5-2012

Implementation of Orthogonal Frequency Division Multiplexing with FPGA

Qi Hao Yang
University of Arkansas, Fayetteville

IMPLEMENTATION OF ORTHOGONAL FREQUENCY DIVISION MULTIPLEXING
WITH FPGA

IMPLEMENTATION OF ORTHOGONAL FREQUENCY DIVISION MULTIPLEXING WITH FPGA

A thesis submitted in partial fulfillment
of the requirements for the degree of
Masters of Science in Electrical Engineering

By

Qi Yang
University of Arkansas
Bachelor of Science in Electrical Engineering, 2010

May 2012
University of Arkansas

ABSTRACT

In recent years, there have been dramatic shifts in cellular and telecommunication industries. As smartphones are dominating on the cellphone market, more and more people use these mobile devices to access internet either through third generation network or IEEE802.11 wireless local area network. Orthogonal frequency division multiplexing (OFDM) has been widely used in IEEE802.11 wireless local area network and fourth generation network. This paper will focus on the design and implementation of an Orthogonal Frequency Division Multiplexing system on field-programmable gate array (FPGA). The major components of an OFDM system include a modulator, an N-input inverse Fast Fourier Transform (IFFT), two root raised cosine filters (RRC filter), an N-input Fast Fourier Transform (FFT), and a demodulator. These components are designed by using very-high-speed integrated circuits (VHSIC) hardware description language (VHDL) under development environment of ModelSim, and then implemented onto Altera cyclone II EP2C35F672C6. The FFT accuracy is measured by comparing outputs from ModelSim to Matlab. The performance of the developed OFDM is evaluated in a wireless communication testbed.

This thesis is approved for recommendation
to the Graduate Council

Thesis Director:

____
Dr. Jingxian Wu

Thesis Committee:

____
Dr. Scott Christopher Smith

____
Dr. Randy Brown

ACKNOWLEDGEMENTS

I would like to acknowledge Dr. Jingxian Wu for his guidance and support throughout my Master’s program. I would also like to thank my parents and all my friends who gave me support and advice during my higher education.

Table of Contents

LIST OF TABLES

LIST OF FIGURES

Chapter 1. Introduction

1.1 Background

In recent years, there have been dramatic shifts in the cellular telecommunication industry. As smartphones have begun dominating the cellphone market, more and more people use these mobile devices to access the internet either through a third generation network or an IEEE802.11 wireless local area network. Mobile users are going to increase continuously and the speed of the network is very important to users. Because of these changes, the major wireless providers are switching third generation (3G) network to fourth generation (4G) network. The other reason that the providers are changing to 4G is that 4G offers internet speeds up to 10 times faster than 3G according to AT&T.

Orthogonal frequency division multiple access (OFDMA) is the standard used in IEEE802.11 wireless local area network and 4G network. It is a multi-user version of orthogonal frequency division multiplexing (OFDM). OFDM is a critical component in 4G network. Thus, to design and implement efficiency OFDM system is very important. Since the major component in an OFDM system is FFT, in order to have an efficient OFDM system, FFT unit needs to be efficient. Also, FFT unit is a critical component in other systems, such as digital signal processing (DSP) system. This thesis addresses an approach to design an OFDM system by using very-high-speed integrated circuits (VHSIC) hardware description language (VHDL). FFTs are designed and tested in Matlab first, then implemented on Field-Programmable Gate Arrays (FPGA) Cyclone II using VHDL. FFTs used in this research use decimation-in-time algorithm. The power consumption and complexity of FFTs are compared in the results. After completion of FFTs design, OFDM system is built upon FFTs. Implementation outputs of OFDM are compared with the simulation outputs.

1.2 Major modern wireless communication standards

1.2.1 1st generation: analog voice

First generation wireless communication networks were first introduced in the 1980s. It uses analog radio signals by modulating the voice signals to higher frequency directly, typically above 150MHz. The standard used in North America is Advanced Mobile Phone System (AMPS). It uses frequency division multiple access (FDMA) to divide available spectrums into different channels, and each user is assigned to a channel.

1.2.2 2nd generation: digital voice and narrowband data

Second generation wireless communication networks were commercially launched in 1991. Global system for mobile communication (GSM) is the most popular standard in 2G networks. It usually uses time division multiple access (TDMA) on top of FDMA. The principle of TDMA is to divide time into frames, and each frame is divided into slots [6]. Each user shares the same frequency channel that is assigned to its time slot. Three primary advantages that 2G has over 1G are digitally encoded voices, efficient spectrum usage which supports more users and data services which include text messages.

1.2.3 3rd generation: digital voice and broadband data

Third generation wireless communication networks were first commercially launched in 2001. It was relatively slow to get adopted by network providers since upgrading from 2G would require to build entirely new networks and replacing most broadcast towers. However, 3G has many advantages over 2G network. The advantages over 2G include much faster downlink speed, uplink speed, and higher security. cdma2000 is a family of 3G mobile network standards; it is primarily used in North America. It uses code division multiple access (CDMA) channel access to make communications between base station and mobile phones. In CDMA, a unique code that is known at both base station and mobile phones is assigned to each user. All users can transmit at the same frequency and time. The receiver picks up the desired signals by using a unique code.

1.2.4 4th generation: broadband data

Current 3G wireless communication network is migrating to fourth generation (4G) wireless communication network since more and more users are using smart phones and tablets to stream data from the network. International Mobile Telecommunications Advanced (IMT-Advanced) requirements for 4G standards are 1Gbit/s for peak download speed and 500Mbit/s for upload speed [2]. Current 3GPP Long Term Evolution (LTE) provide the highest LTE data rates of 300 Mbit/s [2], which is still slow compared to the standard . However, LTE Advanced is the standard that is targeted to surpass the IMT-Advanced standard requirements to be “real” 4G. 4G uses orthogonal frequency division multiple access (OFDMA) for its mobile internet access. OFDMA is a multi-user version of OFDM.

1.2.5 Orthogonal Frequency Division Multiplexing (OFDM)

OFDM is a frequency division multiplexing scheme that uses orthogonal subcarriers to carry data. It is the most popular Multi-Carrier Modulation (MCM). MCM achieves broadband communication by modulating a large number of narrow-band data streams over closely spaced sub-carriers. The transmitted data stream is divided into many sub-streams, and one sub-stream is transmitted in one sub-channel [6]. MCM is very effective to robust inter-symbol interference (ISI) when comparing to single-carrier broadband communication. The comparison between single-carrier and multi-carrier in time domain is shown in Figure 1. As shown in Figure 1, single-carrier broadband communication suffers serious ISI as multipath signals interfere with adjacent symbols due to short symbol period in one single carrier, however, MCM has less ISI due to its longer symbol period as the symbols are spread into different carriers, and the ISI in MCM can be completely removed by using guard interval between adjacent symbols.

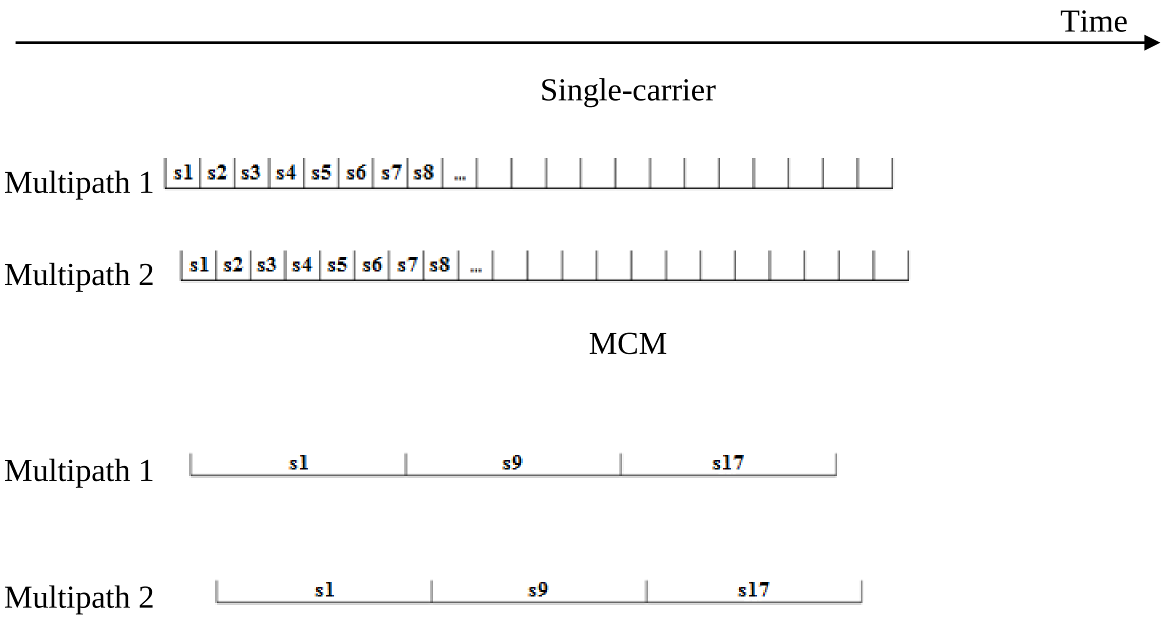


Figure 1: MCM vs single-carrier [6]

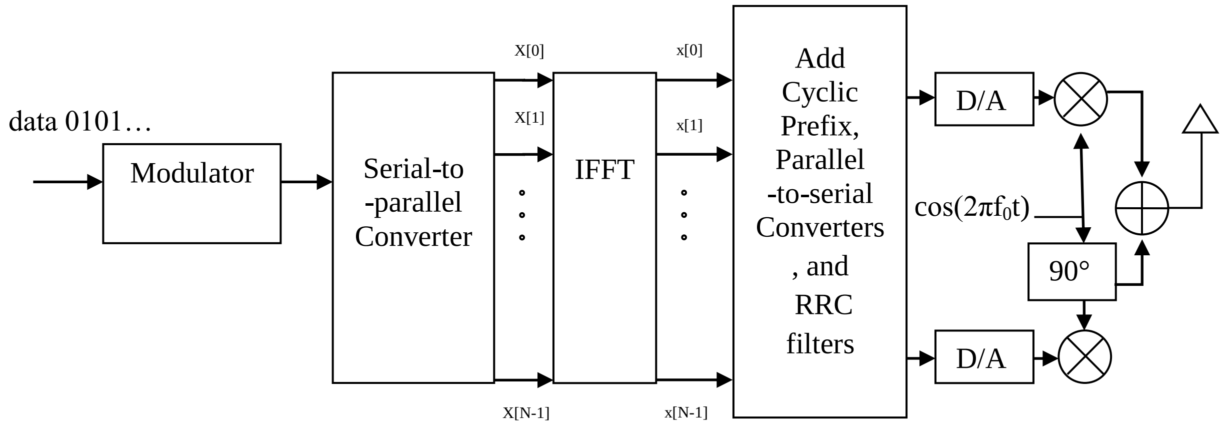
The other advantages of OFDM are no inter-carrier interference (ICI) among the subcarriers in quasi-static fading channel, where the channel is constant within one symbol period, high spectral efficiency as compared to conventional modulation schemes, and efficient implementation using Fast Fourier Transform (FFT).

1.3 Objectives

The primary goal of this thesis is to efficiently implement OFDM on FPGAs. The specific objectives leading to this goal are: understanding of the software development environments of Modelsim and Matlab, OFDM simulation on Matlab and Modelsim based on the theory of OFDM, and implementation of OFDM on FPGAs. The simulation result of OFDM's bit error rate (BER) performance should match analytical results. The output of OFDM on FPGA should be identical or close to the output of Matlab software environments.

