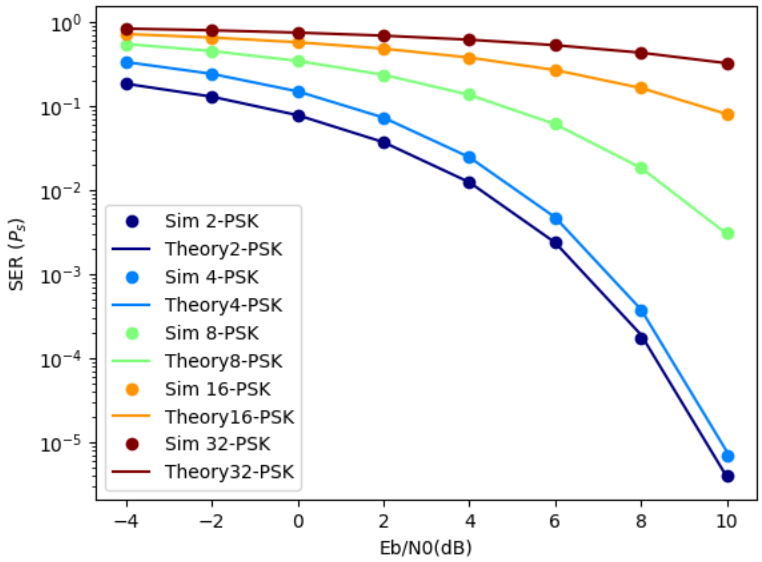


Figure 4-1: M-PSK AWGN

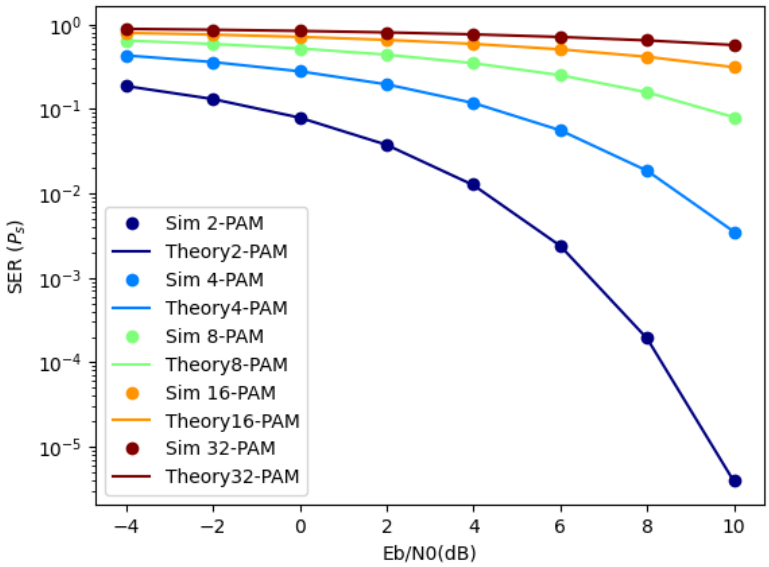


Figure 4-2: M-PAM AWGN

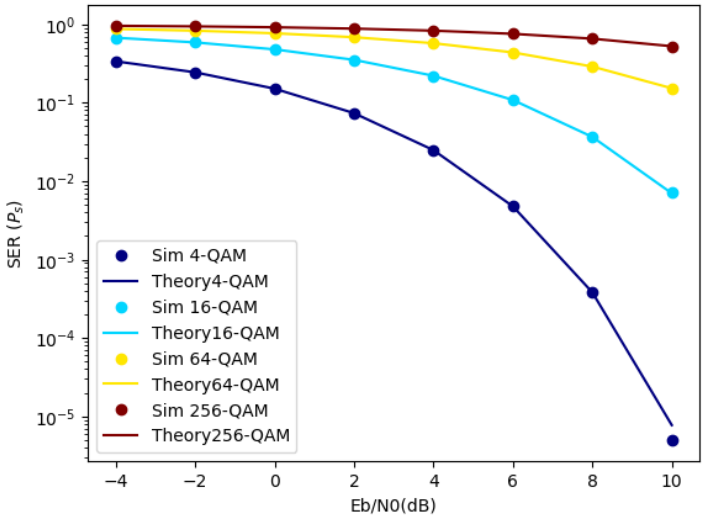


Figure 4-3: M-QAM AWGN

As we can observe here the simulated and theoretical curves align proving that the simulation matches the theoretical for AWGN channel in QAM, PAM and PSK modulation schemes

4.2.2.2 Rayliegh channel

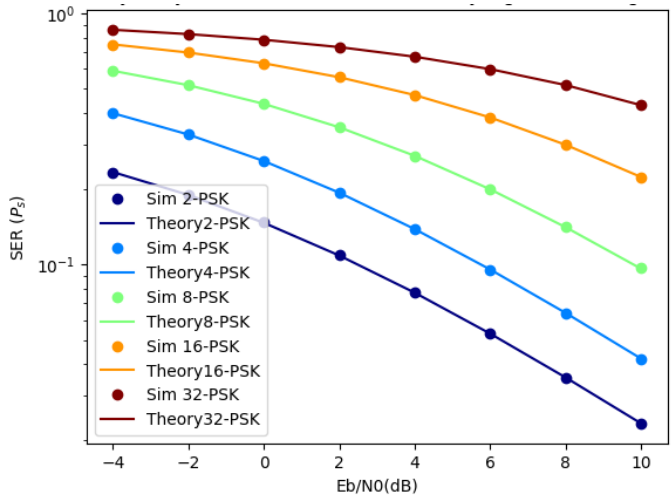


Figure 4-4: M-PSK Rayleigh

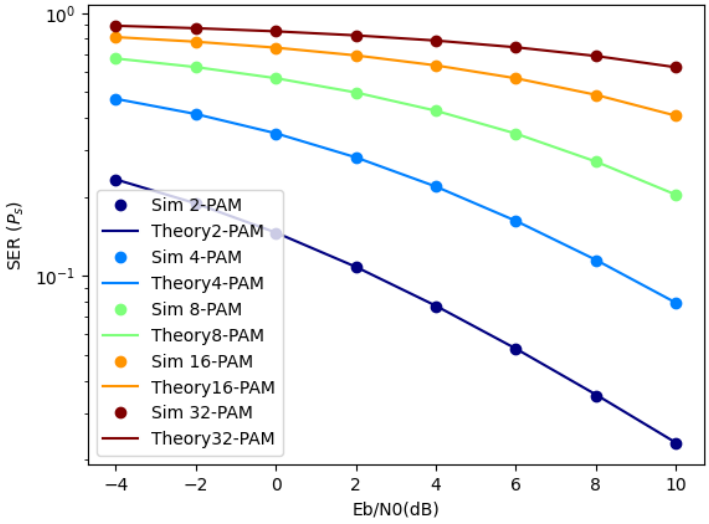


Figure 4-5: M-PAM Rayleigh

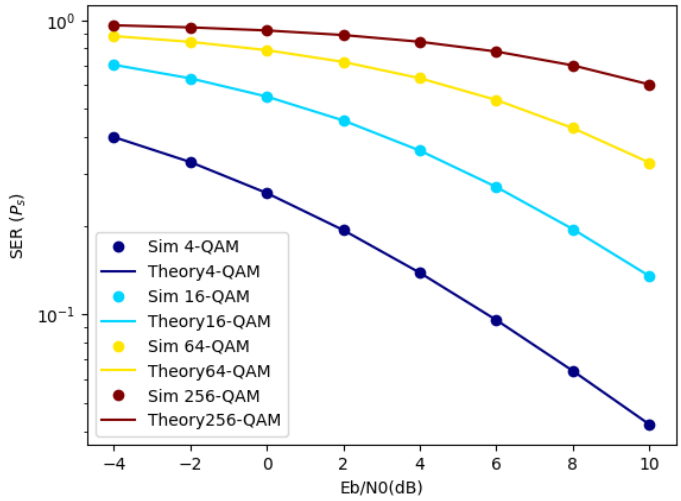


Figure 4-6: M-QAM Rayleigh

As we can observe here the simulated and theoretical curves align proving that the simulation matches the theoretical for Rayleigh Flat fading channel in QAM, PAM and PSK modulation schemes

4.2.3 Uncoded OFDM Results

The results below are done using ideal estimation and synchronization, we are only taking into consideration the channel non-ideality and are trying to overcome it by using an OFDM system with channel coding to correct relative errors.

4.2.3.1 AWGN channel

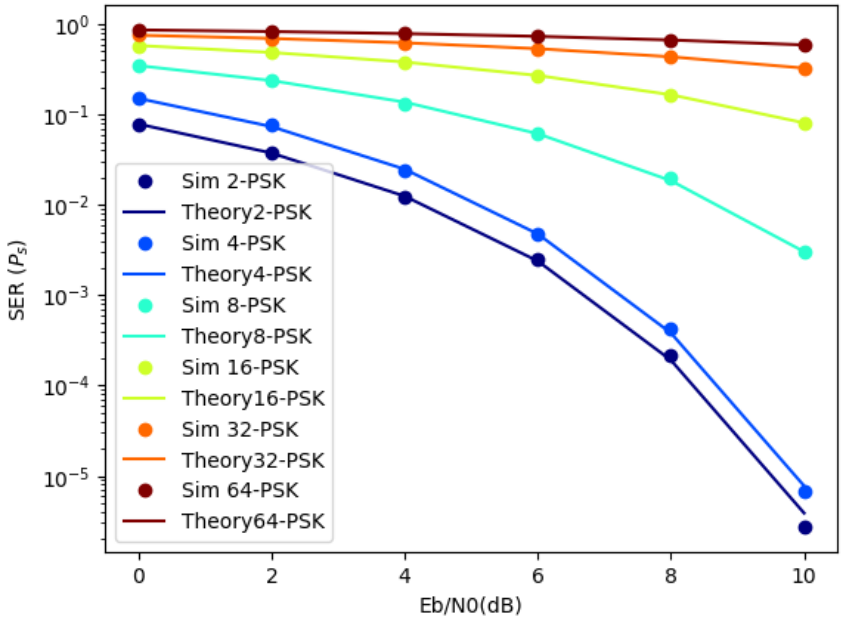


Figure 4-7: M-PSK CP-OFDM AWGN SER

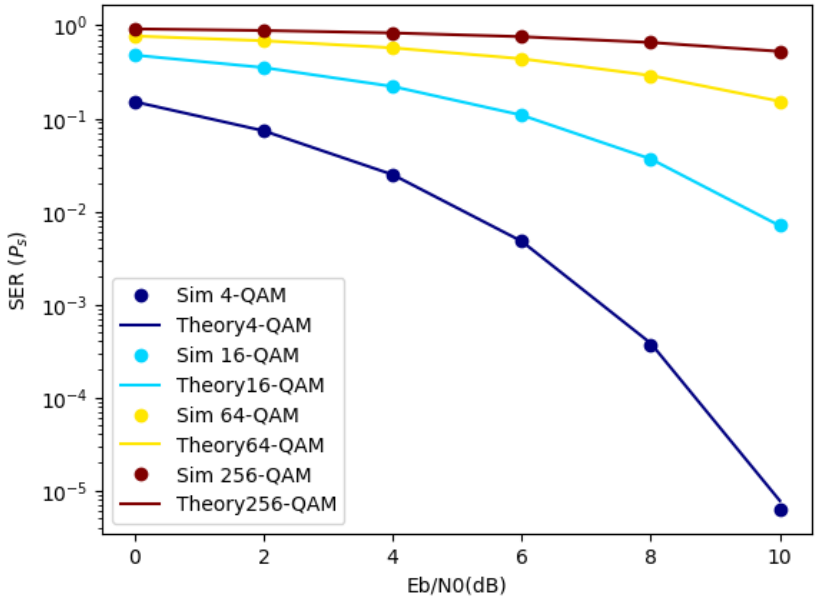


Figure 4-8: M-QAM CP-OFDM AWGN SER

In the scenario of an uncoded Orthogonal Frequency Division Multiplexing (OFDM) system operating in an Additive White Gaussian Noise (AWGN) channel, the results predominantly reflect the expected behaviour. As AWGN channels lack the multipath-induced fading effects present in frequency-selective channels, the inherent advantages of OFDM, particularly its ability to mitigate frequency-selective fading, are less pronounced. Consequently, the simulation outcomes typically align closely with the theoretical curves, revealing minimal performance improvements. Since OFDM is designed to excel in environments characterized by frequency-selective fading, where its subcarriers can adapt to varying channel conditions, the absence of fading in AWGN limits the potential for significant enhancements. In such scenarios, the focus often shifts to optimizing other aspects of the system, such as modulation and coding schemes, to maximize overall performance in the absence of fading-induced challenges.

4.2.3.2 Rayleigh Frequency Selective Channel

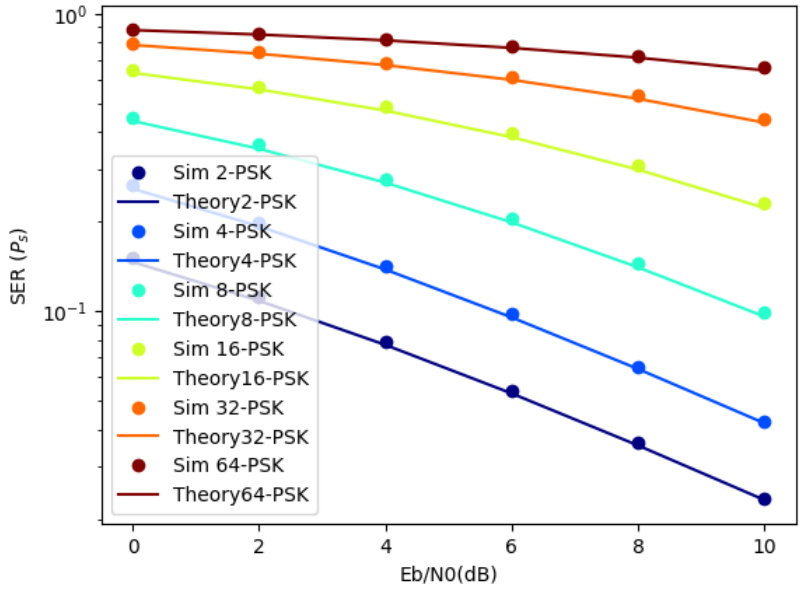


Figure 4-9: M-PSK CP-OFDM RAYLEIGH SER

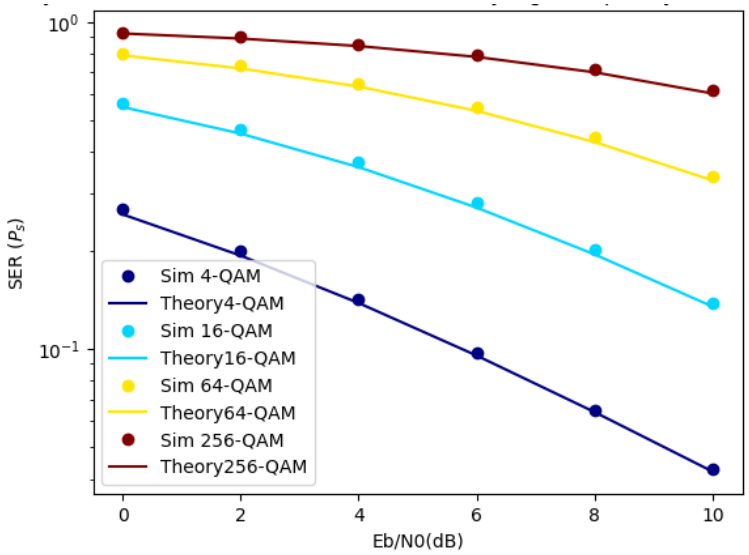


Figure 4-10: M-QAM CP-OFDM RAYLEIGH SER

The dotted curves represent the OFDM simulations under frequency selective channel, the theoretical curves are for a flat fading channel, the 2 curves overlap which means that the OFDM improves the data reception and mitigates the effect of the frequency selective channel as the he available spectrum into multiple narrowband subcarriers. Since these subcarriers are spaced closely together, they experience different frequency-selective fading characteristics. Some subcarriers may experience deep fades due to the channel conditions, but others may not be as affected. This frequency diversity helps in combating selective fading.

: The use of cyclic prefix and guard intervals simplifies the equalization process in the receiver. The receiver can perform a simple frequency-domain equalization by dividing each received subcarrier by its channel gain, which helps in compensating for the frequency-selective fading effects.

4.2.4 Coded OFDM Results

The results below are done using ideal estimation and synchronization, we are only taking into consideration the channel non-ideality and are trying to overcome it by using an OFDM system with channel coding to correct relative errors. Here we included some error correction coding like convolutional encoder and Viterbi decoder, bit interleaving, and windowing

4.2.4.1 AWGN channel

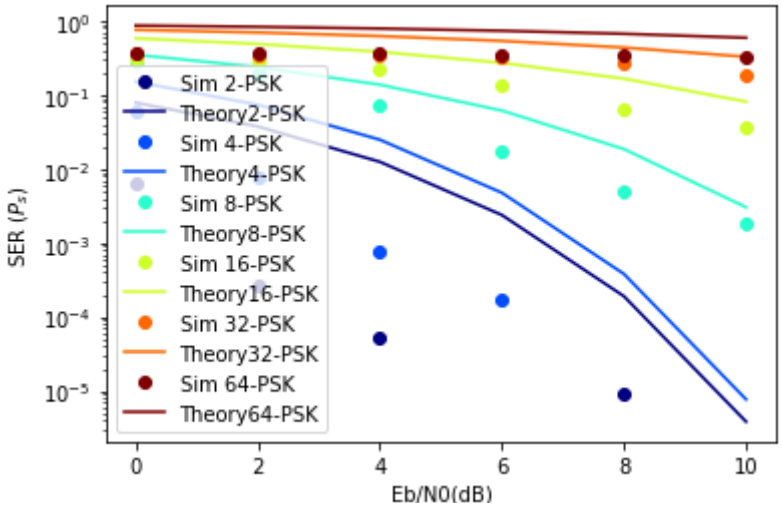


Figure 4-11: M-PSK Coded CP-OFDM AWGN SER

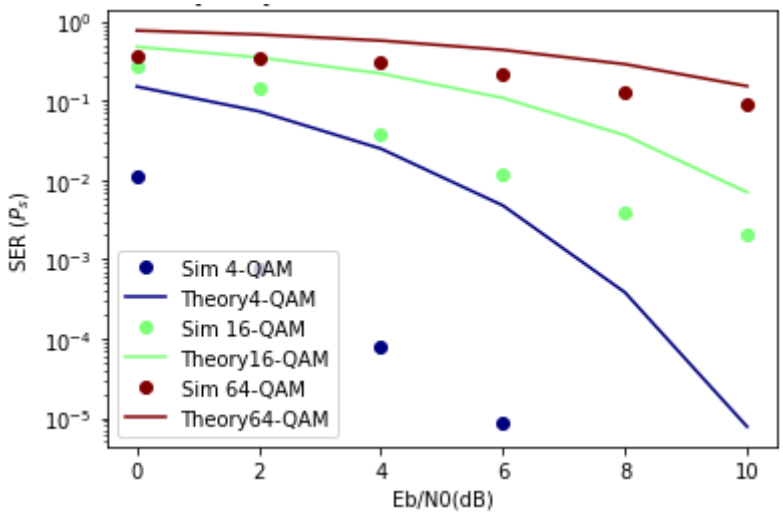


Figure 4-12: M-QAM Coded CP-OFDM AWGN SER

Coded Orthogonal Frequency Division Multiplexing (OFDM) significantly improves communication system performance, particularly in the presence of time-varying channels. By incorporating channel coding and interleaving techniques, the system gains the ability to mitigate burst and random effects induced by the channel's temporal variability. Channel coding introduces redundancy for error correction, making the transmission more robust against noise and interference. Meanwhile, interleaving rearranges the transmitted data, providing temporal diversity that helps combat burst errors by spreading them over time and subcarriers. This combined approach synergizes with OFDM's inherent advantages, such as frequency diversity and efficient spectrum utilization, creating a powerful solution that adapts dynamically to varying channel conditions and enhances overall system reliability.

4.2.4.2 Rayleigh Channel

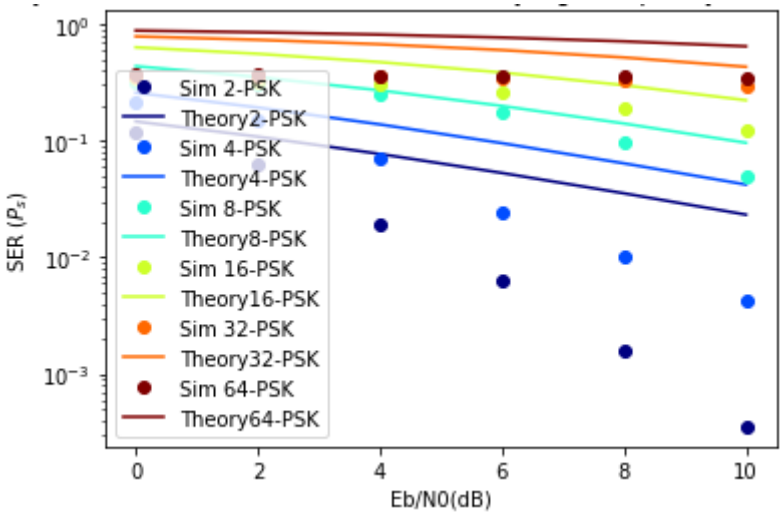


Figure 4-13: M-PSK Coded CP-OFDM RAYLEIGH SER

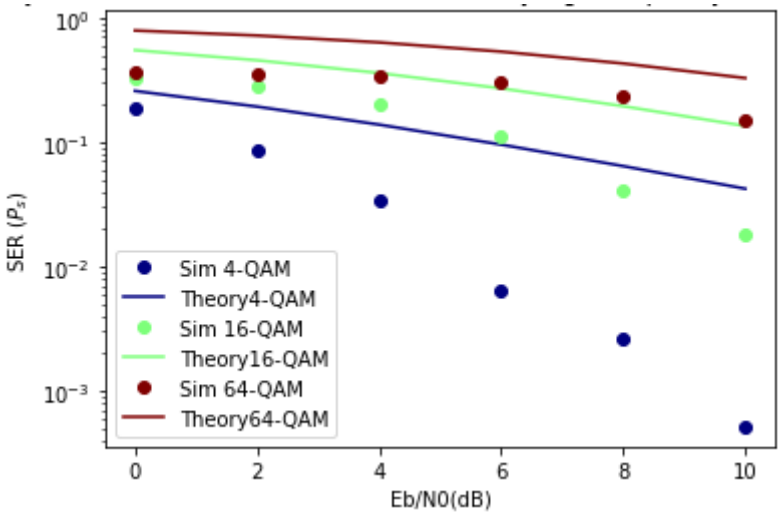


Figure 4-14: M-QAM Coded CP-OFDM RAYLEIGH SER

In the context of Rayleigh fading channels, where the multipath propagation induces fluctuations in signal amplitude and phase, the deployment of Coded Orthogonal Frequency Division Multiplexing (OFDM) is particularly advantageous. By introducing channel coding and interleaving, the system gains resilience against the challenges posed by Rayleigh fading. Additionally, the application of windowing techniques further enhances spectral efficiency. Windowing helps in reducing the interference caused by the abrupt truncation of OFDM symbols, improving the overall performance of the system. Notably, the impact of windowing extends beyond merely aligning theoretical flat fading curves with simulated frequency-selective fading; it also leads to an enhancement in spectral efficiency. The combined effect of windowing and channel coding in Coded OFDM proves instrumental in overcoming the impairments introduced by Rayleigh fading, ensuring robust communication in dynamic wireless environments.