Chapter 2. Orthogonal Frequency Division Multiplexing (OFDM)

2.1 Transmitter



data 0101…
Modulator
Serial-to -parallel Converter
X[0]
X[1]
X[N-1]
IFFT
x[0]
x[1]
x[N-1]
Add Cyclic Prefix, Parallel -to-serial Converters , and RRC filters
D/A
D/A
cos(2πf0t)
90°

Figure 2: Overall architecture of the OFDM transmitter

Figure 2 shows the overall architecture of the OFDM transmitter. The transmitter of OFDM is composed of a modulator, a serial-to-parallel converter, an N-point IFFT, two parallel-to-serial converters, and two root raised cosine filters. The data is first modulated by a modulator. The modulated data are transmitted through a serial-to-parallel converter, parallel frequency domain data, $\boldsymbol{X} = \left[X[0], X[1], X[2], \ldots, X[N-1]\right]$, are obtained. After performing IFFT on the parallel data, time domain data, $\boldsymbol{x} = \left[x[0], x[1], x[2], \ldots, x[N-1]\right]$, are obtained. The cyclic prefix is then added to the data, so prefixed data are $\tilde{x} = \left[x[N-u], \ldots, x[N-1], x[0], x[1], \ldots, x[N-1]\right]$, where u is the length of the wireless channel. The prefixed data are then passed through a parallel-to-serial converter, the real part and imaginary part of the data are both being filtered by a RRC filter and then a digital to analog converter; these analog signals are then combined with 90 degree phase difference. The result signals are then transmitted wirelessly through an antenna.

2.2 Receiver

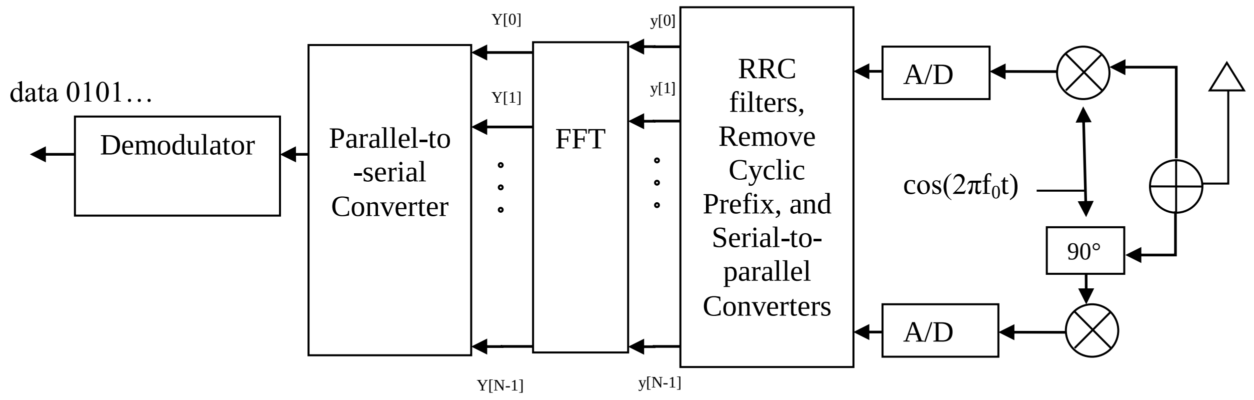

data 0101…
Demodulator
Parallel-to-serial Converter
Y[0]
Y[1]
Y[N-1]
FFT
y[0]
y[1]
y[N-1]
RRC filters, Remove Cyclic Prefix, and Serial-to-parallel Converters
A/D
A/D
cos(2πf0t)
90°

Figure 3: Overall architecture of the OFDM receiver

Figure 3 shows the overall architecture of the OFDM receiver. The receiver of OFDM composes a demodulator, a parallel-to-serial converter, an N-point FFT, two serial-to-parallel converters, and two root raised cosine filters. The received data is first separate into two parts: one is real number and the other is imaginary number. Each of these parts goes through an analog to digital converter and a RRC filter, and then the cyclic prefix is removed. Sampled complex data, $\boldsymbol{y} = [y[0], y[1], y[2], \dots, y[N-1]]$, from RRCs are then combined to form the parallel data. After performing FFT on these parallel data, frequency domain data, $\boldsymbol{Y} = [Y[0], Y[1], Y[2], \dots, Y[N-1]]$, are obtained. Once these frequency domain data goes through parallel to serial converter, a demodulator is used to demodulate the serial data and recover the original data.

2.3 Wireless Communication Background

2.3.1 Basic concepts

In wireless communication, information is transmitted through the propagation of unguided electromagnetic waves, which is called channel (or carrier). The information will be processed in the transmitter by modulation, and the coding technique to ensure the information would be robust to the noise during the wireless transmission. After receiver receives the information, it uses demodulation and decoding techniques to recover the original information. Figure 4 shows the major communication blocks.

Transmitter
Channel
Receiver

Figure 4: Major communication blocks

2.3.2 Noise

Noise is any unwanted electrical signals interfering with the desired signal. These unwanted electrical signals can come from natural or artificial sources, such as automobile ignition, thermal noise, and signals from other communication systems, etc. In this paper, only additive white Gaussian noise (AWGN) is considered during the simulation. AWGN comes from many natural sources, such as movements of electrons inside the conductor when the temperature is above 0K, radiation from the sun and other warm objects. AWGN follows Gaussian distribution with zero mean.

2.3.3 Fading

In this paper, frequency selective Rayleigh fading is considered as the only fading situation during the simulation. Frequency selective fading happens when multipath propagation caused different parts of transmitted symbols attenuated differently. Rayleigh fading is used to simulate for the situations, such as urban environments and tropospheric signal propagation. Rayleigh fading is based on Rayleigh distribution-summation of two zero-mean complex Gaussian distributed variables. Rayleigh fading function equals to

$$h(t) = h_I(t) + j * h_Q(t)$$

Where h_I is the real part, h_Q is the imaginary part, h_I and h_Q are both Gaussian distributed.

2.3.4 Modulation

Modulation is used to modulate signal from low frequency, which is not suitable for wireless communication, to high frequency signal to carry the information over wireless channel. The original signal is called baseband signal, $m(t)$, while the modulated signal is called bandpass signal, $s(t)$. The carrier signal, $c(t)$, is usually a high frequency sinusoid signal. Carrier signal has three parameters that can be modified to carry information. They are amplitude, frequency and phase as shown in the formula of carrier signal: $c(t) = A_c \sin(2\pi f_c t + \theta)$. Figure 5 shows general model of a modulation system.

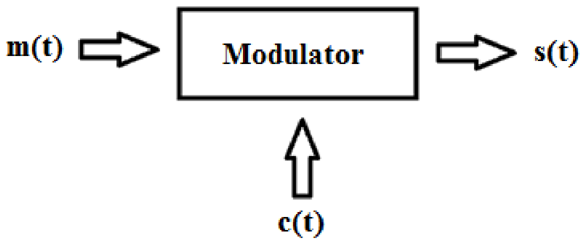
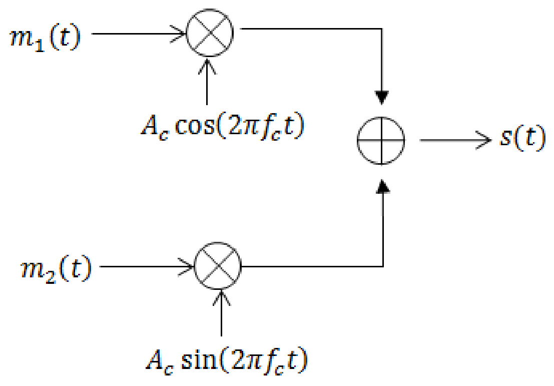


Figure 5: Modulation system [6]

Modulation can be decomposed into two steps: complex baseband modulation and frequency upconversion. Only baseband modulation is used in this paper. The passband modulation would not be performed in FPGA, since FPGA cannot perform frequency upconversion. However, it would be sufficient to examine baseband modulation only and the proof will be shown in the next paragraph. The modulation theme used in this paper is a baseband modulation based on quadrature phase-shift keying (QPSK). QPSK is a combination of two binary phase-shift keying (BPSK). It is called quadrature, because two carriers are mutually orthogonal. Figure 6 is the QPSK model with its mathematical representation.



$$s(t) = A_c m_1(t)\cos(2\pi f_c t) + A_c m_2(t) sin(2\pi f_c t)$$

Figure 6: QPSK model with its mathematical representation [6]

To show that baseband modulation would be sufficient to examine for the modulation, suppose band-pass signal, $s(t)$, is

$$s(t) = s_I(t)\cos(2\pi f_c t) - s_Q(t) sin(2\pi f_c t).$$

The complex base-band signal, $\hat{s}(t)$, is

$$\hat{s}(t) = s_I(t) + j s_Q(t).$$

Relationship between complex base-band signal and band-pass is

$$s(t) = Re\{\hat{s}(t)\exp(j2\pi f_c t)\}.$$

Figure 7 shows the decomposed modulation system.

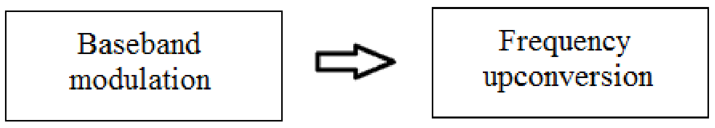
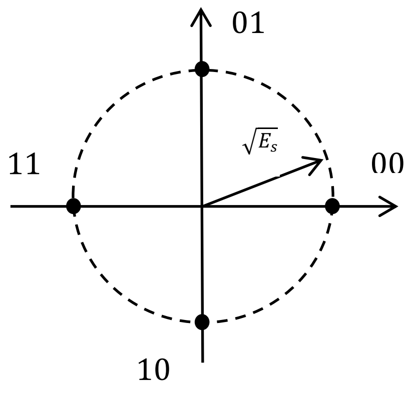


Figure 7: Decomposed modulation system

Baseband modulation signals are represented in complex numbers. For QPSK, two bits represent one symbol. All possible symbols based on two bits combination are shown in Figure 8.



$$\hat{s}(t)\epsilon \begin{Bmatrix} \sqrt{E_s}\exp\left(j\frac{0\pi}{4}\right), \sqrt{E_s}\exp\left(j\frac{2\pi}{4}\right) \\ \sqrt{E_s}\exp\left(j\frac{4\pi}{4}\right), \sqrt{E_s}\exp\left(j\frac{6\pi}{4}\right) \end{Bmatrix}$$

Figure 8: Symbol mapping based on bits combination on QPSK (where Es is the energy of one symbol)

2.3.5 Demodulation

Demodulation is used to find the original signals based on the received signals. Different modulation techniques have corresponding demodulations. The maximum likelihood decision rule determines that the modulated symbol with the shortest Euclidean distance to the received signal is the detected symbol at the receiver. The Euclidean distance between two symbols is defined as the length of distance between these two symbols. As shown in Figure 9, for QPSK demodulation, the received signal r will be demodulated as symbol A since its shortest Euclidean distance is to A. Let $\mathrm{r} = \mathrm{a} + \mathrm{jb}$ and $\mathrm{A} = 1$, Euclidean distance = $|\mathrm{a} + \mathrm{jb} - 1|^2$.

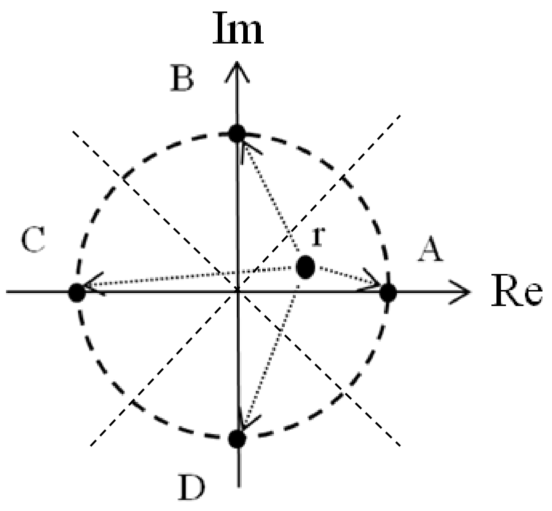
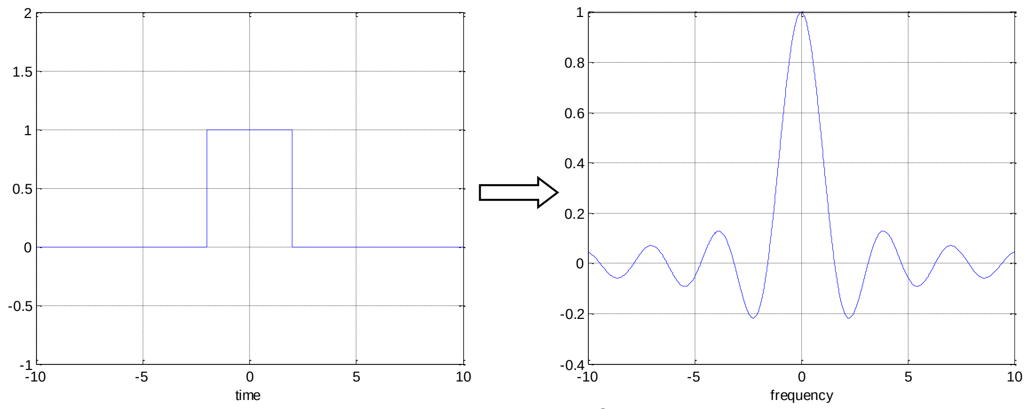


Figure 9: QPSK demodulation

2.3.6 Pulse shaping



Fourier Transform

Figure 10: Rectangular pulse has unlimited bandwidth

In wireless communication, rectangular pulse has unlimited bandwidth in frequency domain as shown in Figure 10. Unlimited bandwidth is a very bad property since frequency has been allocated into many different ranges for cell phone, Global Positioning System (GPS) navigation, Bluetooth, and so many more. In order to limit the bandwidth, pulse shaping is required. However, pulse shaping needs to satisfy Nyquist criterion, so that symbols would not interfere with each other. In Nyquist criterion, overall response of Tx filter and Rx filter needs to satisfy the property, such that signals do not interference with each other at $= 0, Ts, 2Ts\ \ldots nTs$.