4.2.5 Fixed point uncoded OFDM system

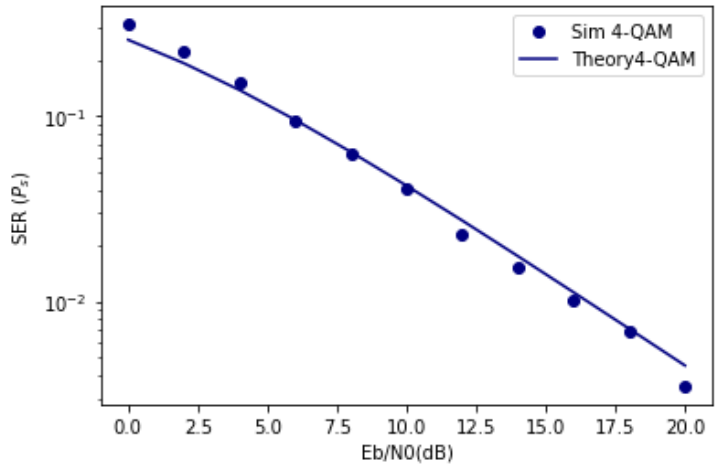


Figure 4-15 Fixed point SER of 4-QAM system

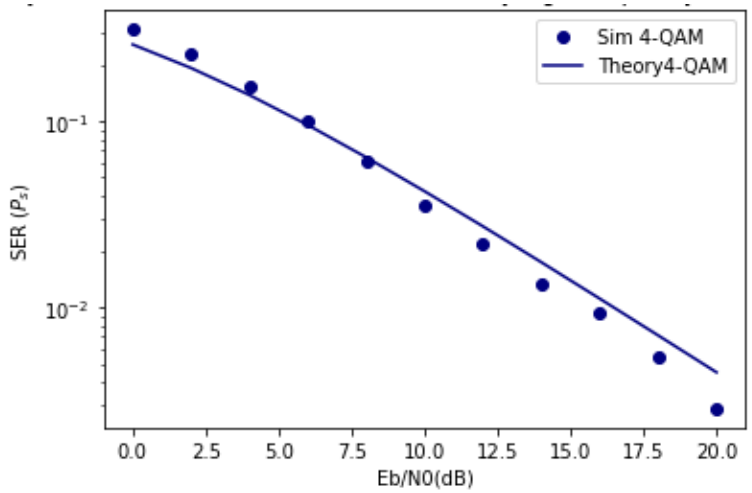


Figure 4-16 Floating point SER of 4-QAM system

The results presented here focus exclusively on the effects of Signal-to-Noise Ratio (SNR) and Root Mean Square (RMS) delay spread of the channel. From the performance data, we observe that the hardware implementation of the receiver closely aligns with the theoretical performance curves for a 4-QAM Orthogonal Frequency-Division Multiplexing (OFDM) receiver. This alignment is evident despite the finite precision limitations inherent in hardware implementations. Notably, floating-point implementations, which offer higher precision, demonstrate a slightly lower Symbol Error Rate (SER) overall, reflecting their superior accuracy compared to fixed-point hardware implementations. In our simulations, the receiver's performance matches the theoretical predictions accurately, as we have utilized an uncoded OFDM receiver. While uncoded systems provide a baseline for performance evaluation, incorporating coding schemes can significantly improve the SER, as detailed in the earlier sections of our analysis. This improvement underscores the potential benefits of coding in mitigating errors and enhancing the robustness of communication systems against channel impairments.

4.3 USRP BLOCK DIAGRAM VERIFICATION

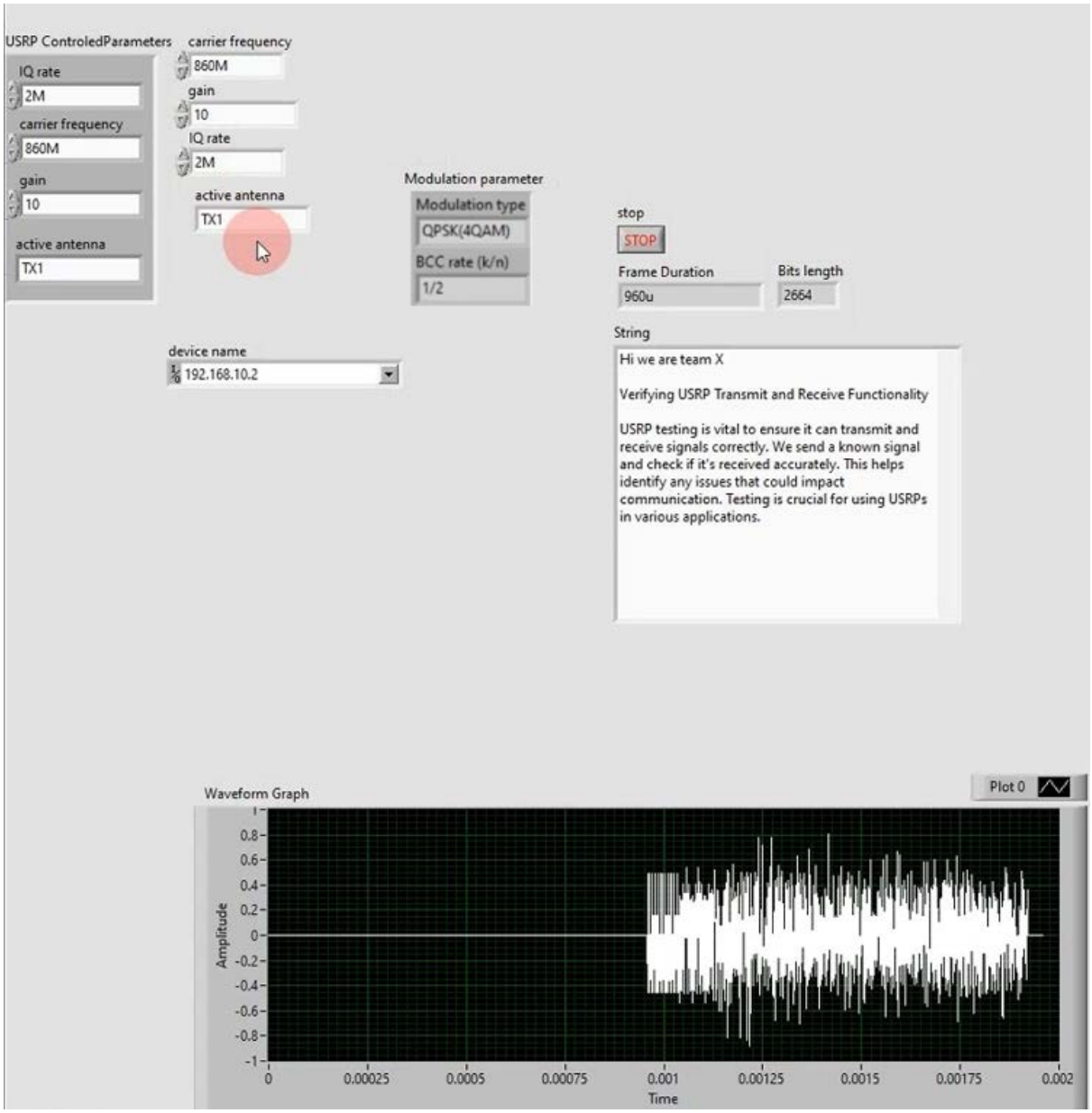


Figure 4-17 Transmitter LabView interface

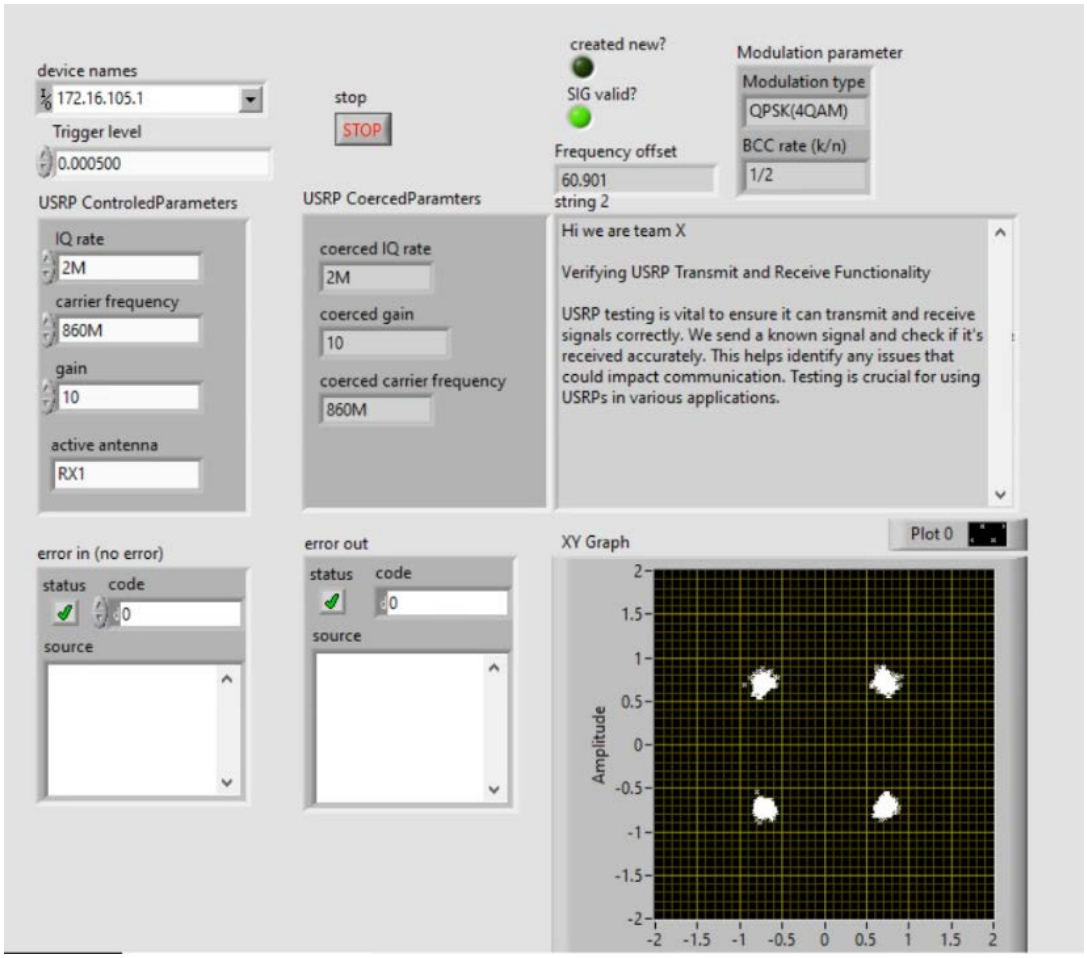


Figure 4-18 Receiver LabVIEW interface

In this experiment, we transmitted a string using our system's modulation scheme, which employs 4-QAM (Qquadrature Amplitude Modulation). The 4-QAM modulation, known for its efficiency in representing data with four distinct symbols, allowed the string to be encoded and sent over the communication channel. Upon transmission, the string was successfully received by the receiver USRP (Universal Software Radio Peripheral), highlighting the reliability of our system's transmission capabilities.

The constellation diagram at the receiver confirmed the use of 4-QAM, showcasing the expected four distinct points, each representing a unique symbol. This clear and accurate constellation pattern indicates minimal distortion and noise interference, ensuring the integrity of the transmitted data. Consequently, the string was correctly decoded, demonstrating the effective performance of our 4-QAM modulation scheme in maintaining data accuracy throughout the communication process.

4.4 HDL SIMULATION RESULTS

Table 4-3 SER using different channel realizations

Test Number	Eb/N0 (dB)	CFO	RMS delay spread (ns)	Number of incorrect symbols
1	15	1.2	25	26/192
2	15	1.85	25	30/192
3	15	-1.5	30	0/192
4	15	0.3	50	8/192
5	15	0.3	80	17/192

In this testing strategy, we maintained a constant Signal-to-Noise Ratio (SNR), as the impact of SNR variations was previously analysed in detail in Section 4.3.5. Our current focus is to isolate and understand the effects of Carrier Frequency Offset (CFO) and delay spread on the channel's performance. The typical number of incorrect symbols, which are crucial for generating the Bit Error Rate (BER) curves presented in Section 4.3.5, is around 15 erroneous bits when the energy per bit to noise power spectral density ratio is at 10 dB. The hardware implementation shows promising performance even under conditions of relatively high Root Mean Square (RMS) delay spread, indicating the system's robustness to delay variations. However, when we introduce a significant CFO, there is a noticeable increase in the number of incorrect symbols, rising to approximately 25-30. Although this represents a deviation from the ideal performance, it is still within acceptable limits for our system. This result suggests that while the system handles high delay spreads well, mitigating the effects of CFO remains a key area for optimization to maintain overall performance and reliability.

4.5 SYNTHESIS REPORTS

4.5.1 Transmitter Implementation results

4.5.1.1 Utilization

Table 4 Transmitter Utilization

	Slice LUTs	Slice Registers	RAM Bits
Available	22320	58993	608256
Utilized	1529	2499	36000
Percentage	6.86	4.24	5.92

4.5.1.2 Verification

In this step we compare the generated outputs with the expected outputs that were generated from the simulation.