Raised cosine (RC) pulse satisfies this property as shown in Figure 11.

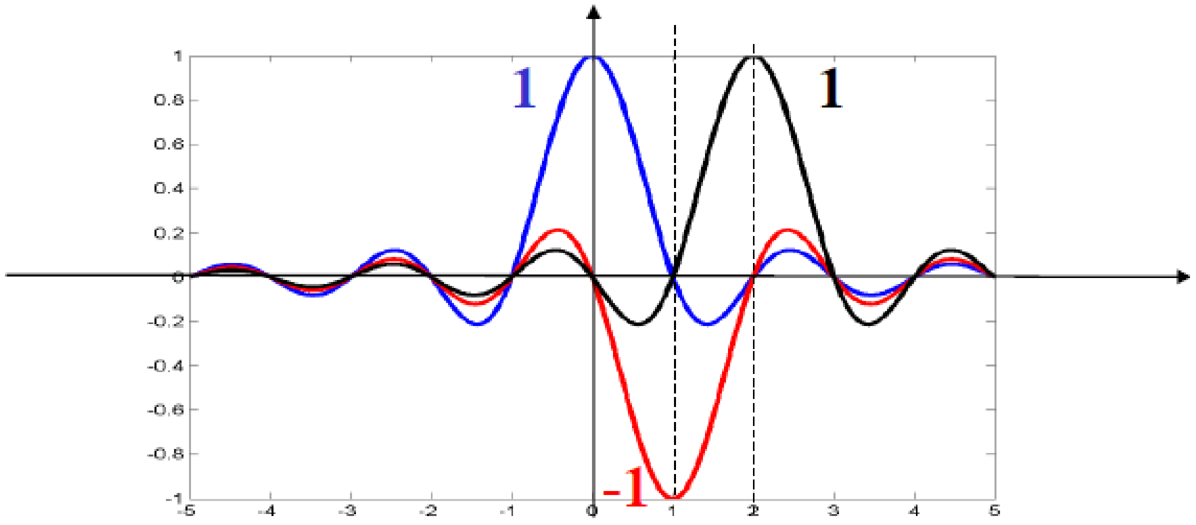


Figure 11: RC pulse symbols do not interference each other at nTs [6]

RC pulse also has limited bandwidth in frequency domain as shown in Figure 12.

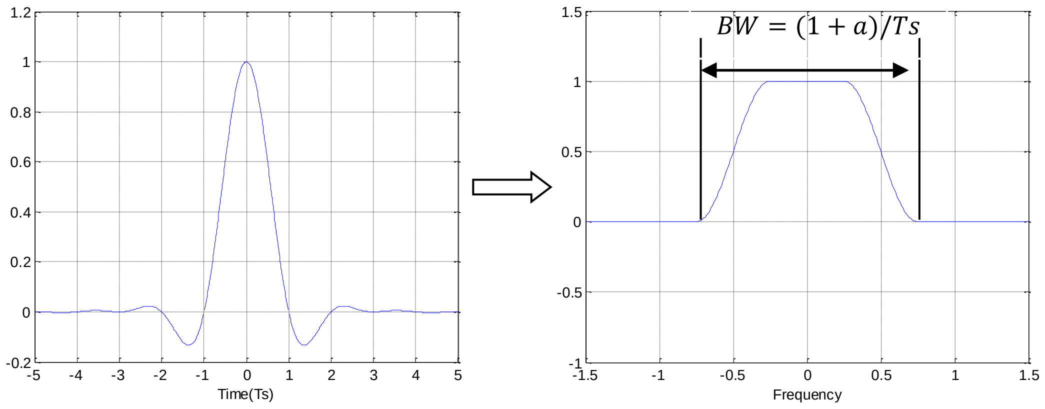


Figure 12: Time domain and frequency domain of RC pulse

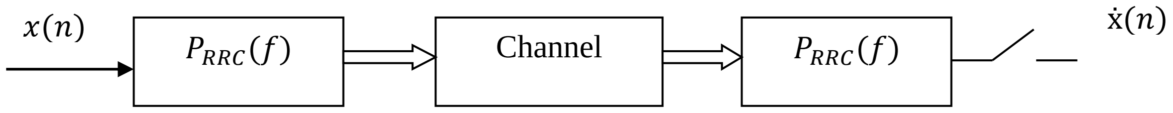
The function of raised cosine (RC) pulse $p(t)$ is

$$p(t) = \frac{\sin\left(\frac{\pi t}{Ts}\right)}{\frac{\pi}{Ts}} * \frac{\cos\left(\frac{\pi a t}{ts}\right)}{1 - \left(\frac{2at}{Ts}\right)^2},$$

where Ts is the symbol period, a is the roll off factor. Fourier transform of $p(t)$ is

$$P_{RC}(f) = \begin{cases} Ts & 0 \le |f| \le \frac{1-a}{2Ts} \\ \frac{Ts}{2}\left\{1 + cos\left[\frac{\pi Ts}{a}(|f| - \frac{1-a}{2Ts})\right]\right\} & \frac{1-a}{2Ts} \le |f| \le \frac{1+a}{2Ts} \\ 0 & |f| > \frac{1+a}{2Ts} \end{cases}$$

Since transmitter filter and receiver filter are used in the communication system, using two root raised cosine (RRC) filters for the pulse shaping results in an overall RC filter in the wireless communication system as shown in Figure 13.



$$P_{overall}(f) = \sqrt{P_{RC}(f)} * \sqrt{P_{RC}(f)} = P_{RC}(f)$$

Figure 13: Two RRC filters results an overall RC filter [6]

The frequency domain response of RRC filter is the square root of the RC filter, which is

$$P_{RRC}(f) = \sqrt{P_{RC}(f)} \text{ [6].}$$

The function of RRC is

$$p(t) = 4a\frac{\cos[(1+a)\pi t R_s] + \sin[(1-a)\pi t R_s]\,(4atR_s)^{-1}}{\pi\sqrt{T_s}[1 - 16a^2t^2{R_s}^2]} \text{ [6].}$$

2.3.7 Bit error rate (BER)

In wire wireless communications, a number of bit errors occur due to the noise, interference, and distortion during the transmission. BER equals to the total amount of bit errors divided by the total number of transmitted bits. It is one of the most important characteristics to measure the performance of wireless network.

2.4 Theory of OFDM

OFDM is the most popular MCM technique. Since the subcarriers are mutually orthogonal, there is no inter-carrier interference (ICI) among subcarriers. Suppose modulated data is

$$\boldsymbol{X} = \left[X[0], X[1], \dots, X[N-1]\right],$$

where $\boldsymbol{X}$ is in frequency domain. After IFFT, the time domain data are

$$x[n] = \frac{1}{\sqrt{N}} \sum_{i=0}^{N-1} X[i] e^{\frac{j2\pi ni}{N}}, where\ x = [x[0], x[1], \dots, x[N-1]].$$

Circular convolution in time domain is equivalent to multiplication in frequency domain [6]:

$$DFT\{y[n] = x[n] \otimes h[n]\} = X[i]H[i], \qquad i = 0,1,2, \dots, N-1,$$

where $H[i]$ is wireless channel between transmitter and receiver in frequency domain.

A practical system has the effect of linear convolution. In order to achieve a circular convolution, cyclic prefix is added to the data

$$\tilde{x} = \left[x[N-u], \dots, x[N-1], x[0], x[1], \dots, x[N-1]\right],$$

where u is the number of circular prefix, $\tilde{x}$ is cyclic-prefixed data in time domain. The cyclic-prefixed data is transmitted through channel. The channel functions like a linear convolution with data. Once the data is received, the cyclic prefix is removed, and then the circular convolution is achieved. Let $\boldsymbol{y} = [y[0], y[1], y[2], \dots, y[N-1]]$ to be the received data,

$$y[n] = \sum_{k=0}^{u} h[k]\,\tilde{x}\,[n-k] + z[n] = \sum_{k=0}^{u} h[k] x[n-k]_N + z[n]$$

$$= h[n] \otimes x[n] + z[n] \qquad n = 0, \dots, N-1$$

After FFT,

$$Y[i] = H[i]X[i] + Z[i] \qquad i = 0, \dots, N-1$$

Once the channel **H** is estimated by using known symbol patterns, effects of channel on received symbols can be removed. The results of demodulation are outputs of OFDM receiver.

2.5 Mathematic background

- Discrete Fourier transform (DFT)

$$X[k] = \frac{1}{\sqrt{N}}\sum_{n=0}^{N-1} x[n]W_N^{nk}\ \ k = 0,1,2\ \ldots.,N-1, W_N^{nk} = e^{-j2\pi nk/N}$$

- Inverse discrete Fourier transform (IDFT)

$$x[k] = \frac{1}{\sqrt{N}}\sum_{n=0}^{N-1} x[n]W_N^{-nk}\ k = 0,1,2\ \ldots.,N-1, W_N^{nk} = e^{-j2\pi nk/N}$$

- Linear convolution

$$y[n] = x[n] * h[n] = \sum\nolimits_{-\infty}^{\infty} h[k]x[n-k]$$

- Circular convolution

$$y[n] = x[n] \otimes h[n] = \sum\nolimits_{k=0}^{N-1} h[k]x[n-k]_N$$

$$[k]_N = k\ modulo\ N$$

- Linear convolution vs circular convolution

$$Let\ h = \{h[0], h[1], h[2], h[3]\} \qquad x = \{x[0], x[1], x[2], x[3]\}$$

$Linear\ convolution\ formula$: $y[n] = x[n] * h[n] = \sum\nolimits_{-\infty}^{\infty} h[k]x[n-k]$

An example of Linear convolution between two length-4 vectors **x**=[x[0], x[1], x[2], x[3]] and **h**=[h[0], h[1], h[2], h[3]] is:

$y[0] = h[0]x[0]$

$y[1] = h[0]x[1] + h[1]x[0]$

$y[2] = h[0]x[2] + h[1]x[1] + h[2]x[0]$

$y[3] = h[0]x[3] + h[1]x[2] + h[2]x[1] + h[3]x[0]$

$Circular\ convolution\ fomular: y[n] = x[n] \otimes h[n] = \sum_{k=0}^{N-1} h[k]x[n-k]_N$

An example of Circular convolution between **x** and **h** is:

$y[0] = h[0]x[0] + h[1]x[3] + h[2]x[2] + h[3]x[1]$

$y[1] = h[0]x[1] + h[1]x[0] + h[2]x[3] + h[3]x[2]$

$y[2] = h[0]x[2] + h[1]x[1] + h[2]x[0] + h[3]x[3]$

$y[3] = h[0]x[3] + h[1]x[2] + h[2]x[1] + h[3]x[0]$

Chapter 3. Hardware Implementation of OFDM

3.1 Introduction to Altera Cyclone II EP2C35F672C6 and VHDL

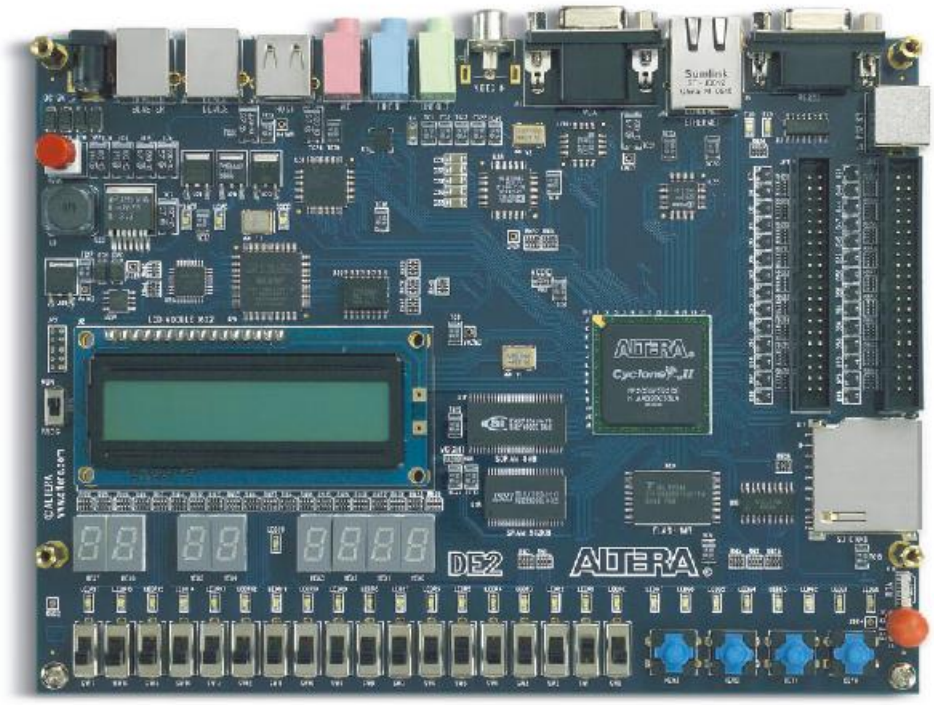


Figure 14: Altera Cyclone II EP2C35F672C6 [7]

Figure 14 is a picture of Altera Cyclone II EP2C35F672C6, it is a development and education board from Altera. It is mostly used for undergraduate courses and graduate projects. The board offers a rich set of features that make it suitable for this OFDM project. It features a state-of-the-art Cyclone II 2C35 FPGA in a 672-pin package [7]. All important components on the board are connected to pins of this chip, allowing users to control all aspects of the board's operation [7]. It has a built-in USB Blaster for FPGA configuration which eases the digital simulation process, two 40-pin expansion headers for external input/output, 16 x 2 LCD display, and more as shown in table 1. VHDL programming language is used during the implementation of OFDM. VHDL is used to write text models for logic circuits. Different logic circuits can be designed individually and then easily used as parts of design blocks after some tuning in parameters. Unlike C or Java, one advantage of VHDL is that it describes a concurrent system. The other advantage is that VHDL describes the behavior of a system that can be modeled and simulated in Modelsim development environment before translating the design into real hardware and wires in FPGA. In this way, the design cycle can be reduced through ModelSim simulations.

Table 1: Specifications for Altera Cyclone II EP2C35F672C6 [8]

Feature	**Description**
I/O Interfaces	• Built-in USB-Blaster for FPGA configuration • Line In/Out, Microphone In (24-bit Audio CODEC) • Video Out (VGA 10-bit DAC) • Video In (NTSC/PAL/Multi-format) • RS232 • Infrared port • PS/2 mouse or keyboard port • 10/100 Ethernet • USB 2.0 (type A and type B) • Expansion headers (two 40-pin headers)
Memory	• 8 MB SDRAM, 512 KB SRAM, 4 MB Flash • SD memory card slot
Displays	• Eight 7-segment displays • 16 x 2 LCD display
Switches and LEDs	• 18 toggle switches • 18 red LEDs • 9 green LEDs • Four debounced pushbutton switches
Clocks	• 50 MHz clock • 27 MHz clock • External SMA clock input

3.2 Q-15 number representation

In hardware implementation, different types of number representation need to be taken into consideration. The fixed point representation is implemented in this project since it has higher speed and lower cost when compared to the floating point representation. Although the floating point has higher dynamic range and no need for scaling, it is slower and much more complex. In order to design a fast and low cost OFDM system, the fixed point representation is used.

Q-15 number representation is used during implementation. In Q-15 format, 16 bit is used to represent a number. The decimal value of a 2's-complement number $B = b_{15}b_{14} \dots b_1 b_0, b_i \in \{0,1\}$, is given by

$$D(B) = -b_{15}2^{15} + b_{14}2^{14} + \cdots + b_1 2^1 + b_0 2^0 \text{ [9].}$$

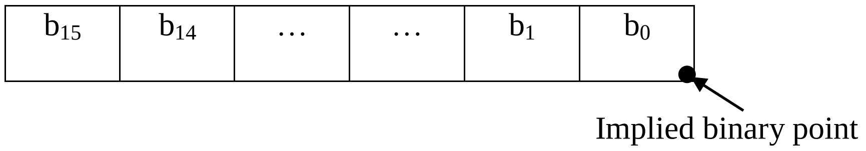


Figure 15: Integer representation [9]

As shown in Figure 15, 2's complement number representation is an integer representation. There are limitations on this representation. For example, it is not possible to represent a number that is larger than 2^{15}-1=32767 or smaller than -2^{15}=-32768. In order to overcome this limitation, numbers need to be normalized between -1 and 1. Hence, the fractional representation is used as shown in Figure 16.

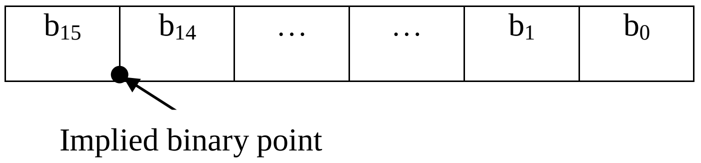


Figure 16: Fractional representation [9]

In Figure 16, the implied binary point is moved behind the most significant bit. The fractional value is given by

$$F(B) = -b_{15}2^0 + b_{14}2^{-1} + \cdots + b_1 2^{-14} + b_0 2^{-15} \text{ [9].}$$

For fractional representation, each number consists of 1 sign bit plus 15 fractional bits [9]. Since all numbers are normalized between 0 and 1, there is no overflow during multiplication. Overflow would never happen during addition of two numbers with non-identical signs, but could possibly occur with two numbers with identical signs. For example, if two numbers (a and b) have non-identical signs, let $1 > a > 0$ and $0 > b > -1$, $1 > a + b > -1$. During multiplication operation, a Q-30 format is obtained, where multiplication of two sign bits result in two sign bits. Bit-31 is the sign bit and bit-32 is the extended sign bit. Figure 17 shows multiplication between two Q-15 numbers A and B, which results in C.

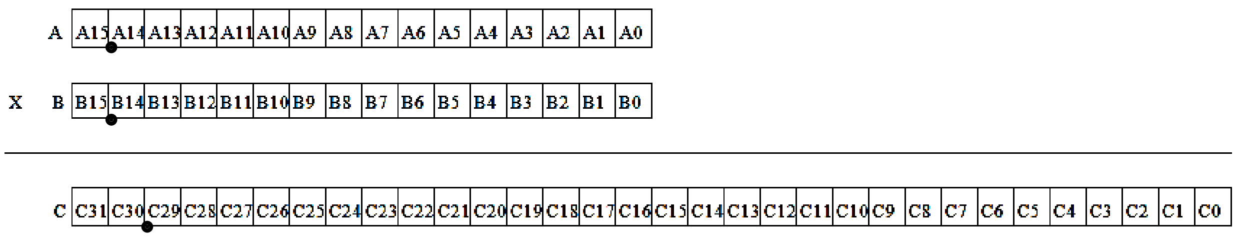

A
A15 A14 A13 A12 A11 A10 A9 A8 A7 A6 A5 A4 A3 A2 A1 A0
X
B
B15 B14 B13 B12 B11 B10 B9 B8 B7 B6 B5 B4 B3 B2 B1 B0
C
C31 C30 C29 C28 C27 C26 C25 C24 C23 C22 C21 C20 C19 C18 C17 C16 C15 C14 C13 C12 C11 C10 C9 C8 C7 C6 C5 C4 C3 C2 C1 C0

Figure 17: Multiplication of two Q-15 numbers

Since Q-15 format is used, only 16 bits can be stored from multiplication of two Q-15 numbers. By storing one sign bit and following 15 fractional bits, the product can be saved in Q-15 format with slight loss in precision as shown in Figure 18.

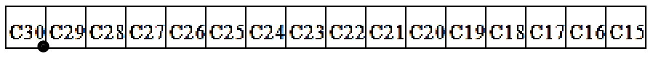

C30 C29 C28 C27 C26 C25 C24 C23 C22 C21 C20 C19 C18 C17 C16 C15

Figure 18: Stored product in Q-15 format

In the implementation, two Q-15 format numbers are used to represent a complex number. One Q-15 format number is to represent a real number, while the other one is to represent an imaginary number. The data format is shown in Figure 19 for a complex number.

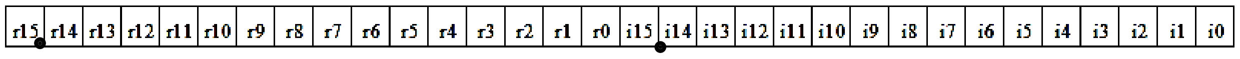

r15 r14 r13 r12 r11 r10 r9 r8 r7 r6 r5 r4 r3 r2 r1 r0 i15 i14 i13 i12 i11 i10 i9 i8 i7 i6 i5 i4 i3 i2 i1 i0

Figure 19: Complex number representation in fixed number

3.3 FFT implementation

3.3.1 Discrete Fourier Transform

Discrete Fourier Transform (DFT) is an important component in a wireless communication system, as it transforms the system from time domain to frequency domain. Thus, designing an efficient DFT is important for a wireless communication system since it lowers overall cost by reducing the total logic number and total power consumption.

The DFT of a finite-length sequence of length N is defined as

$$X[k] = \sum_{n=0}^{N-1} x[n] W_N^{kn}, \quad k = 0,1,\dots,N-1,$$

where $W_N = e^{-j(\frac{2\pi}{N})}$. The inverse discrete Fourier transform (IDFT) is defined as

$$x[n] = \frac{1}{N}\sum_{k=0}^{N-1} X[k] W_N^{-kn}, \quad k = 0,1,\dots,N-1,$$

where $X[k]$ and $x[n]$ are complex in both equations. Since the differences between DFT and IDFT are the sign of the exponent of W_N and scale factor $\frac{1}{N}$, it would be pretty straightforward to obtain IDFT from slight modification on DFT.

Complexity of logic circuit is one important factor that needs to be considered especially when doing large scale logic design. To evaluate the complexity of DFT with a finite-length sequence of length N with complex multiplications and complex additions,

$$\begin{aligned} DFT\ complexity &= N * (total\ summation\ and\ multiplication) \\ &= N * (N - 1 + N) \\ &= 2N^2 - N \\ &= O(N^2). \end{aligned}$$

IDFT has same complexity with DFT based on the equation. However, $O(N^2)$ is a bad result, since it produces large logic circuits when N is large. Every time N doubles, the total complexity of DFT or IDFT quadrupled. In order to make DFT or IDFT to be more practical in real design, FFT or IFFT is used as a substitution.

3.3.2 FFT

There are many different algorithms to achieve FFT, such as decimation in time, decimation in frequency, etc. In this research, decimation in time algorithm is used. FFT is the result after recursively decomposing the computation of DFT into smaller DFTs. The calculation of $X[k]$ can be alternatively represented as

$$X[k] = \sum_{n\ even} x[n]W_N^{kn} + \sum_{n\ odd} x[n]W_N^{kn}.$$

Supposing $n = 2r$ for even numbered n and $n = 2r + 1$ for odd numbered n, $X[k]$ can be expressed as

$$X[k] = \sum_{r=0}^{\frac{N}{2}-1} x[2r]W_N^{2rk} + \sum_{r=0}^{\frac{N}{2}-1} x[2r+1]W_N^{(2r+1)k}$$

$$= \sum_{r=0}^{\frac{N}{2}-1} x[2r](W_N^2)^{rk} + W_N^k \sum_{r=0}^{\frac{N}{2}-1} x[2r+1](W_N^2)^{rk}.$$

Since $W_N^2 = e^{-2j(\frac{2\pi}{N})} = e^{-j2\pi/(\frac{N}{2})} = W_{N/2}$, $X[k]$ can be expressed as

$$X[k] = \sum_{r=0}^{\frac{N}{2}-1} x[2r]W_{\frac{N}{2}}^{rk} + W_N^k \sum_{r=0}^{\frac{N}{2}-1} x[2r+1]W_{\frac{N}{2}}^{rk}$$

$$= G[k] + W_N^k H[k],$$

where $G[k]$ and $H[k]$ are both $(N/2)$-point DFT. $G[k]$ is the DFT of even numbered points of original sequence, and $H[k]$ is the DFT of odd numbered points of original sequence.

$G[k]$ and $H[k]$ are both periodic in k with period N/2, $G[0] = G[N/2]$ and $H[0] = H[N/2]$. Based on the derived equation, $X[0]$ equals to $G[0]$ plus result from $H[0]$ multiplying W_N^0. Similarly, $X[1]$, $X[2]$, ..., and $X[N/2-1]$ can be obtained using this method. Since

$$X[N/2] = G[N/2] + W_N^{N/2} H[N/2],$$

by substitution,

$$X[N/2] = G[0] + W_N^{N/2} H[0].$$

Using substitutions, $X\left[\frac{N}{2}+1\right]$, $X\left[\frac{N}{2}+2\right]$,..., and $X[N-1]$ can be obtained. Figure 19 shows the flow graph of an 8-point DFT after decomposing into two 4-point DFT computations, where even and odd inputs are grouped separately for N/2-point DFTs.

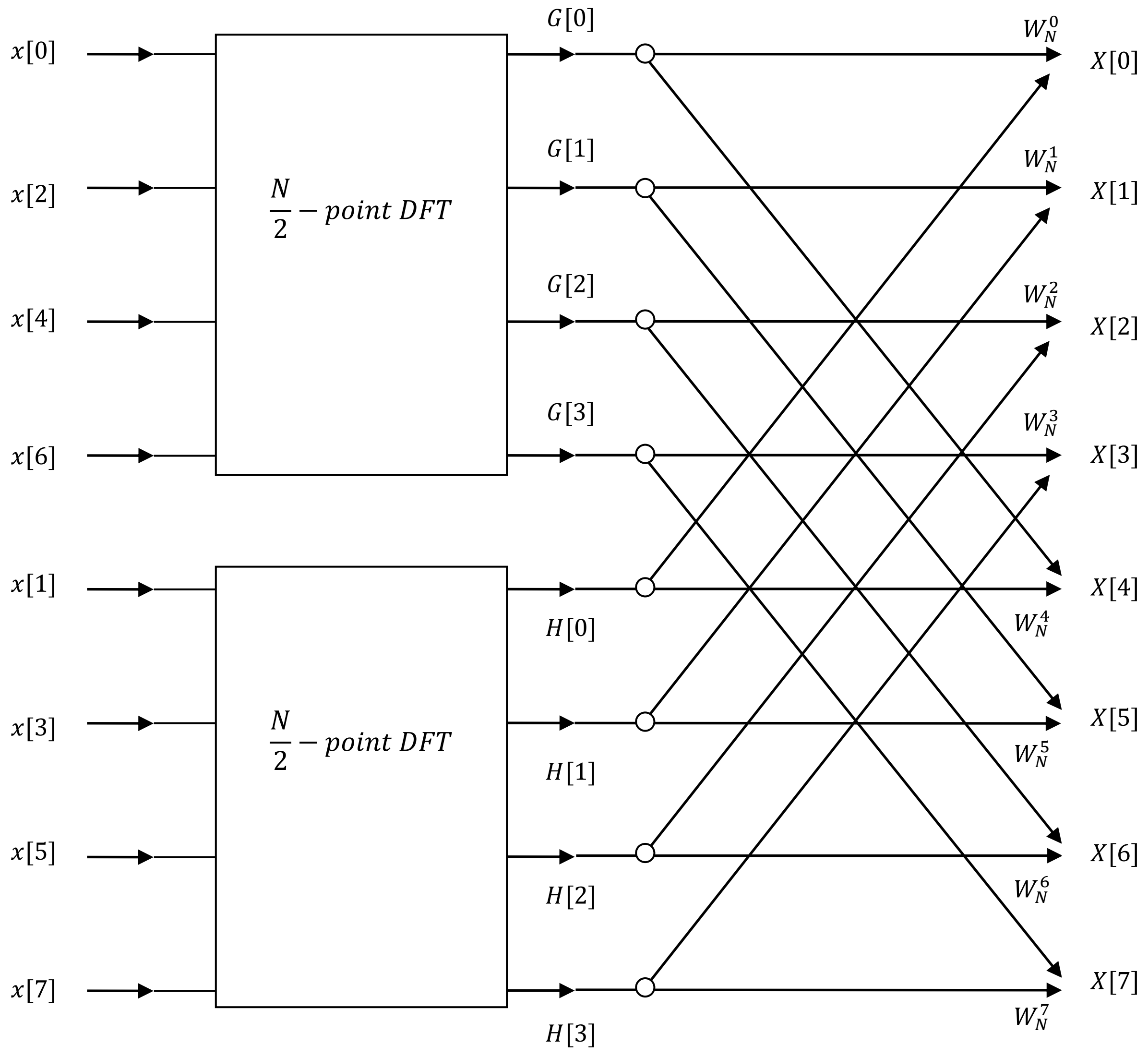


Figure 20: Flow graph of an 8-point DFT after decomposing into two 4-point DFTs computations [4]

DFT structure in Figure 20 needs to be decomposed further in order to achieve FFT. Using the same algorithm from decomposing N-point DFT to $N/2$-point DFTs, $N/2$-point DFT is decomposed into $N/4$-point DFTs:

$$G[k] = \sum_{l=0}^{N/4-1} g[2l]W_{N/4}^{lk} + W_{N/2}^{k} \sum_{l=0}^{N/4-1} g[2l+1]W_{N/4}^{lk},$$

$$H[k] = \sum_{l=0}^{N/4-1} h[2l]W_{N/4}^{lk} + W_{N/2}^{k} \sum_{l=0}^{N/4-1} h[2l+1]W_{N/4}^{lk}.$$

For example, the first $N/2$-point DFT in Figure 19 is composed into two $N/4$-point DFTs as shown in Figure 21. Even numbered input $X[0]$ and $X[4]$ are grouped as inputs of upper $N/4$-point DFT, and odd numbered input $X[2]$ and $X[6]$ are grouped as inputs of lower $N/4$-point DFT.

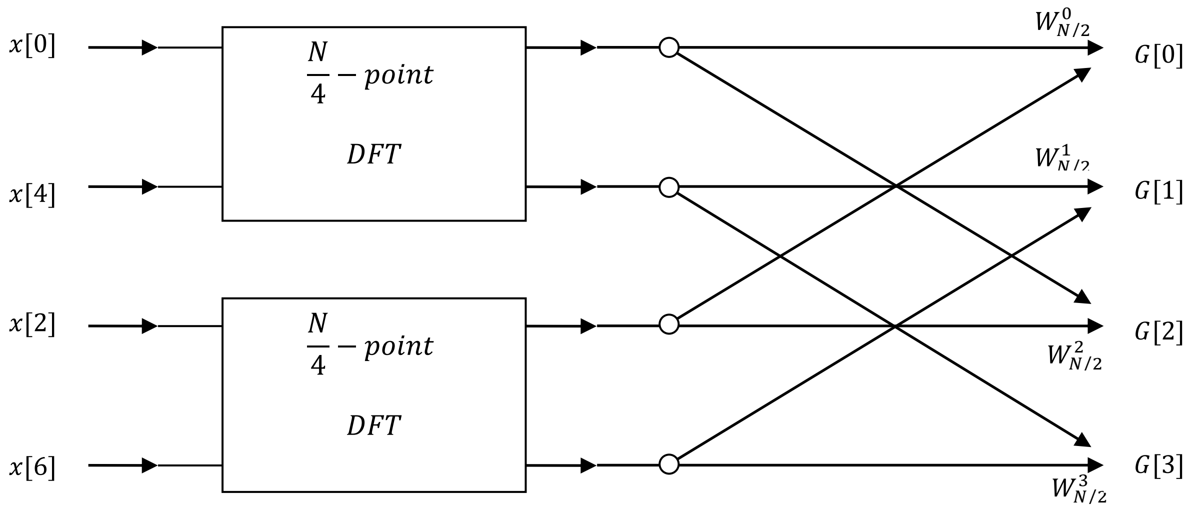


Figure 21: Flow graph of 4-point DFT after decomposing into two 2-point DFTs computation [4]

Since $W_{N/2}^{k} = e^{-j(\frac{2\pi k}{N/2})} = e^{-j(\frac{2\pi * 2k}{N})} = W_N^{2k}$, by substitute $W_{N/2}^{k}$ with W_N^{2k}, the flow graph of full 8-point DFT is then decomposed into 2-point DFTs as shown in Figure 22.

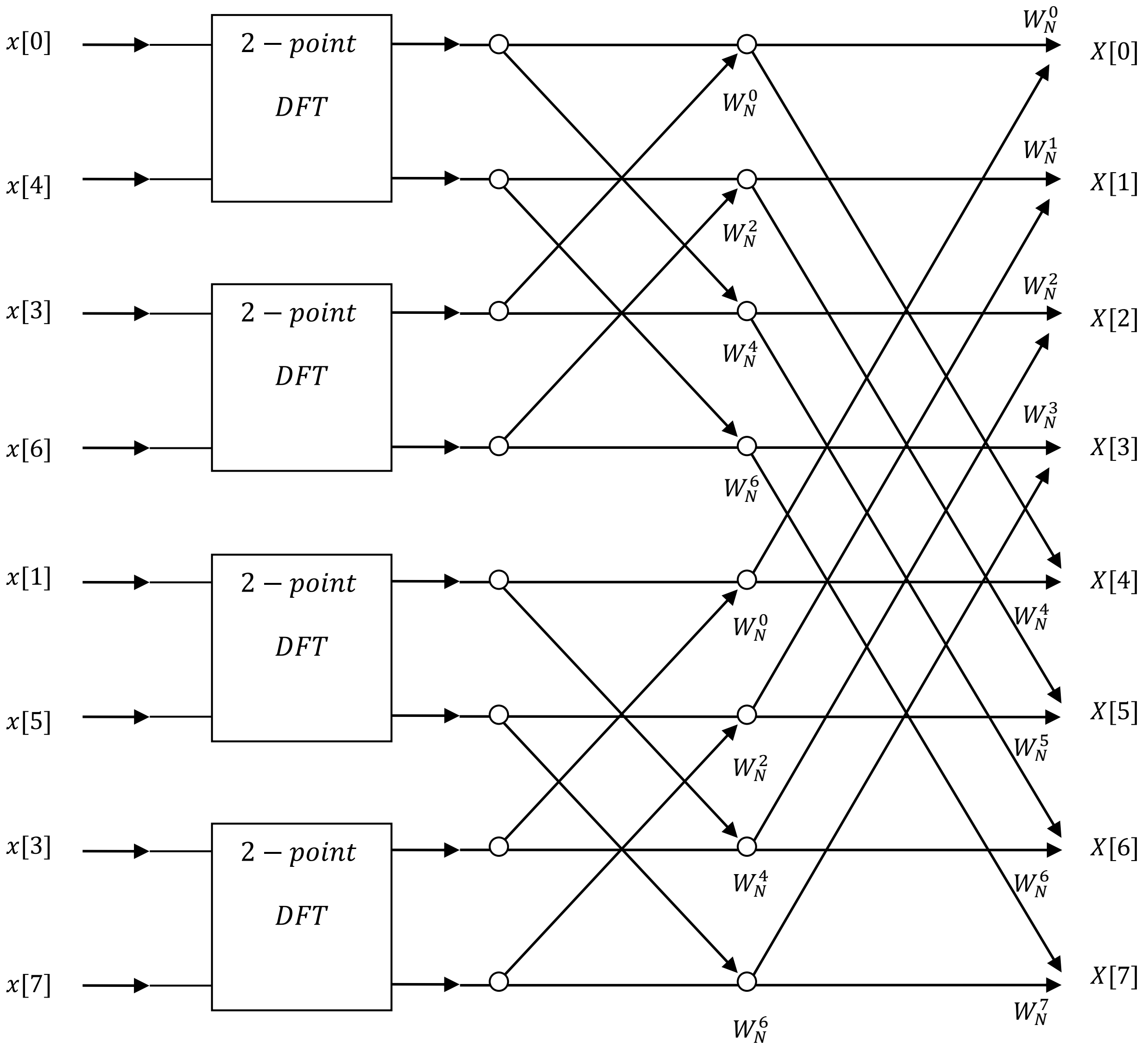


Figure 22: Flow graph of 8-point DFT after decomposing into two 2-point DFTs computation [4]

The first 2-point DFT in Figure 23 consists of $X[0]$ and $X[4]$ as the inputs. Figure 23 shows the flow graph for this DFT.