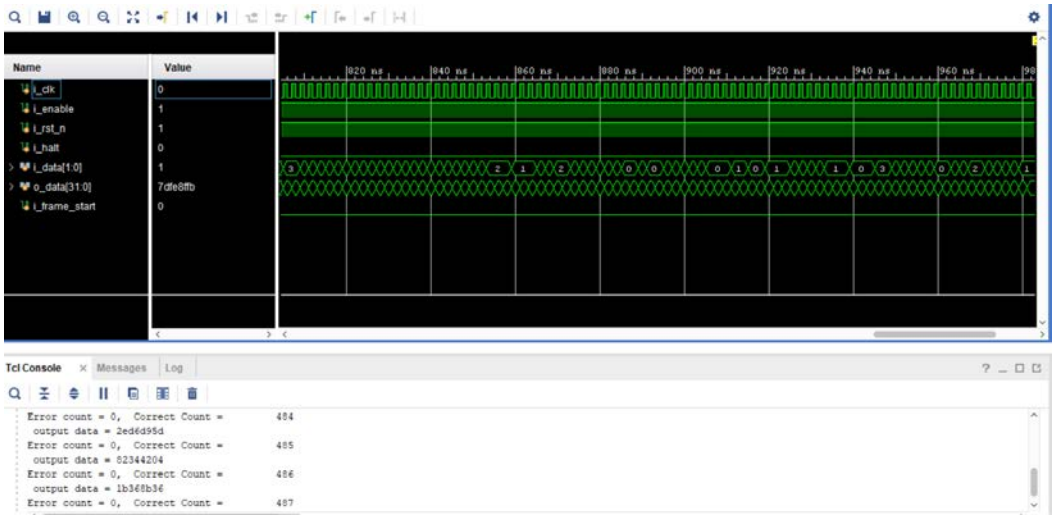


Figure 4-19 Functional simulation Waveform

4.5.2 Receiver Processor Chain

4.5.2.1 System Power and area

```
****************************************
Report : area
Design : top
Version: K-2015.06
Date   : Fri Jun 21 01:47:24 2024
****************************************
Library(s) Used:

    scmetro_tsmc_cl013g_rvt_ss_1p08v_125c (File: /home/IC/Labs/Lab_Syn_2.0/std_cells/
                                        scmetro_tsmc_cl013g_rvt_ss_1p08v_125c.db)

Number of ports:                         7223
Number of nets:                         39272
Number of cells:                        29478
Number of combinational cells:          19424
Number of sequential cells:              9941
Number of macros/black boxes:               0
Number of buf/inv:                       1704
Number of references:                       8

Combinational area:             263450.189043
Buf/Inv area:                     6438.902538
Noncombinational area:          233723.210468
Macro/Black Box area:                0.000000
Net Interconnect area:      undefined  (No wire load specified)

Total cell area:                497173.399511
Total area:                 undefined

Hierarchical area distribution
------------------------------

                                  Global cell area              Local cell area
                                  --------------------  --------------------------------
Hierarchical cell                 Absolute     Percent  Combi-       Noncombi-    Black-
                                  Total        Total    national     national     boxes   Design
-------------------------------- -----------  -------  -----------  -----------  ------  -------
top                               497173.3995    100.0       0.0000       0.0000  0.0000  top
```

Figure 4-20 System area report

```
Global Operating Voltage = 1.08
Power-specific unit information :
    Voltage Units = 1V
    Capacitance Units = 1.000000pf
    Time Units = 1ns
    Dynamic Power Units = 1mW    (derived from V,C,T units)
    Leakage Power Units = 1pW


--------------------------------------------------------------------------------
                                   Switch   Int      Leak     Total
Hierarchy                          Power    Power    Power    Power    %
--------------------------------------------------------------------------------
top                                   0.293    9.239 3.56e+08    9.888 100.0
```

Figure 4-21 System power report

4.5.2.2 Fast Fourier Transform FFT

Hierarchical area distribution

Hierarchical cell	Global cell area: Absolute Total	Global cell area: Percent Total	Local cell area: Combi-national	Local cell area: Noncombi-national	Local cell area: Black-boxes	Design
FFT	335374.7945	67.5	0.0000	0.0000	0.0000	FFT
FFT/SDF_1	80237.9952	16.1	2787.6023	2711.1167	0.0000	SdfUnit_N64_M64_WIDTH16
FFT/SDF_1/BF1	1986.2696	0.4	0.0000	0.0000	0.0000	Butterfly_WIDTH16_RH0_0
FFT/SDF_1/BF2	1986.2696	0.4	0.0000	0.0000	0.0000	Butterfly_WIDTH16_RH1_0
FFT/SDF_1/DB1	21688.9336	4.4	0.0000	21688.9336	0.0000	DelayBuffer_DEPTH32_WIDTH16
FFT/SDF_1/DB2	10844.4668	2.2	0.0000	10844.4668	0.0000	DelayBuffer_DEPTH16_WIDTH16
FFT/SDF_1/MU	35955.2454	7.2	0.0000	0.0000	0.0000	Multiply_WIDTH16_0
FFT/SDF_1/TW	2278.0912	0.5	1600.3120	677.7792	0.0000	Twiddle_0
FFT/SDF_2	55734.3953	11.2	2681.6993	2713.4701	0.0000	SdfUnit_N64_M16_WIDTH16
FFT/SDF_2/BF1	1986.2696	0.4	0.0000	0.0000	0.0000	Butterfly_WIDTH16_RH0_2
FFT/SDF_2/BF2	1986.2696	0.4	0.0000	0.0000	0.0000	Butterfly_WIDTH16_RH1_2
FFT/SDF_2/DB1	5422.2334	1.1	0.0000	5422.2334	0.0000	DelayBuffer_DEPTH8_WIDTH16
FFT/SDF_2/DB2	2711.1167	0.5	0.0000	2711.1167	0.0000	DelayBuffer_DEPTH4_WIDTH16
FFT/SDF_2/MU	35955.2454	7.2	0.0000	0.0000	0.0000	Multiply_WIDTH16_2
FFT/SDF_2/TW	2278.0912	0.5	1600.3120	677.7792	0.0000	Twiddle_2
FFT/SDF_3	12568.3326	2.5	2271.0310	2013.3337	0.0000	SdfUnit_N64_M4_WIDTH16
FFT/SDF_3/BF1	1986.2696	0.4	0.0000	0.0000	0.0000	Butterfly_WIDTH16_RH0_1
FFT/SDF_3/DB1	1355.5583	0.3	0.0000	1355.5583	0.0000	DelayBuffer_DEPTH2_WIDTH16
FFT/SDF_3/DB2	677.7792	0.1	0.0000	677.7792	0.0000	DelayBuffer_DEPTH1_WIDTH16
FFT/SDF_3/TW	2278.0912	0.5	1600.3120	677.7792	0.0000	Twiddle_1
FFT/address_gen	4351.4365	0.9	236.5167	217.6895	0.0000	address_gen
FFT/memory_1	89031.4748	17.9	44712.2476	44319.2272	0.0000	mem1
FFT/memory_2	89031.4748	17.9	44712.2476	44319.2272	0.0000	mem2
FFT/mux0	405.9615	0.1	376.5440	0.0000	0.0000	out_mux

Figure 4-22 Area Report of FFT

Hierarchy	Switch Power	Int Power	Leak Power	Total Power	%
FFT (FFT)	0.169	6.536	2.47e+08	6.952	70.3
memory_2 (mem2)	5.18e-03	1.680	6.68e+07	1.752	17.7
memory_1 (mem1)	5.18e-03	1.680	6.69e+07	1.752	17.7
address_gen (address_gen)	3.86e-02	0.150	1.88e+06	0.191	1.9
SDF_3 (SdfUnit_N64_M4_WIDTH16)	5.55e-03	0.179	9.07e+06	0.194	2.0
TW (Twiddle_1)	0.000	2.56e-02	1.31e+06	2.69e-02	0.3
DB2 (DelayBuffer_DEPTH1_WIDTH16)	7.27e-06	2.57e-02	4.31e+05	2.61e-02	0.3
BF2 (Butterfly_WIDTH16_RH1_1)	2.01e-05	1.02e-04	1.81e+06	1.94e-03	0.0
DB1 (DelayBuffer_DEPTH2_WIDTH16)	4.36e-05	5.16e-02	8.63e+05	5.25e-02	0.5
BF1 (Butterfly_WIDTH16_RH0_1)	6.24e-05	2.94e-04	1.82e+06	2.17e-03	0.0
SDF_2 (SdfUnit_N64_M16_WIDTH16)	8.00e-03	0.463	4.19e+07	0.513	5.2
MU (Multiply_WIDTH16_2)	9.47e-04	1.83e-03	2.81e+07	3.08e-02	0.3
TW (Twiddle_2)	6.43e-06	2.56e-02	1.31e+06	2.69e-02	0.3
DB2 (DelayBuffer_DEPTH4_WIDTH16)	1.34e-04	0.105	1.73e+06	0.107	1.1
BF2 (Butterfly_WIDTH16_RH1_2)	2.25e-04	1.18e-03	1.82e+06	3.23e-03	0.0
DB1 (DelayBuffer_DEPTH8_WIDTH16)	5.20e-04	0.219	3.47e+06	0.223	2.3
BF1 (Butterfly_WIDTH16_RH0_2)	5.22e-04	3.66e-03	1.84e+06	6.02e-03	0.1
SDF_1 (SdfUnit_N64_M64_WIDTH16)	6.18e-02	2.242	5.86e+07	2.363	23.9
MU (Multiply_WIDTH16_0)	7.35e-03	1.47e-02	2.82e+07	5.03e-02	0.5
TW (Twiddle_0)	6.43e-06	2.56e-02	1.31e+06	2.69e-02	0.3
DB2 (DelayBuffer_DEPTH16_WIDTH16)	2.70e-03	0.515	7.04e+06	0.525	5.3
BF2 (Butterfly_WIDTH16_RH1_0)	1.97e-03	1.40e-02	1.90e+06	1.79e-02	0.2
DB1 (DelayBuffer_DEPTH32_WIDTH16)	1.42e-02	1.317	1.44e+07	1.345	13.6
BF1 (Butterfly_WIDTH16_RH0_0)	2.26e-02	0.205	1.96e+06	0.229	2.3

Figure 4-23 Power Report of FFT

4.5.2.3 Pilot-null detector

```
Hierarchical area distribution
------------------------------

                                Global cell area              Local cell area
                                --------------------  --------------------------------
Hierarchical cell               Absolute     Percent  Combi-       Noncombi-    Black-
                                Total        Total    national     national     boxes   Design
------------------------------  -----------  -------  -----------  -----------  ------  ------------------------
pilot_null_detector              11529.3065      2.3    3591.2884    5642.2765  0.0000  pilot_null_detector
pilot_null_detector/d_2             56.4816      0.0       3.5301      52.9515  0.0000  delay_n_D2
pilot_null_detector/d_64          2239.2601      0.5       4.7068    2234.5533  0.0000  delay_n_D81
```

Figure 4-24 Pilot Null detector area report

```
--------------------------------------------------------------------------------
                                    Switch   Int      Leak     Total
Hierarchy                           Power    Power    Power    Power    %
--------------------------------------------------------------------------------
  pilot_null_detector (pilot_null_detector)
                                    7.34e-02    0.293 5.63e+06    0.372   3.8
    d_2 (delay_n_D2)                5.93e-04 1.86e-03 2.45e+04 2.48e-03   0.0
    d_64 (delay_n_D81)              1.81e-02 8.78e-02 9.90e+05    0.107   1.1
```

Figure 4-25 pilot null detector power report

4.5.2.4 Demodulator

```
Hierarchical area distribution
------------------------------

                                Global cell area              Local cell area
                                --------------------  --------------------------------
Hierarchical cell               Absolute     Percent  Combi-       Noncombi-    Black-
                                Total        Total    national     national     boxes   Design
------------------------------  -----------  -------  -----------  -----------  ------  --------------------------------
```

Figure 4-27 demodulator area report

```
--------------------------------------------------------------------------------
                                    Switch   Int      Leak     Total
Hierarchy                           Power    Power    Power    Power    %
--------------------------------------------------------------------------------
  demodulator (demod)               8.51e-04 2.84e-03 6.33e+04 3.76e-03   0.0
    u1 (delay_3)                    2.84e-04 9.89e-04 1.21e+04 1.29e-03   0.0
```