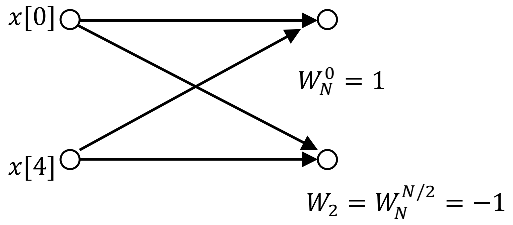


Figure 23: Flow graph of 2-point DFT [4]

By substituting the 2-point DFT flow graph in to Figure 22, we obtain the full flow graph for an 8-point FFT as shown in Figure 24.

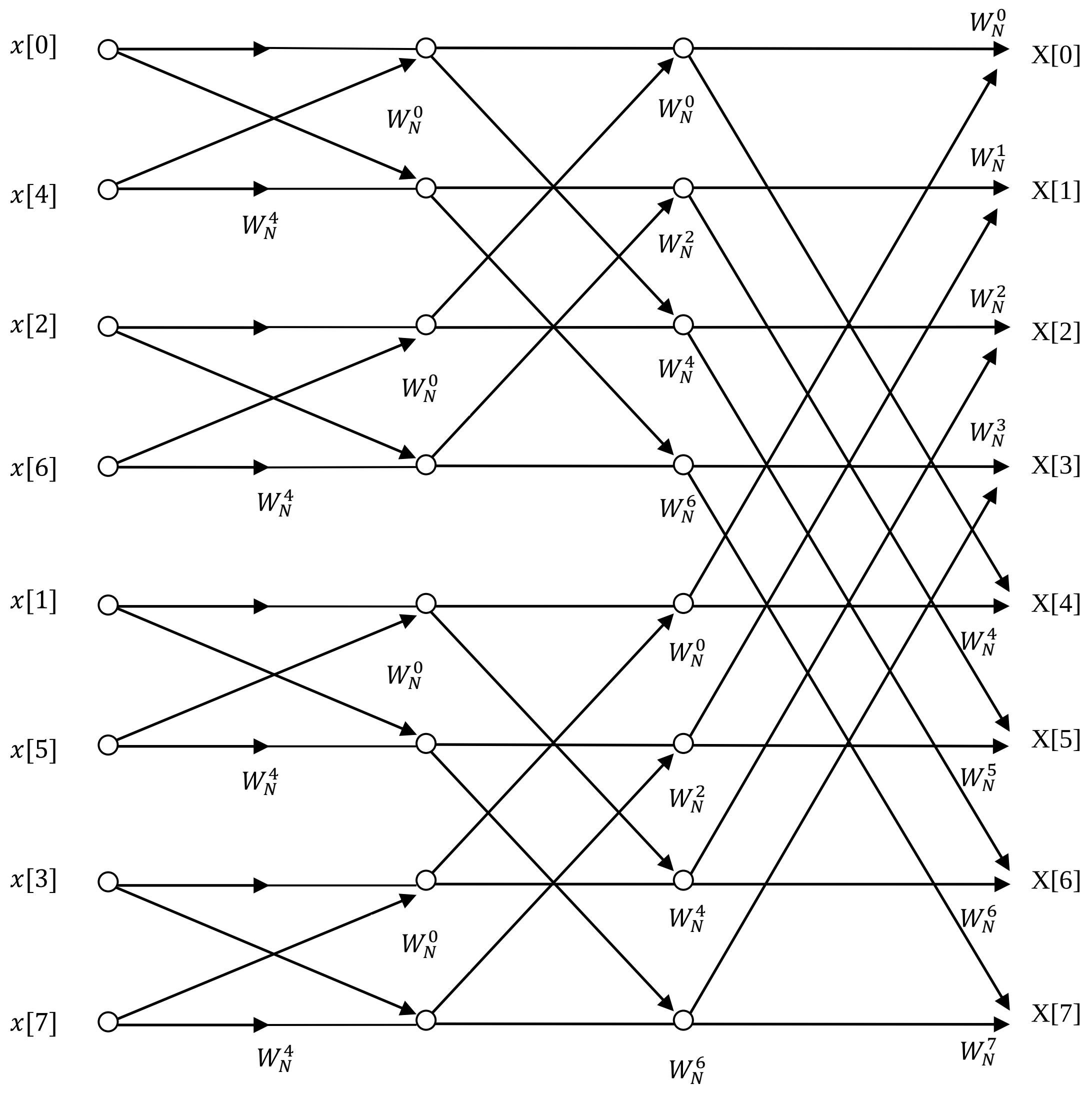


Figure 24: Flow graph for FFT [4]

The FFT in Figure 24 has three stages, which are calculated by $log2(N)$ with N equals 8. Each stage has N complex multiplications and $N - 1$ complex additions, so the total complexity of an N-point FFT is

$$FFT\ complexity = log_2(N) * (total\ summation\ and\ multiplication)$$

$$= log_2(N) * (N - 1 + N)$$

$$= 2N\ log_2(N) - log_2(N)$$

$$= O\big(N\ log_2(N)\big).$$

By comparing the complexity of FFT to that of DFT, it is obvious that FFT has better performance. For example, supposing input number $N = 2^{10} = 1024$, the complexity of DFT is $N^2 = 1024^2 = 1{,}048{,}576$, while the complexity of FFT is $log_2 N * N = 10{,}240$. By using FFT over DFT, the total amounts of computation logic units are largely reduced. Obviously, DFT is not suitable for hardware implementation, and FFT is a great method to get the same result as DFT while maintaining lower power consumption and lower logic units.

3.3.3 Butterfly operation

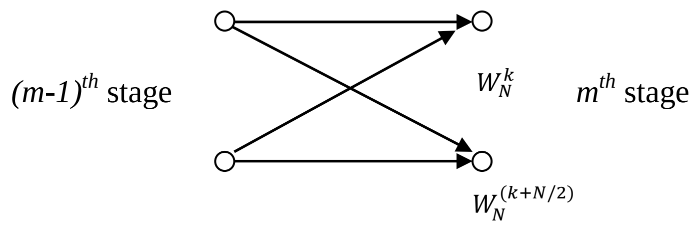


Figure 25: Butterfly operation [4]

The flow graph in Figure 25 can be further reduced based on the symmetry and periodicity of the coefficient W_N^k. By observing Figure 24, one can see that each stage of FFT can be decomposed into a structure called “butterfly”, based on the shape of graph as shown in Figure 25. Butterfly takes two outputs from the previous stage as inputs and calculates the inputs for the next stage. The coefficients on these butterflies have a pattern: the upper coefficient W_N^k times $W_N^{N/2}$ equals to lower coefficient $W_N^{(k+N/2)}$. Since

$$W_N^{N/2} = e^{-j\left(\frac{2\pi}{N}\right)*N/2} = e^{-j\pi} = -1,$$

the coefficient $W_N^{(k+N/2)}$can be rewritten as

$$W_N^{(k+N/2)} = W_N^{N/2} * W_N^k = -W_N^k.$$

The flow graph in Figure 24 can be improved further based on this property. Instead of using two complex multiplications in a butterfly, one complex multiplication would be sufficient to complete one butterfly calculation. In this way, it simplifies the calculation. The Butterfly unit is then simplified as shown in Figure 26.

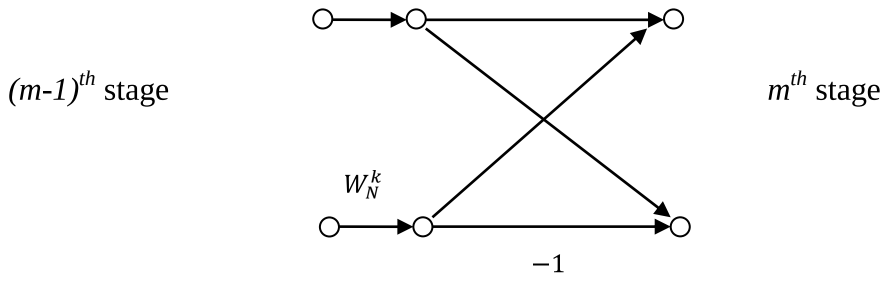


Figure 26: Simplified butterfly unit [4]

3.3.4 Revised FFT

By replacing the butterfly units in Figure 24 with simplified butterfly units, the FFT unit is simplified as shown in Figure 27. On the simplified FFT, there are only $N/2$ coefficients used for complex multiplications. Each stage has $N/2$ complex multiplication deductions by using simplified butterflies.

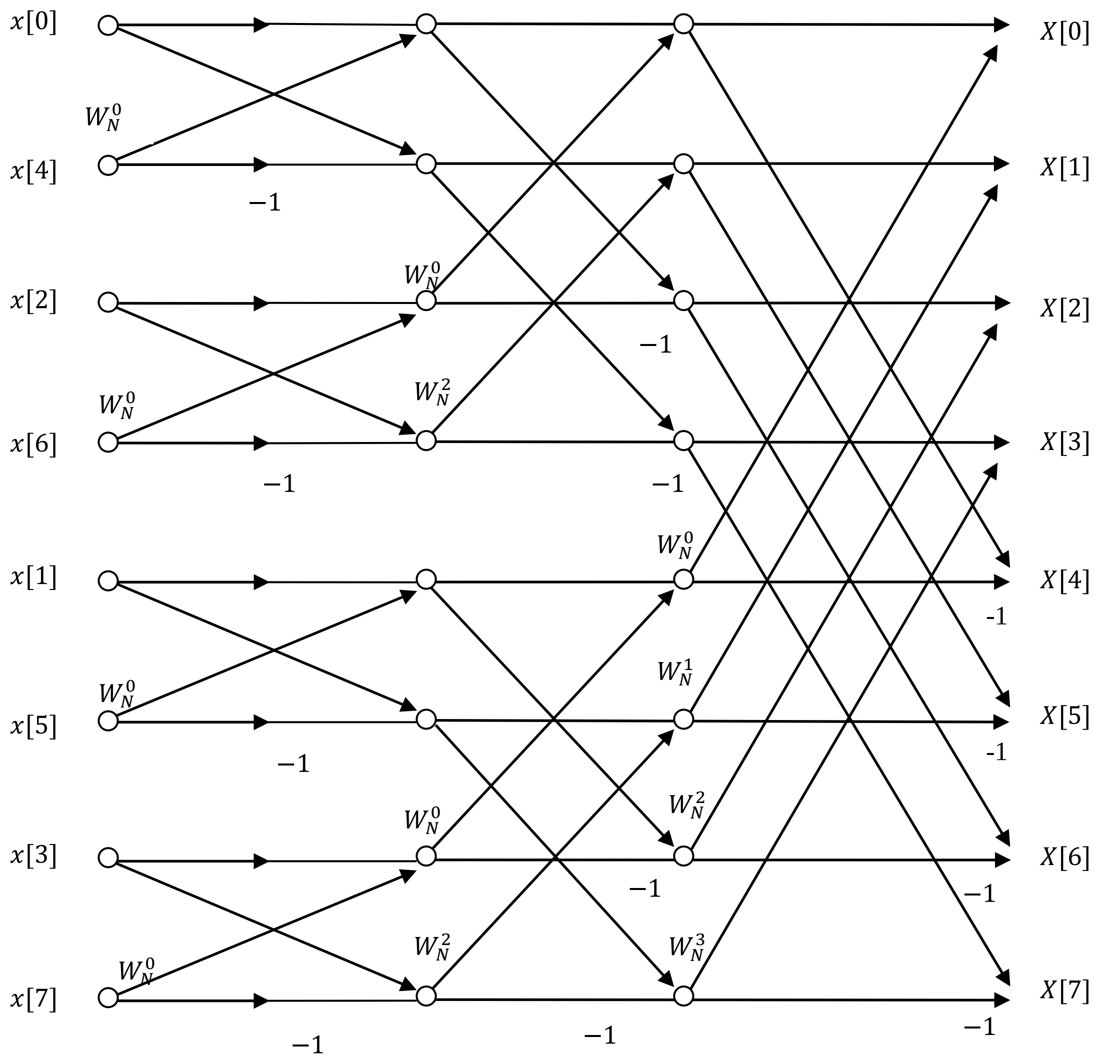


Figure 27: Simplified FFT [4]

As can be seen on different stages of the simplified FFT, the inputs for each butterfly need to be rearranged, so that the FFT will perform the function correctly. For example, on the first stage of 8 input FFT, the inputs are rearranged as 0, 4, 2, 6, 2, 5, 3, 7; on the second stage of 8 input FFT, the inputs are rearranged as 0, 2, 1, 3, 4, 6, 5, 7; on the third stage inputs are rearranged as 0, 4, 1, 5, 2, 6, 3, 7. These input rearrangements will require the Read-Only Memory (ROM) to store the location information during hardware implementation since recursively using butterflies is recommended in order to reduce logic units. Supposing butterflies are not recursively used, let the FFT input number equal 2^{10}; there are 10 stages to complete the FFT. 512 butterflies are used in each stage; total amount of butterflies would be 512*10=5120, which is a huge number. By recursively using butterflies, the logic units would be greatly reduced with the help of ROM. Although using ROM can greatly reduce the logic units, it can be removed by using a Singleton flow graph as shown in Figure 28.