Figure 4-26 demodulator power report

4.5.2.5 Channel equalizer

```
Hierarchical area distribution
------------------------------

                                Global cell area             Local cell area
                                --------------------  --------------------------------
Hierarchical cell               Absolute     Percent  Combi-       Noncombi-    Black-
                                Total        Total    national     national     boxes   Design
-------------------------------  -----------  -------  -----------  -----------  ------  ----------------------
channel_equalizer               42853.0605       8.6     909.5891   10432.6220  0.0000  cmult
channel_equalizer/add_108         584.8199       0.1     584.8199       0.0000  0.0000  cmult_DW01_add_3
channel_equalizer/add_120         497.7441       0.1     497.7441       0.0000  0.0000  cmult_DW01_add_2
channel_equalizer/add_123         584.8199       0.1     584.8199       0.0000  0.0000  cmult_DW01_add_0
channel_equalizer/mult_106       9226.5047       1.9    8957.0404       0.0000  0.0000  cmult_DW02_mult_1
channel_equalizer/mult_106/FS_1   269.4643       0.1     269.4643       0.0000  0.0000  cmult_DW01_add_4
channel_equalizer/mult_121       9226.5047       1.9    8957.0404       0.0000  0.0000  cmult_DW02_mult_0
channel_equalizer/mult_121/FS_1   269.4643       0.1     269.4643       0.0000  0.0000  cmult_DW01_add_1
channel_equalizer/mult_93        9226.5047       1.9    8957.0404       0.0000  0.0000  cmult_DW02_mult_2
channel_equalizer/mult_93/FS_1    269.4643       0.1     269.4643       0.0000  0.0000  cmult_DW01_add_5
channel_equalizer/sub_105         556.5791       0.1     556.5791       0.0000  0.0000  cmult_DW01_sub_0
channel_equalizer/sub_92          556.5791       0.1     556.5791       0.0000  0.0000  cmult_DW01_sub_1
channel_equalizer/u0              184.7419       0.0       3.5301     181.2118  0.0000  delay_n_D7
channel_equalizer/u1               30.5942       0.0       3.5301      27.0641  0.0000  delay_0
channel_equalizer/u2              417.7285       0.1       3.5301     414.1984  0.0000  delay_w_D16_0
channel_equalizer/u3              417.7285       0.1       3.5301     414.1984  0.0000  delay_w_D16_1
memory_lts                      59284.5013      11.9    7735.6257   51548.8756  0.0000  mem1_N48
ref_symbols_lut                  2391.0544       0.5    1081.3873    1309.6671  0.0000  ROM_ref
```

Figure 4-28 Channel equalizer area report

```
--------------------------------------------------------------------------------
                                    Switch   Int      Leak     Total
Hierarchy                           Power    Power    Power    Power    %
--------------------------------------------------------------------------------
  channel_equalizer (cmult)         1.31e-02    0.431 3.19e+07    0.476   4.8
    sub_92 (cmult_DW01_sub_1)          0.000    0.000 5.17e+05 5.17e-04   0.0
    mult_93 (cmult_DW02_mult_2)        0.000    0.000 7.17e+06 7.17e-03   0.1
    sub_105 (cmult_DW01_sub_0)         0.000    0.000 5.17e+05 5.17e-04   0.0
    mult_106 (cmult_DW02_mult_1)       0.000    0.000 7.17e+06 7.17e-03   0.1
    add_108 (cmult_DW01_add_3)         0.000    0.000 4.70e+05 4.70e-04   0.0
    add_120 (cmult_DW01_add_2)         0.000    0.000 4.46e+05 4.46e-04   0.0
    mult_121 (cmult_DW02_mult_0)       0.000    0.000 7.17e+06 7.17e-03   0.1
    add_123 (cmult_DW01_add_0)         0.000    0.000 4.70e+05 4.70e-04   0.0
    u3 (delay_w_D16_1)              4.54e-03 1.39e-02 1.67e+05 1.86e-02   0.2
    u2 (delay_w_D16_0)              4.54e-03 1.39e-02 1.67e+05 1.86e-02   0.2
    u1 (delay_0)                    3.10e-04 1.00e-03 1.42e+04 1.33e-03   0.0
    u0 (delay_n_D7)                 1.99e-03 6.16e-03 7.42e+04 8.22e-03   0.1
  ref_symbols_lut (ROM_ref)         1.16e-02 4.62e-02 1.06e+06 5.89e-02   0.6
  memory_lts (mem1_N48)             7.59e-03    1.458 3.64e+07    1.502  15.2
```

Figure 4-29 Channel equalizer power report

4.5.2.6 Phase Offset equalizer

Hierarchical area distribution

Hierarchical cell	Global cell area: Absolute Total	Global cell area: Percent Total	Local cell area: Combi-national	Local cell area: Noncombi-national	Local cell area: Black-boxes	Design
pilot_avg	4311.4288	0.9	1266.1292	1927.4346	0.0000	pilot_avg
pilot_avg/add_44	558.9325	0.1	558.9325	0.0000	0.0000	pilot_avg_DW01_add_1
pilot_avg/add_45	558.9325	0.1	558.9325	0.0000	0.0000	pilot_avg_DW01_add_0
pilot_complex_mult	41316.2903	8.3	264.7575	10432.6220	0.0000	cmult_p
pilot_complex_mult/add_100	584.8199	0.1	584.8199	0.0000	0.0000	cmult_p_DW01_add_0
pilot_complex_mult/add_85	584.8199	0.1	584.8199	0.0000	0.0000	cmult_p_DW01_add_3
pilot_complex_mult/add_97	497.7441	0.1	497.7441	0.0000	0.0000	cmult_p_DW01_add_2
pilot_complex_mult/mult_70	9226.5047	1.9	8957.0404	0.0000	0.0000	cmult_p_DW02_mult_2
pilot_complex_mult/mult_70/FS_1	269.4643	0.1	269.4643	0.0000	0.0000	cmult_p_DW01_add_5
pilot_complex_mult/mult_83	9226.5047	1.9	8957.0404	0.0000	0.0000	cmult_p_DW02_mult_1
pilot_complex_mult/mult_83/FS_1	269.4643	0.1	269.4643	0.0000	0.0000	cmult_p_DW01_add_4
pilot_complex_mult/mult_98	9226.5047	1.9	8957.0404	0.0000	0.0000	cmult_p_DW02_mult_0
pilot_complex_mult/mult_98/FS_1	269.4643	0.1	269.4643	0.0000	0.0000	cmult_p_DW01_add_1
pilot_complex_mult/sub_69	556.5791	0.1	556.5791	0.0000	0.0000	cmult_p_DW01_sub_1
pilot_complex_mult/sub_82	556.5791	0.1	556.5791	0.0000	0.0000	cmult_p_DW01_sub_0
pilot_complex_mult/u0	158.8545	0.0	3.5301	155.3244	0.0000	delay_n_D6

Figure 4-30 phase offset equalizer area report

Hierarchy	Switch Power	Int Power	Leak Power	Total Power	%
pilot_avg (pilot_avg)	1.82e-02	7.06e-02	2.65e+06	9.14e-02	0.9
add_44 (pilot_avg_DW01_add_1)	0.000	0.000	5.00e+05	5.00e-04	0.0
add_45 (pilot_avg_DW01_add_0)	0.000	0.000	5.00e+05	5.00e-04	0.0
pilot_complex_mult (cmult_p)	1.72e-03	0.401	3.10e+07	0.434	4.4
sub_69 (cmult_p_DW01_sub_1)	0.000	0.000	5.17e+05	5.17e-04	0.0
mult_70 (cmult_p_DW02_mult_2)	0.000	0.000	7.17e+06	7.17e-03	0.1
sub_82 (cmult_p_DW01_sub_0)	0.000	0.000	5.17e+05	5.17e-04	0.0
mult_83 (cmult_p_DW02_mult_1)	0.000	0.000	7.17e+06	7.17e-03	0.1
add_85 (cmult_p_DW01_add_3)	0.000	0.000	4.70e+05	4.70e-04	0.0
add_97 (cmult_p_DW01_add_2)	0.000	0.000	4.46e+05	4.46e-04	0.0
mult_98 (cmult_p_DW02_mult_0)	0.000	0.000	7.17e+06	7.17e-03	0.1
add_100 (cmult_p_DW01_add_0)	0.000	0.000	4.70e+05	4.70e-04	0.0
u0 (delay_n_D6)	1.70e-03	5.30e-03	6.39e+04	7.07e-03	0.1

Figure 4-31 phase offset equalizer power report

The receiver in question encompasses a total area of approximately 500,000 square micrometres, a metric that is essential in the domain of microelectronics and semiconductor manufacturing. This area represents the physical space occupied by the receiver's components on a silicon die. Smaller areas are typically desirable as they reduce the cost of production and can enhance the overall performance by minimizing the electrical path lengths. The total power consumption of the receiver is around 10 milliwatts (mW), a relatively low figure that indicates high energy efficiency. This is particularly significant in applications where the device operates on battery power or needs to maintain minimal heat dissipation. Low power consumption not only extends battery life in portable devices but also contributes to the reliability and longevity of the receiver by reducing thermal stress.

A critical component within this receiver is the Fast Fourier Transform (FFT) unit, which is integral to signal processing tasks. The FFT unit is remarkably demanding, accounting for 70% of both the total power consumption and the overall area of the receiver. This translates to the FFT consuming 7 mW of power and occupying 350,000 micrometers of the area. The FFT's high resource usage stems from its need to perform complex multiplications and manage extensive memory for lookup tables. These multiplications involve handling real and imaginary numbers, which are computationally intensive and thus require significant energy. Additionally, the FFT unit's reliance on memory for storing and retrieving data efficiently adds to its area and power consumption. The lookup tables, essential for speeding up the transform process, necessitate substantial memory resources, which are both space-consuming and power-hungry. Consequently, the FFT unit's demands highlight the challenges in balancing performance, power efficiency, and area in modern receiver design.

4.6 CONCLUSION

From the findings presented in the preceding section, it is evident that a discernible shift exists between the theoretical and simulated Symbol Error Rates (SER). Remarkably, this shift is in a downward direction, signifying a reduction in error rates upon the implementation of the Orthogonal Frequency Division Multiplexing (OFDM) system. The introduction of OFDM serves as a countermeasure against errors induced by the channel.

In comparing the results obtained in Additive White Gaussian Noise (AWGN) scenarios with those in Rayleigh frequency-selective channels, it becomes apparent that the shift in SER is less pronounced in the former. This disparity can be attributed to the intrinsic suitability of the OFDM system for addressing frequency-selective fading. This adaptability arises from the integration of an interleaver and Forward Error Correction (FEC) encoder, both of which were previously highlighted. Additionally, the OFDM system effectively mitigates the multipath problem.

It is essential to note that while the theoretical curves for Rayleigh fading consider a flat fading channel, the channel utilized in the OFDM system is inherently frequency selective. Despite this discrepancy, OFDM not only aligns with the theoretical curves but also demonstrates improved SER performance when confronted with frequency selectivity.

5 MIMO-GANS

5.1 INTRODUCTION

Overview

The MIMO-GAN model is a pivotal advancement within the broader scope of the project, which focuses on the implementation and analysis of the physical layer of the IEEE 802.11 standard. The project's comprehensive approach includes floating point analysis, fixed point analysis, and a hardware implementation on an FPGA. The initial stages of the project utilized a SISO (Single Input Single Output) channel simulator, employing a TDL (Tapped Delay Line) model and signal processing techniques to conduct the floating-point analysis. As wireless communication standards evolve to incorporate MIMO (Multiple Input Multiple Output) technology, the relevance and application of the MIMO-GAN model become increasingly significant.

Extending Floating Point Analysis

The floating-point analysis in the project has relied on SISO channel models to evaluate the performance of the physical layer of the IEEE 802.11 standard. However, newer iterations of the 802.11 standard, such as 802.11n, 802.11ac, and 802.11ax, have introduced MIMO technology to enhance data throughput, reliability, and overall network performance. These standards leverage multiple antennas at both the transmitter and receiver to exploit spatial diversity, improve spectral efficiency, and achieve higher data rates.

The MIMO-GAN model is directly relevant to these newer standards as it provides a sophisticated tool for simulating MIMO channel conditions, crucial for accurate floating-point analysis. The generative capabilities of the MIMO-GAN model allow it to replicate the complex multipath environments characteristic of MIMO systems, ensuring that the floating-point analysis reflects real-world scenarios more accurately than traditional SISO models.

Integration with Existing Analyses

By incorporating the MIMO-GAN model into the floating-point analysis, the project can extend its evaluation framework to include the physical layer performance of MIMO-based IEEE 802.11 standards. The model's ability to generate realistic MIMO channel samples enhances the accuracy of simulation results, providing deeper insights into the performance of MIMO systems under various channel conditions. This is particularly relevant for assessing the impact of channel variations on signal integrity and data throughput.

Conclusion

The MIMO-GAN model plays a crucial role in advancing the project's objectives by enabling the simulation of MIMO channels, which are essential for evaluating the physical layer of newer IEEE 802.11 standards. Its integration into the floating-point analysis framework allows the project to accurately assess the performance of MIMO systems, aligning with the evolving landscape of wireless communications. The MIMO-GAN model's relevance extends beyond floating point analysis, influencing fixed point analysis and hardware implementations, thereby providing a comprehensive toolset for developing and validating next-generation wireless technologies.