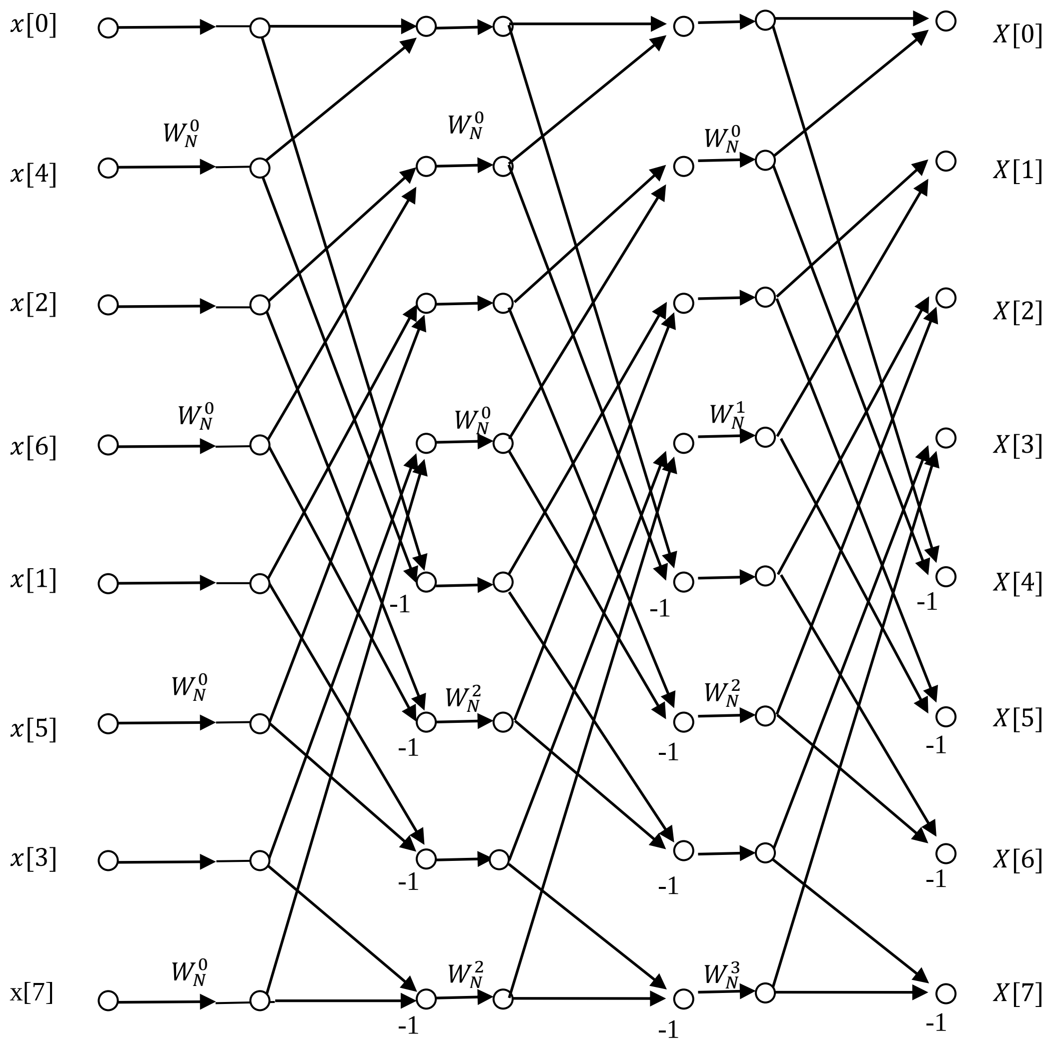


Figure 28: Flow graph of FFT originally given by Singleton [4]

A rearrangement of flow graph shown in Figure 27 results in the Singleton FFT flow graph in Figure 28. In the Singleton FFT flow graph, each stage has an identical geometric structure. By using the Singleton FFT flow graph, no ROM is required. This greatly reduces the design complexity and logic units.

Noted that inputs of FFT are in bit-reverse order, let the digital number be **a**=$a_0a_1a_2a_3\ldots a_n$, and its bit reverse order be $a_n\ldots a_3a_2a_1a_0$. Suppose $N = 8$, table 2 shows the bit-reversed order of each digital number.

Table 2: Bit-reversed table

Digital number	Bit-reversed order
000 (0)	000 (0)
001 (1)	100 (4)
010 (2)	010 (2)
011 (3)	110 (6)
100 (4)	001 (1)
101 (5)	101 (5)
110 (6)	011 (3)
111 (7)	111 (7)

By denoting the m stage input array as $Xm[l]$, where $l = 0, 1, 2, 3 \ldots\ N - 1$, the first stage input array of the 8-input FFT is acquired, shown in Figure 29, after making the arrangement on the original input array.

$$X_0[0] = x[0]$$

$$X_0[1] = x[4]$$

$$X_0[2] = x[2]$$

$$X_0[3] = x[6]$$

$$X_0[4] = x[1]$$

$$X_0[5] = x[5]$$

$$X_0[6] = x[3]$$

$$X_0[7] = x[7]$$

Figure 29: Rearrangement on first stage input array

Since the Singleton design approach is implemented, the butterfly is revised. The revised butterfly has different output locations when compared to the original one. The original butterfly's input and output relationship is shown in Figure 30, and the Singleton butterfly's input and output relationship is shown in Figure 31, where X_{m-1} represents a previous state and X_m represents the next state. Figure 32 shows the revised butterfly architecture in the Quartus II RTL viewer. Detailed architecture of revised butterfly will be introduced in next section.

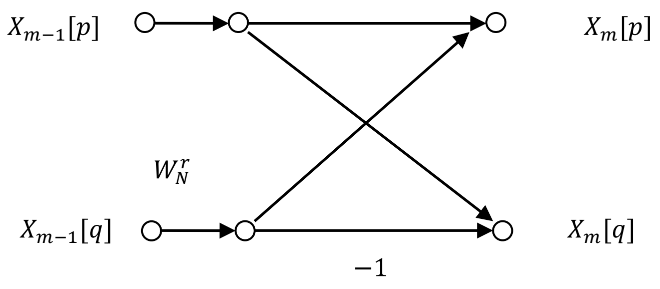


Figure 30: Original butterfly [4]

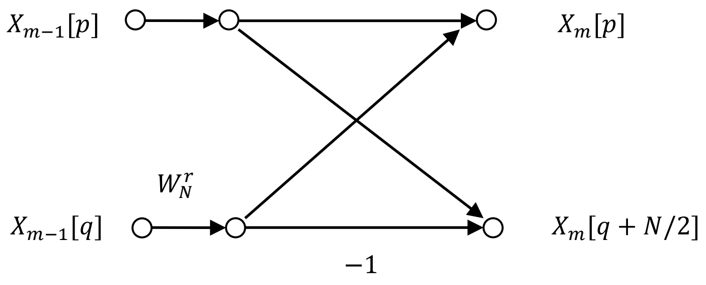


Figure 31: Revised Butterfly [4]

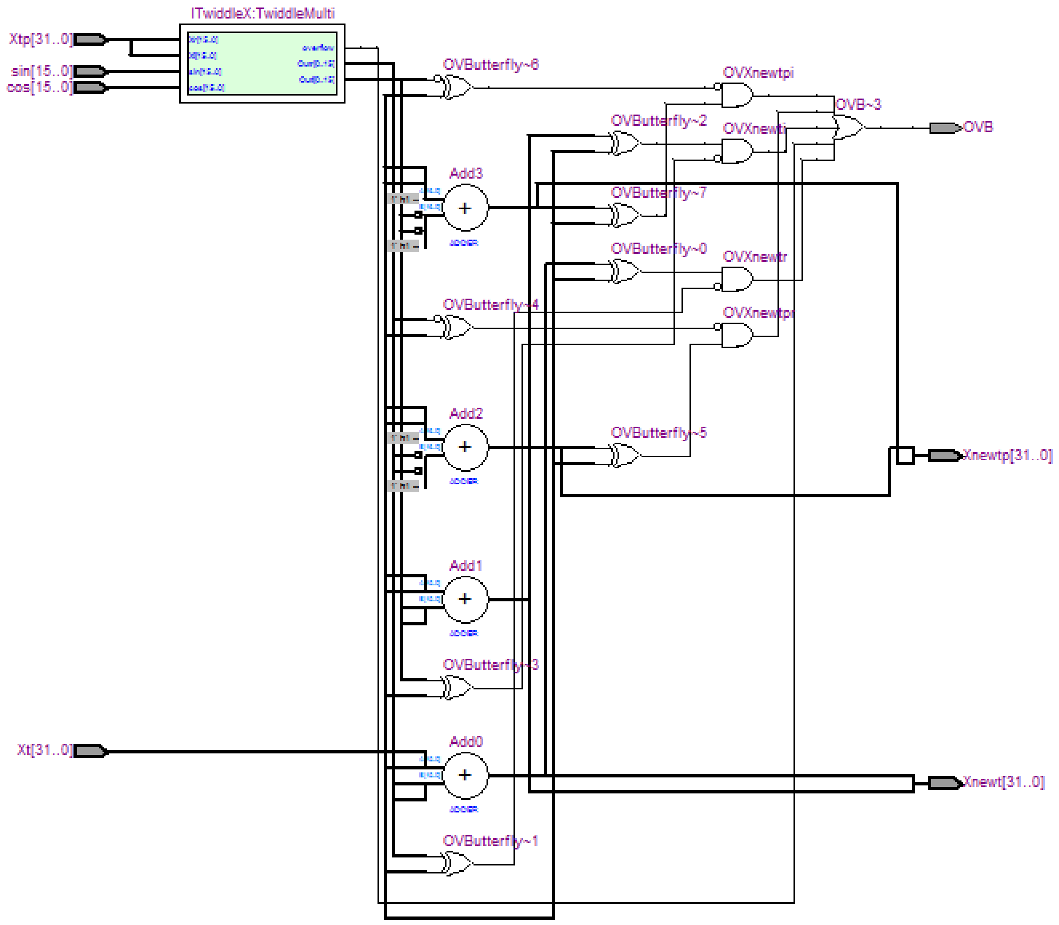


Figure 32: Butterfly architecture in the Quartus II RTL viewer

In Figure 32, twiddelX calculates the product of the second input of butterfly (Xtp) and the twiddle factor, which is a complex number composed of real numbers and imaginary numbers (sin and cos). This product is added to the first input of butterfly (Xt) to get the output. Overflows are checked for the complex additions.