5.2 PROPOSED SOLUTION NEURAL SURROGATE FOR MIMO CHANNEL

This chapter is adapted from [30]

5.2.1 Generative Data-driven Neural Surrogate for Channel Models

In the domain of wireless communications, the accurate modeling of channels is pivotal for the design and analysis of communication systems. Traditionally, statistical channel models such as Rayleigh and Rician fading models, as well as those standardized by 3GPP like TDL and CDL, have been extensively used for benchmarking and system design. These models typically employ simplified, predefined delay profiles and stochastic channel gains to facilitate easy sampling and rapid benchmarking across generic scenarios.

However, these traditional models often fall short in capturing the true variability and complexity of field data distributions, which can lead to performance degradation when systems designed based on these models are deployed in real-world scenarios. This discrepancy is becoming increasingly critical as the complexity of communication environments continues to escalate, driven by the advent of massive MIMO systems and the expansion into mmWave and Terahertz frequency bands.

To address these challenges, recent advances in the field of machine learning, particularly in generative modeling, offer promising new avenues for channel modeling. Generative models, such as Generative Adversarial Networks (GANs), provide powerful tools for learning complex data distributions directly from data, without the need for predefined model structures. This capability makes them highly suitable for applications where the underlying data distributions are difficult to model with traditional techniques.
Here are some of the advantages of using a generative data-driven neural surrogate for channel modeling:

- **Enhancing Simulation Efficiency:** GANs offers tractability and ease of sampling, crucial for speeding up simulation rounds. It utilizes neural networks for parallel computation, enhancing the efficiency of simulating large-scale and complex channel models like those in mmWave and Terahertz applications. This efficiency is vital for rapid prototyping and testing in dynamic communication environments.

- **Parallel Computation and System Design Integration:** The neural implementation of GANs allows for inherent parallel computation, reducing simulation time and improving testing throughput. The differentiability of the model supports its integration into end-to-end communication system designs, enabling gradient-based optimization and adaptive system tuning. This property is particularly beneficial for developing systems that require gradient-based optimization techniques.

- **Addressing Non-Linear Effects:** While initially assuming a linear channel model, GANs can be extended to include non-linear mappings. This adaptation allows for more accurate simulations of non-linear effects due to hardware impairments, enhancing the model's practical applicability. By incorporating non-linear dynamics into the model, GANs can provide more accurate simulations that reflect the performance of physical communication systems under practical operating conditions.

5.2.2 System Model

Consider a static MIMO channel with its discrete impulse response $H(\tau) \in \mathbb{C}^{N_R \times N_T}$ at a fixed sampling frequency given by:[31]

$$H(\tau) = \begin{pmatrix} h_{1,1}(\tau) & h_{1,2}(\tau) & \cdots & h_{1,N_T}(\tau) \\ h_{2,1}(\tau) & h_{2,2}(\tau) & \cdots & h_{2,N_T}(\tau) \\ \vdots & \vdots & \ddots & \vdots \\ h_{N_R,1}(\tau) & h_{N_R,2}(\tau) & \cdots & h_{N_R,N_T}(\tau) \end{pmatrix}$$

where $\tau = 0, \ldots, L-1$, and L is the number of taps. The transmit signal at sample n with the sample sampling frequency is given by $x[n]$. The receive signal is given by the discrete convolution:

$$y = H * x.$$

We choose T samples from the input and output of the channel. T is chosen large enough to cover the communication duration. Note that the sampling frequency choice determines the maximum bandwidth we can simulate using the proposed generative model. We denote the transmit signal from j'th antenna as $x_j \in \mathbb{C}^T$ and the receive signal from i'th antenna as $y_i \in \mathbb{C}^T$.

5.2.3 Motivation and Key Idea

Motivation

The advancement of channel modelling in wireless communication faces significant challenges with traditional deep learning methods that rely on explicitly learning the input-output relationship using functions like $f_\theta(x, z)$. These methods, while initially promising due to their ability to model complex behaviors, often fall short in dynamic environments where input conditions and system configurations frequently change. Traditional models typically require extensive retraining to adapt to new scenarios such as changes in cyclic prefix length, demonstrating a critical flexibility gap. Moreover, their inability to effectively handle out-of-distribution data often results in unreliable and erratic outputs, highlighting the need for a more robust and adaptable modelling approach.

Key Idea

To overcome these shortcomings, MIMO-GAN introduces an innovative approach by implicitly learning the input-output relation through the generative model function $f_\theta(x, z)$. This method shifts focus from the explicit and rigid modeling techniques to a more flexible, generative framework capable of simulating a variety of channel conditions dynamically. The function $f_\theta(x, z)$ operates within a Generative Adversarial Network (GAN), where it serves as the generator, producing diverse channel outputs that closely mimic real-world conditions. The stochastic component z enriches the model by introducing randomness, allowing the generator to cover an extensive range of possible channel states without the need for retraining.

The training process involves an adversarial setup with a discriminator that challenges f_θ by distinguishing between generated and actual channel data. This interaction not only refines the generator's output to be indistinguishable from true measurements but also enhances the model's generalizability and robustness against novel or changing conditions. At inference, the model can dynamically simulate different channel responses by varying z, providing a powerful tool for system analysis and design that remains effective across a spectrum of scenarios.

5.2.4 Model Architecture

5.2.4.1 Generator

The backbone of our generative channel model is a Deep Neural Network (DNN) generator model G, which is pivotal in transforming a noise vector $\mathbf{z} \sim \mathcal{N}(0, I)$ into channel samples from the distribution of interest $p(\mathbf{h})$. This transformation is specifically tailored to generate channel instances that are conditioned on the spatial coordinates of the transmit-receive (tx-rx) antenna pairs, denoted as (i, j). The model employs 1-hot encodings for these coordinates, embedding them into a D-dimensional space—set to 4 in our experiments—using a linear layer. This setup allows the generator G to condition the channel instance generation effectively.[32]

The conditional generation of channels based on embedded link-level information provides several strategic advantages. Firstly, because the parameters of G are shared across all channel instances $h_{i,j}$ for any link (i, j), the network is incentivized to learn and utilize correlations between different links. This shared parameter approach not only enhances the network's ability to generalize across different spatial configurations but also ensures computational efficiency.

Secondly, parameter efficiency is significantly improved as the size of G's parameters remain constant despite variations in antenna array sizes. This attribute is crucial for scalability, allowing the model to adapt to arrays of varying dimensions without necessitating an increase in the number of trainable parameters.

Thirdly, the flexibility of G allows it to be employed across different network configurations without retraining. For instance, the same network trained for an 8x8 configuration can be utilized for 4x4 or 8x2 setups, assuming critical spacing is maintained. This capability demonstrates G's adaptability and the practical utility of the model in diverse operational contexts.

To construct and sample the MIMO channel H, we parallelize iterations over the spatial co-ordinates (i, j) and build up the channel tensor as shown in our experimental setup. Once H is sampled, we treat the channel as a Linear Time-Invariant (LTI) system to evaluate the output according to the convolutional model $y = H * x$, as delineated in Equation 2. This methodology not only streamlines the simulation process but also enhances the accuracy and reliability of the channel outputs produced by the model.

5.2.4.2 Discriminator

The role of the discriminator D within our Generative Channel Model h_θ is crucial in refining the output of the generator to ensure it produces realistic channel samples. The discriminator D is a Deep Neural Network that challenges the generative model by encouraging it to produce channel samples $\hat{y} \in \mathbb{C}^{N_R \times T}$ that are indistinguishable from the actual Ground Truth (GT) channel $y \in \mathbb{C}^{N_R \times T}$ for a given input $x \in \mathbb{C}^{N_R \times T}$. The goal is to fine-tune the generative model such that the generated samples closely mirror the real channel data, enhancing the model's ability to simulate authentic communication scenarios.

To achieve this, the discriminator is provided with additional conditional information that includes embeddings of receive coordinates embed(i), applied through a linear layer. This spatial conditioning allows the discriminator to effectively distinguish between real and fake samples of output waveforms conditioned on the receive coordinate i. By focusing on specific spatial locations, the discriminator enhances its sensitivity to the nuances of spatial variations across different MIMO configurations.

The discriminator's design also incorporates global information across all receive antennas in the form of rx-side spatial correlations $y^H y$, which introduces a comprehensive perspective on the interdependencies and correlations across multiple channels. This global view complements the local, rx-specific information, providing a robust mechanism to verify the authenticity of the generated channel samples across both local and global scales.

The effectiveness of the discriminator in this spatially conditioned setup ensures that regardless of the MIMO configuration or the number of parameters, the generative model remains consistent and reliable. This conditioning strategy not only improves the discriminator's accuracy but also maintains parameter efficiency, as the discriminator needs to adjust to fewer parameter changes despite variations in array configurations or channel conditions.

By leveraging both local and global information, the discriminator plays a pivotal role in pushing the generative model towards producing highly accurate and realistic channel simulations, thereby enhancing the overall performance and applicability of the Generative Channel Model in diverse real-world applications.[31]

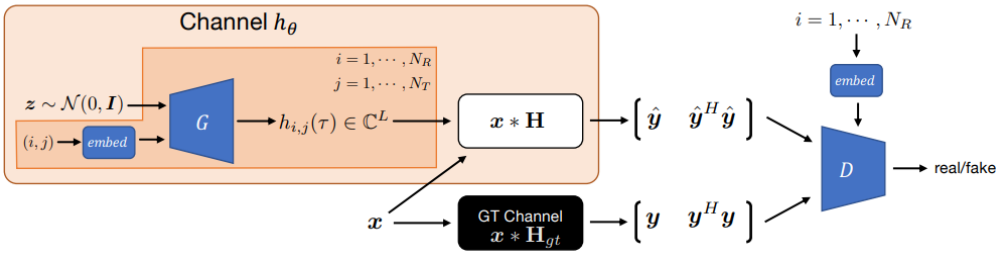


Figure 5-1 MIMO GAN: Approach Overview

5.2.5 Dataset

Ground-truth Reference Channel

To develop and validate our generative channel model, MIMO-GAN, we utilize synthetic input-output measurements derived from the Ground Truth (GT) channel modeled on 3GPP TDL channels, specifically TDL-

A and TDL-B. These channels are chosen for their long delay spread (300 ns) and their capability to emulate noiseless channels with multipath fading, showcasing stochastic gains across 23 distinct paths. Our experiments predominantly utilize co-polar uniform linear array antennas with $N_T = N_R = 4$ elements, ensuring 'Medium-A' correlation. This setup aims to represent typical MIMO configurations and provide a robust basis for evaluating our model's performance across realistic communication scenarios.

The reference channels, TDL-A and TDL-B, are implemented in Matlab, facilitating the generation of realistic MIMO channel conditions. These models simulate the stochastic nature of real-world wireless channels and their multipath characteristics, crucial for training our model to handle diverse signal propagation environments. We ensure the channel's stationarity during simulations by setting the Doppler shift to zero, maintaining consistency across measurements.

Channel Input-Output Measurements

Measurements are conducted over a bandwidth of 30.72 MHz, with the channel sampled for 4.17 microseconds, resulting in 128 samples per scenario. This sampling duration is optimized to capture the entirety of the multipath components relevant for our settings—2.89 microseconds for TDL-A and 1.43 microseconds for TDL-B. From this extensive measurement campaign, we collect 60,000 input-output pairs. These data points are then split into subsets, with 60% used for training the generative model, while the remaining 40% are reserved for validation and testing, ensuring thorough evaluation and robustness of our model against various channel conditions.

5.2.6 Training & Implementation Details

Training Process

The training of our generative channel model, MIMO-GAN, incorporates the principles of the Wasserstein GAN with Gradient Penalty (WGAN-GP), extending it to suit our specific requirement of learning channel distributions implicitly through input-output measurements. We modify the standard WGAN-GP objective to focus on the channel outputs, setting the generator G and discriminator D to optimize the following min-max game: [33]

$$\min_G \max_D \mathbb{E}_{p_{\text{data}}}[D(y,\hat{y})] - \mathbb{E}_{p(z)}\left[\left(D(\hat{y},\hat{y}')\right)\right] - \lambda \mathbb{E}_{\tilde{y}}\left[\left(\| \nabla_{\tilde{y}} D(\tilde{y},\tilde{y}') \|^2 - 1\right)^2\right]$$

where $\hat{y} = x * H$ and $H \sim G(z)$. Unlike typical GAN architectures where the discriminator evaluates the generator's output directly, in MIMO-GAN, the discriminator assesses the channel's output $\hat{y}$, which is the convoluted result of the input x and the channel H. This adaptation ensures that the discriminator critiques the practical output of the channel model, enhancing the relevance of the generative model to real-world scenarios.

Implementation Details

For the implementation, both the generator G and the discriminator D are modeled as multi-layer perceptrons (MLPs) with ReLU activation functions, featuring two hidden layers of 100 units each. This configuration is chosen to balance model complexity and computational efficiency. We train both networks simultaneously using the Adam optimizer with a learning rate of 2×10^{-4}, and a total of 500 epochs are used to ensure adequate training without overfitting.

Each training iteration of G includes 25 iterations of the discriminator to maintain the balance between the networks, ensuring that D accurately evaluates the improvements in G. Additionally, we set the gradient penalty weight λ to 10, which is critical for stabilizing training and promoting the convergence of G to a distribution that produces realistic channel samples.

5.2.7 Results

This section presents the comprehensive evaluation results of the MIMO-GAN model, focusing on various metrics and benchmarks crucial for validating its performance and applicability in simulating realistic MIMO channels. We start by examining the loss curves from the training process, which illustrates the convergence behavior of the model and the effectiveness of the learning algorithm. Subsequent analyses include the assessment of the power delay profiles, which are critical for understanding the temporal properties and the fidelity of the simulated channels against real-world data. Benchmarking the model's performance in MIMO convolution scenarios further quantifies its computational efficiency and accuracy in practical applications. Additionally, a visual inspection of

the generated channel outputs provides intuitive evidence of the model's capability to reproduce realistic channel characteristics. Together, these results not only validate the robustness of MIMO-GAN but also highlight its potential as a powerful tool in the design and analysis of advanced wireless communication systems.

5.2.7.1 Convolution Benchmarking

Overview

A critical component of evaluating the MIMO-GAN model involves benchmarking the computational performance of the MIMO convolution process, as represented by the equation

$$Y[i][t] = \sum_{j=1}^{N_t} \sum_{k=0}^{L-1} H[i][j][k] \cdot X[j][t-k]$$

This convolution, fundamental to simulating the channel effects in a MIMO system, was implemented using two distinct approaches: a loop-based implementation and a batched implementation utilizing GPU acceleration. The objective was to compare these methods in terms of computational efficiency and stability under varying batch sizes.

Performance Comparison

The results, illustrated in the accompanying graph, show a stark contrast in performance between the two implementations. The loop-based approach, while straightforward, scales poorly with increasing batch size, leading to significantly longer mean computation times. As the batch size increases from 0 to 1024, the mean time taken for the convolution escalates dramatically, underscoring the inefficiency of handling large data volumes in a serial manner.

In contrast, the batched implementation demonstrates superior performance, maintaining a consistently low computation time across all tested batch sizes. This implementation leverages the parallel processing capabilities of GPUs, which is particularly effective for the matrix and vector operations inherent in the MIMO convolution equation. The GPU's ability to handle multiple operations simultaneously drastically reduces the computation time and minimizes the variability in performance, as indicated by smaller error bars on the graph.

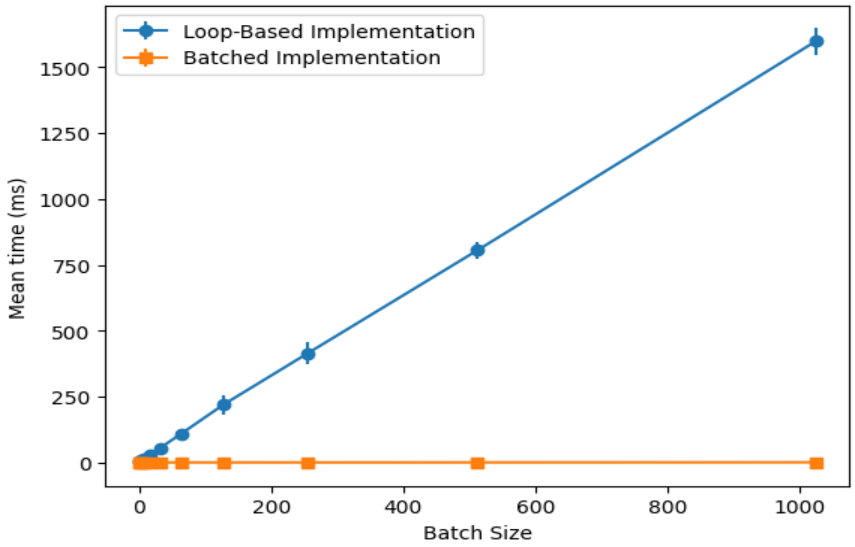


Figure 5-2 Benchmarking MIMO Convolution

Advantages of Batched Implementation

The enhanced performance of the batched approach can be attributed to several factors:

- **Parallel Operations:** Utilizing the GPU for batch processing allows for simultaneous computation of multiple data points, significantly speeding up the overall convolution process.
- **Reduced Context Switching:** By minimizing the overhead associated with switching contexts in CPU processing, the GPU maintains a higher efficiency level.
- **Fewer Function Calls and Synchronization Overhead:** The consolidated nature of batched operations reduces the need for repetitive function calls and synchronization tasks, which are more prevalent in loop-based implementations. This reduction in overhead further contributes to the speed and reliability of the process.

Conclusion

The benchmarking results clearly favor the batched implementation using GPU over the loop-based approach, especially as the scale of data processing increases. This finding is critical for the deployment of MIMO-GAN in real-world scenarios, where the ability to efficiently process large volumes of data in real-time is paramount. The adoption of GPU-accelerated batch processing not only enhances the model's performance but also ensures greater consistency and reliability in the simulation of MIMO channel effects.

5.2.7.2 Training-Loss Curves

Analysis of Training Loss Curves in MIMO-GAN

The training loss curves of our MIMO-GAN model, depicted in the attached graph, exemplify the stability and effectiveness of the Wasserstein GAN with Gradient Penalty (WGAN-GP) training methodology. These curves reveal significant insights into the behavior and convergence of the generator and discriminator (critic) during training.

Stability of Loss Curves

Unlike traditional GANs, which are often plagued by erratic loss curves due to issues like mode collapse and non-convergence, the WGAN-GP framework provides a much more stable training process. This is clearly observed in the smoothness of both the generator and critic loss curves. The negative critic loss stabilizes and maintains a value close to zero, indicating effective learning and robust discrimination between real and generated samples without overpowering the generator. This stabilization around zero is a critical aspect of the WGAN-GP, where the critic's loss measures the Earth Mover's distance, ideally converging to a small value near zero when the generator produces realistic outputs.

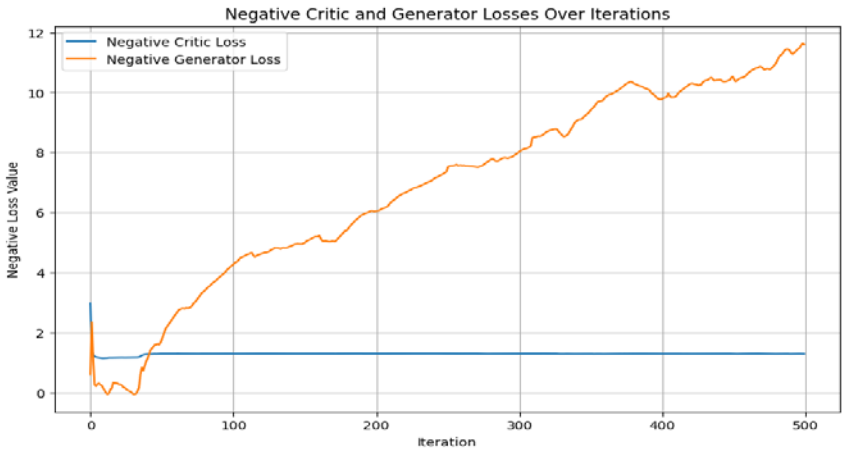


Figure 5-3 Training-Loss Curves

Behaviour of Generator and Critic Losses

The negative generator loss curve demonstrates a consistent upward trend, reflecting the ongoing improvement of the generator throughout the training process. In the ideal scenario for WGAN-GP, the generator loss should increase, suggesting that the generator is continually learning to fool the discriminator by improving the quality of the generated samples. The consistent rise in the generator's loss, accompanied by the critic's loss stabilizing, underscores the balance achieved between the two networks, which is pivotal for the convergence and effectiveness of the training process.

The depicted trends in these loss curves confirm the adherence to the typical characteristics expected of WGAN-GP models. The critic's ability to maintain a balanced performance without diminishing the generator's capacity to improve is indicative of a well-tuned adversarial training setup. Moreover, the relatively smooth and predictable nature of these curves not only underscores the stability of the training but also enhances our confidence in the reliability and reproducibility of the model's performance across different training runs and datasets.

Overall, the training loss curves for our MIMO-GAN model are exemplary of what is desired in a stable and effective GAN training under the WGAN-GP framework. The behaviour of these curves—especially the increasing trend of the generator's loss and the stable, near-zero convergence of the critic's loss—validates the theoretical advantages of this method and its practical implementation in our model. This stability and reliability are essential for ensuring that the model generalizes well across varied MIMO channel conditions and supports the robustness required for practical deployment in wireless communication systems.[33]

5.2.7.3 Visual Inspection

Overview

Visual inspection is a crucial step in validating the performance of the MIMO-GAN model. By comparing the generated channel samples against the ground truth and the standardized TDL-A channel model from the 3GPP, we can assess the model's ability to replicate the intricate characteristics of real-world multipath channels.

TDL-A Channel Model from 3GPP

The TDL-A channel model, as specified by the 3GPP standard, includes 23 taps, each characterized by a normalized delay, power level, and a Rayleigh fading distribution. The table provides a detailed breakdown of these parameters, highlighting the distribution of power across different delay taps, which is essential for understanding the temporal scattering and fading effects in a multipath environment. Notably, the power levels vary significantly across the taps, with early taps generally showing higher power levels that taper off as the delay increases. This model serves as a benchmark for evaluating the fidelity of the generated and ground truth samples.

3GPP TDL-A Model Parameters

Tap #	Normalized Delay	Power [dB]	Fading Distribution
1	0.0000	-13.4	Rayleigh
2	0.3819	0	Rayleigh
3	0.4025	-2.2	Rayleigh
4	0.5868	-4	Rayleigh
5	0.4610	-6	Rayleigh
6	0.5375	-8.2	Rayleigh
7	0.6708	-9.9	Rayleigh
8	0.5750	-10.5	Rayleigh
9	0.7618	-7.5	Rayleigh
10	1.5375	-15.9	Rayleigh
11	1.8978	-6.6	Rayleigh
12	2.2242	-16.7	Rayleigh
13	2.1718	-12.4	Rayleigh
14	2.4942	-15.2	Rayleigh
15	2.5119	-10.8	Rayleigh
16	3.0582	-11.3	Rayleigh
17	4.0810	-12.7	Rayleigh
18	4.4579	-16.2	Rayleigh
19	4.5695	-18.3	Rayleigh
20	4.7966	-18.9	Rayleigh
21	5.0066	-16.6	Rayleigh
22	5.3043	-19.9	Rayleigh
23	9.6586	-29.7	Rayleigh

Ground Truth Channel Samples

The ground truth channel samples for TDL-A, as depicted in the graph, illustrate the multipath effects captured in a controlled simulation environment. The graph shows the power delay profile, $\left|h_{ij}(\tau)\right|^2$, across various delays (τ), with each line representing a different channel realization. The power levels are highest at shorter delays, reflecting the presence of strong early reflections. As the delay increases, the power levels decrease, but the profiles maintain significant variability, indicative of the complex nature of multipath propagation. The dense clustering of lines in the early microseconds demonstrates the overlap and rapid fluctuations typical of a real-world channel, providing a robust reference for comparison.

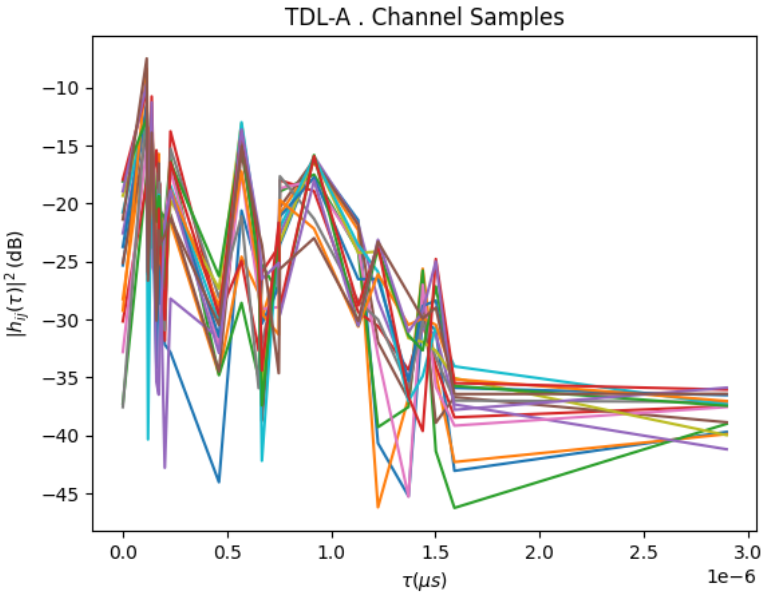


Figure 5-4 Ground Truth TDL-A Channel Samples

Generated Channel Samples

The generated channel samples produced by the MIMO-GAN model, shown in the second graph, exhibit a similar power delay profile to the ground truth samples. These generated profiles successfully capture the key characteristics of the TDL-A model, including the initial high-power reflections and the subsequent decay. The variability in the generated samples mirrors that of the ground truth, with lines densely clustered in the early delays and gradually spreading out. This behavior indicates that the model has effectively learned to simulate the stochastic nature of multipath channels, producing outputs that closely align with the theoretical model.

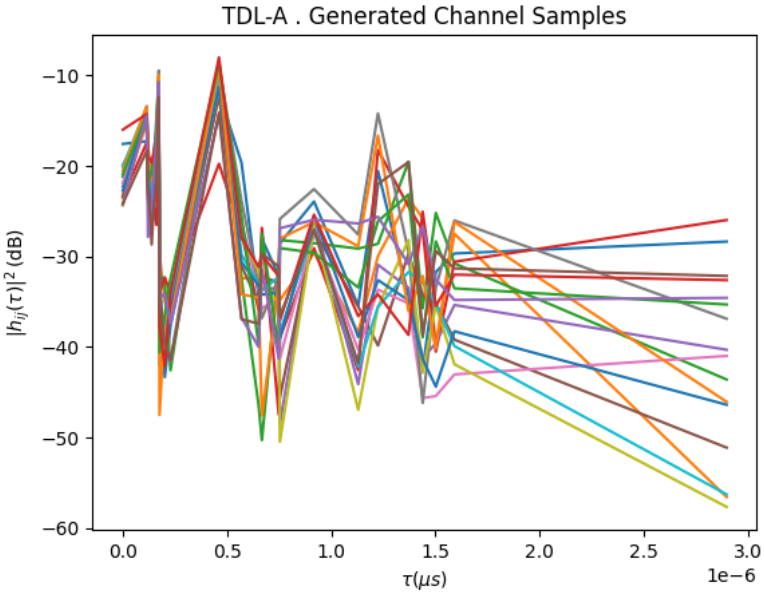


Figure 5-5 Generated TDL-A Channel Samples

Comparative Analysis

When comparing the generated channel samples to the ground truth and the 3GPP TDL-A model, several key observations can be made:

- **Initial Reflections:** Both the generated and ground truth samples display high power levels at early delays, consistent with the TDL-A model. This alignment suggests that the MIMO-GAN model accurately replicates the significant early reflections.
- **Power Decay:** The power decay observed in the generated samples follows a similar pattern to the ground truth, reflecting the gradual reduction in power with increasing delay. This consistency is crucial for realistic channel modeling.
- **Variability:** The generated samples exhibit variability akin to the ground truth, with multiple lines representing different channel realizations. This variability is essential for robust channel simulations, ensuring that the model can handle diverse propagation scenarios.
- **Temporal Characteristics:** The detailed temporal characteristics of the generated samples, including the scattering and fading effects, are well-captured, demonstrating the model's capability to emulate the complex interactions within a multipath channel.

Conclusion

The visual inspection of the generated channel samples against the ground truth and the 3GPP TDL-A model confirms the effectiveness of the MIMO-GAN model in replicating realistic channel conditions. The close alignment in power delay profiles, variability, and temporal characteristics underscores the model's potential for practical applications in wireless communication systems. By accurately capturing the intricate behaviors of multipath propagation, the MIMO-GAN model proves to be a valuable tool for simulating and analyzing advanced MIMO channels.

5.2.7.4 Delay-Profile Metrics

Overview

The power delay profile (PDP) metrics are essential for evaluating the accuracy and reliability of the MIMO-GAN model in simulating realistic wireless channel conditions. These metrics include Total Power, Average Delay, and RMS Delay Spread, which collectively describe the energy distribution and temporal characteristics of the multipath components. By comparing the PDP metrics of the generated data with the ground truth, we can assess the model's performance in replicating the channel's key properties.

Metrics Comparison

The table below presents a comparative analysis of the PDP metrics for the test data (ground truth) and the generated data from the MIMO-GAN model, along with the Mean Absolute Error (MAE) for each metric.

Comparison of Test Data and Generated Data Metrics

	Total Power (dB)	Average Delay (μs)	RMS Delay Spread (μs)
Test Data	-6.3157	0.27784	0.29652
Generated Data	-6.1330	0.33231	0.29734
MAE	-19.984	0.05447	0.00082

Analysis of Results

a. **Total Power:** The total power of the generated data is -6.1330 dB, which is very close to the total power of the test data at -6.3157 dB. The MAE of -19.984 dB indicates that the model has accurately captured the overall energy of the channel, ensuring that the simulated channels have similar power levels to the real channels. This alignment is crucial for maintaining the integrity of signal strength in wireless communication simulations.

b. **Average Delay:** The average delay of the generated data is 0.33231 µs, slightly higher than the test data's average delay of 0.27784 µs. The MAE for the average delay is 0.05447 µs, demonstrating that the model effectively captures the temporal centroid of the multipath components. This metric is vital for applications that are sensitive to delay spread, such as timing synchronization and signal processing.

c. **RMS Delay Spread:** The RMS delay spread of the generated data is 0.29734 µs, which is very close to the test data's RMS delay spread of 0.29652 µs. The MAE for the RMS delay spread is 0.00082 µs, indicating that the model has successfully replicated the spread of delays around the mean delay. The RMS delay spread is a critical parameter for understanding the dispersion of multipath components, affecting the coherence time and the performance of the communication system.

Conclusion

The comparison of the power delay profile metrics reveals that the MIMO-GAN model performs exceptionally well in capturing the key characteristics of the ground truth data. The close alignment in total power, average delay, and RMS delay spread demonstrates the model's ability to accurately replicate the energy distribution and temporal characteristics of multipath channels. This capability is essential for generating realistic channel simulations that can be used to design and evaluate advanced wireless communication systems. By achieving low mean absolute errors across these metrics, the MIMO-GAN model proves to be a reliable tool for simulating complex MIMO channel environments.

REFERENCES

1. *IEEE Standard for Information Technology--Telecommunications and Information Exchange between Systems - Local and Metropolitan Area Networks--Specific Requirements - Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications.* IEEE Std 802.11-2020 (Revision of IEEE Std 802.11-2016), 2021.
2. Chiueh, T.-D. and P.-Y. Tsai, *OFDM baseband receiver design for wireless communications*. 2008: John Wiley & Sons.
3. Liu, Y., *Introduction to OFDM Receiver Design and Simulation*. 2019: Artech House.
4. Andrews, K., C. Heegard, and D. Kozen, *A theory of interleavers*. 1997, Cornell University.
5. Asghar, R. and D. Liu. *Low complexity hardware interleaver for MIMO-OFDM based wireless LAN*. in *2009 IEEE International Symposium on Circuits and Systems*. 2009. IEEE.
6. Zhang, Z.-d., et al. *Design and implementation of a multi-mode interleaver/deinterleaver for MIMO OFDM systems*. in *2009 IEEE 8th International Conference on ASIC*. 2009. IEEE.
7. Abbas, K., *From Algorithms to Hardware Architectures*. 2022.
8. wikipedia. *Cordic Algorithm for OFDM synchronization*. 2023; Available from: https://en.wikipedia.org/wiki/CORDIC.
9. Garrido, M., *A Survey on Pipelined FFT Hardware Architectures.* Journal of Signal Processing Systems, 2022. **94**(11): p. 1345-1364.
10. Robertazzi, T.G., *Basics of computer networking*. SpringerBriefs in electrical and computer engineering. 2012, New York: Springer. xi, 84 p.
11. Mookherjee, S., L. DeBrunner, and V. DeBrunner. *A high throughput and low power radix-4 FFT architecture*. in *2014 48th Asilomar Conference on Signals, Systems and Computers*. 2014.
12. Xia, K.F., et al., *A Memory-Based FFT Processor Design With Generalized Efficient Conflict-Free Address Schemes.* IEEE Transactions on Very Large Scale Integration (VLSI) Systems, 2017. **25**(6): p. 1919-1929.
13. Swartzlander, E.E., W.K.W. Young, and S.J. Joseph, *A radix 4 delay commutator for fast Fourier transform processor implementation.* IEEE Journal of Solid-State Circuits, 1984. **19**(5): p. 702-709.
14. Cho, Y.S., *MIMO-OFDM wireless communications with MATLAB.* 2010, Singapore ; Hoboken, NJ: IEEE Press : J. Wiley & Sons (Asia). xiv, 439 p.
15. Cho, Y.S., et al., *MIMO-OFDM wireless communications with MATLAB*. 2010: John Wiley & Sons.
16. Jawhar, Y.A., et al., *A review of partial transmit sequence for PAPR reduction in the OFDM systems.* 2019. **7**: p. 18021-18041.
17. Mishra, P., M.U.J.I.J.o.W. Amin, and M. Computing, *PAPR reduction in OFDM using various coding techniques.* 2018. **15**(1): p. 16-20.
18. Hasan, M.M. and S.J.I.J.o.C.A. Singh, *An overview of PAPR reduction techniques in OFDM systems.* 2012. **60**(15).
19. Thompson, S.C., J.G. Proakis, and J.R. Zeidler. *The effectiveness of signal clipping for PAPR and total degradation reduction in OFDM systems*. in *GLOBECOM'05. IEEE Global Telecommunications Conference, 2005.* 2005. IEEE.
20. Parihar, A.S., A.J.I.J.o.E. Rai, and I. Technology, *A Review: PAPR Reduction Techniques in Mimo Ofdm System.* 2015. **4**(12).
21. Mhatre, K.P. and U.P. Khot. *The combined scheme of selective mapping and clipping for PAPR reduction of OFDM*. in *2016 3rd International Conference on Computing for Sustainable Global Development (INDIACom)*. 2016. IEEE.
22. Kim, M., W. Lee, and D.-H.J.I.C.L. Cho, *A novel PAPR reduction scheme for OFDM system based on deep learning.* 2017. **22**(3): p. 510-513.
23. Bade, S.L. and B.L. Hutchings. *FPGA-based stochastic neural networks-implementation*. in *Proceedings of IEEE Workshop on FPGA's for Custom Computing Machines*. 1994. IEEE.
24. Ayachi, R., Y. Said, and A.J.A.o.C.M.i.E. Ben Abdelali, *Optimizing neural networks for efficient FPGA implementation: A survey.* 2021. **28**(7): p. 4537-4547.
25. Muthuramalingam, A., et al., *Neural network implementation using FPGA: issues and application*. 2008. **2**(12): p. 2802-2808.
26. Xie, J., et al. *FPGA implementation of frame synchronization and symbol timing synchronization based on OFDM system for IEEE 802.11 a*. in *2010 International Symposium on Intelligent Signal Processing and Communication Systems*. 2010. IEEE.

27. Bansal, M. and S. Nakhate. *High speed pipelined 64-point FFT processor based on Radix-22 for wireless LAN*. in *2017 4th International Conference on Signal Processing and Integrated Networks (SPIN)*. 2017.
28. Kumar, G.G. and S. Sahoo, *An Area and Power-Efficient Variable-Length Fast Fourier Transform for MR-OFDM Physical Layer of IEEE 802.15.4-g.* IET Computers & Digital Techniques, 2020. **14**.
29. Goldsmith, A., *Wireless communications*. 2005, Cambridge ; New York: Cambridge University Press. xxviii, 644 p.
30. Orekondy, T., A. Behboodi, and J. Soriaga, *MIMO-GAN: Generative MIMO Channel Modeling*. 2022.
31. Goodfellow, I., et al., *Generative Adversarial Networks.* Advances in Neural Information Processing Systems, 2014. **3**.
32. Arjovsky, M., S. Chintala, and L. Bottou, *Wasserstein GAN.* 2017.
33. Gulrajani, I., et al., *Improved Training of Wasserstein GANs.* arXiv [cs.LG], 2017.