3.3.5 Overall FFT architecture

By observing the data flow in Figure 28, it is obvious that an 8-input FFT requires three stages to complete. Each stage contains four butterflies. The larger the FFT, the more stages the data needs to go through before getting the final output. It is also true for number of butterflies in each stage. The relationship between the N-input FFT and total stages m is $m = log_2N$. The total butterflies used in one stage are $N/2$. For a 64 input FFT, 6 stages are required and each stage would need 32 butterflies. The overall N-input FFT architecture is shown in Figure 33.

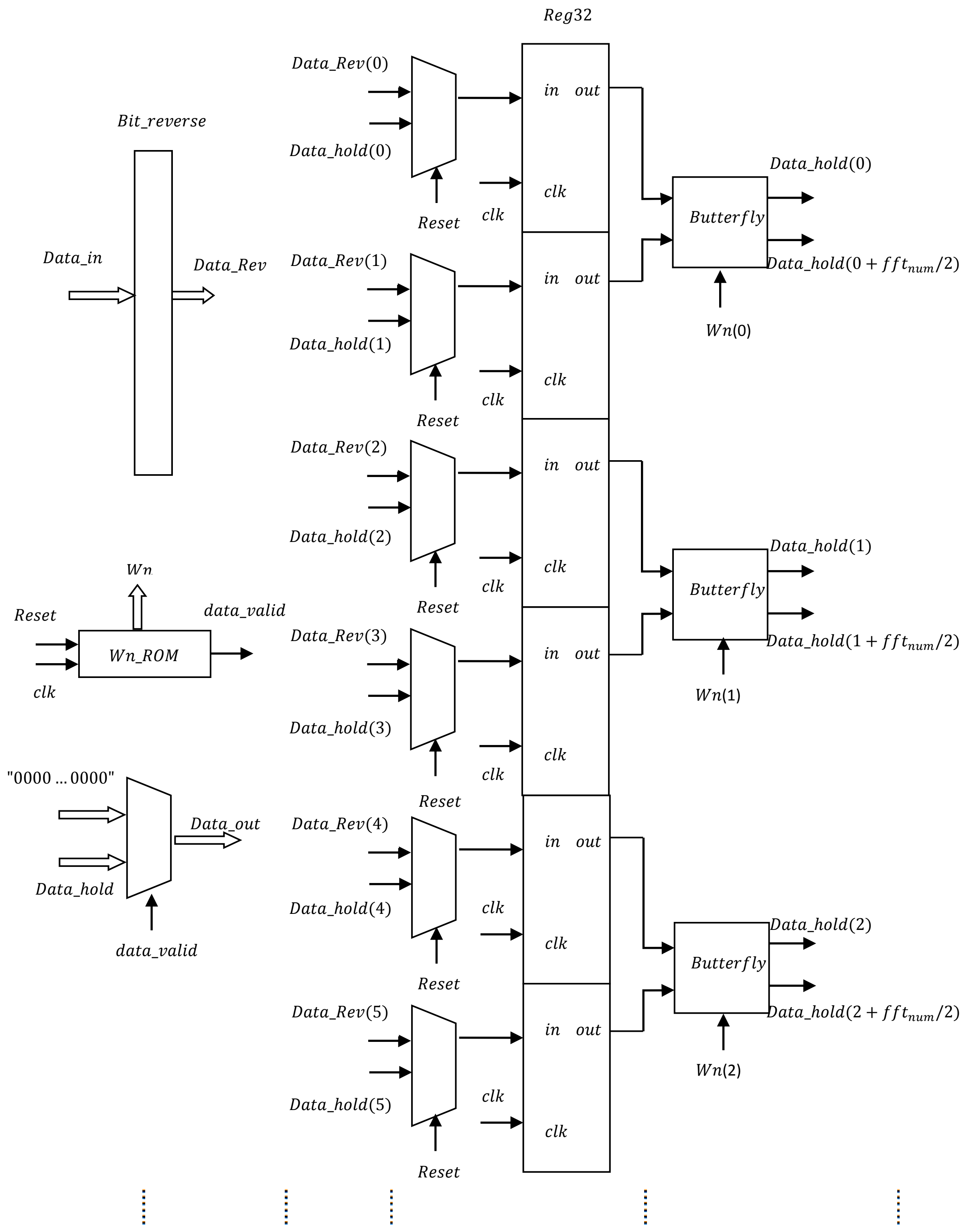


Figure 33: Overall FFT architecture

In Figure 33, the data first go through a Bit_reverse unit that rearranges the input data as $Data\_Rev$. Each $Data\_Rev$ data is then passed as an input of 2-input multiplex, whose output depends on state of $Reset$. The other input of the multiplex is $Data\_hold$, which is the output of the butterfly unit. The $Reset$ is '0' before FFT function is turned on; each $Data\_Rev$ data is the output of multiplex at this moment. The output of each multiplex is then passed to the 32-bit register ($Reg32$). These registers store the values for the butterflies' inputs. The butterfly has three inputs, two from the registers and one from the Wn_Rom, which stores the Wn values for each stage. The Wn_Rom has an internal up-counter based on the input clock and the reset. The Wn_Rom outputs the first stage Wn when the $Reset$ is '0'. Once the $Reset$ is turned on to be '1', the internal counter starts counting and the Wn output will be based on the current state of the counter which indicates the current stage of FFT. The internal architecture of the butterfly would be introduced in next section. For the i^{th} butterfly, the data inputs are from the $(2i)^{th}$ and $(2i+1)^{th}$ registers, where data outputs are arranged as the $(i)^{th}$ and $(i+fft\_num/2)^{th}$ of the $Data\_hold$. $Data\_hold$ is routed back to the second inputs of the multiplexes. Once the $Reset$ is turned on, the $Data\_hold$ would be the outputs of the multiplexes, and used as second stage inputs for the butterflies to produce new processed data. This new processed data is then routed back to the multiplexes again. This recursion keeps continuing until the counter in the Wn_Rom indicates that the FFT output is valid. The Wn_Rom sends a $data\_valid$ signal to indicate the time to send $Data\_hold$ to $Data\_out$, which is the result of the FFT.

Twiddle factor Wn of Wn_Rom is different from stage to stage. For example, let FFT input number equals to 8. The first stage 8-input FFT has W_N^0 on all four butterflies as shown in Figure 34, where $x1$ is the input, $x2$ is the second stage input.

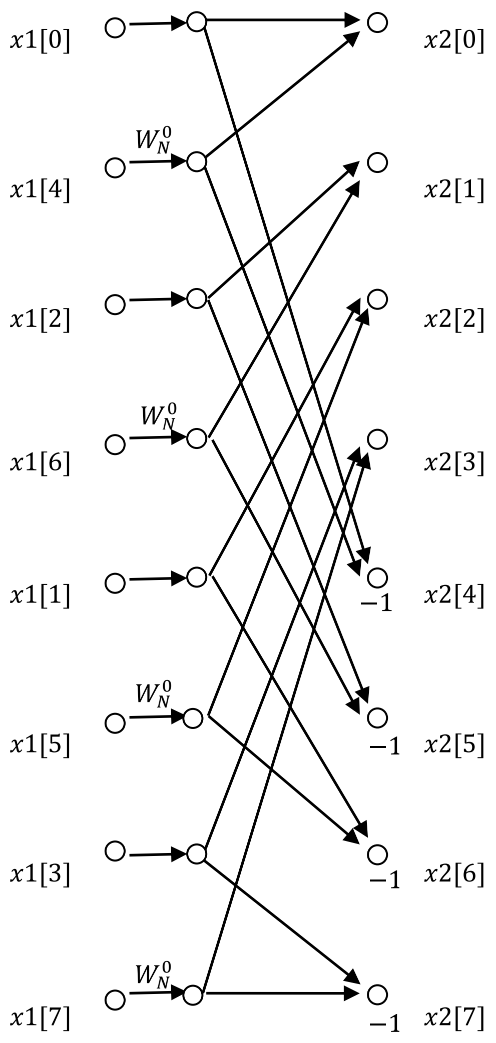


Figure 34: First stage of 8-input FFT

The second stage 8-input FFT has W_N^0 on the first two butterflies and W_N^2 on the last two butterflies as shown in Figure 35, where $x3$ is the input for the third stage. The structure of the second stage is identical to the first stage, with different twiddle factors.

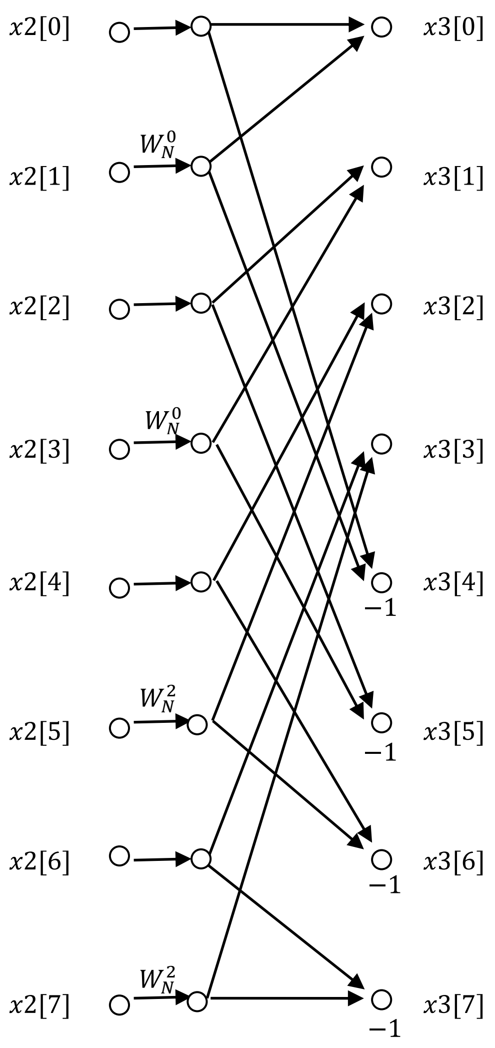


Figure 35: Second stage of 8-input FFT

The last stage of 8-input FFT has twiddle factors $W_N^0, W_N^1, W_N^2,$ and W_N^3 for the butterflies from top to bottom as shown on Figure 36. The structure of the third stage is identical to the previous stages, with different twiddle factors.

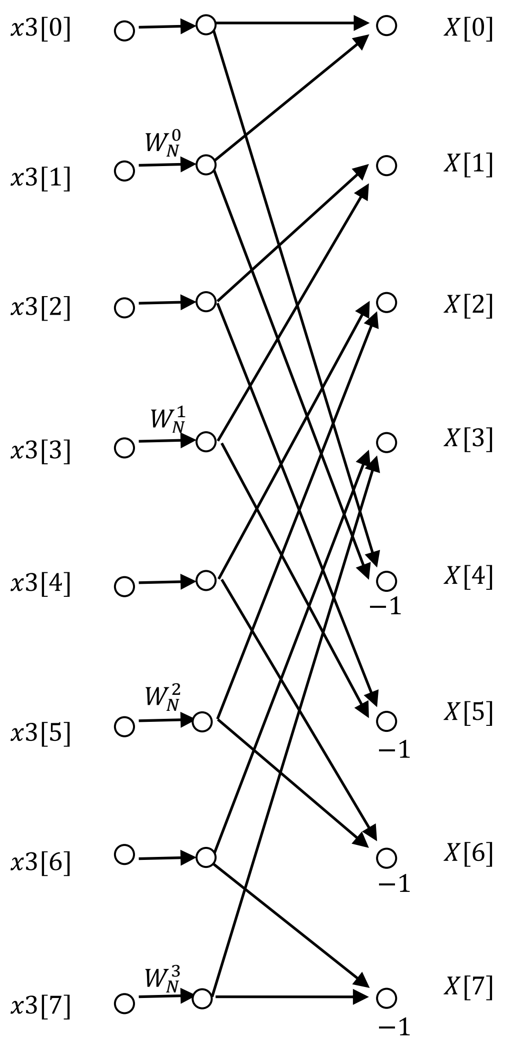


Figure 36: Third stage of 8-input FFT

Based on the observation on the twiddle factors on different stages of 8-input FFT, it is obvious that there is a certain pattern in twiddle factors for different stages. For singleton architecture, the pattern of twiddle factors is shown in table 3 below.

Table 3: Twiddle factor pattern

$Stage\ number$	$Twiddle\ factor$	$Twiddle\ factor\ pattern$
$1st\ stage$	$W_n^i, i = 0\ with \frac{N}{2} repeats$	$W_n^0, W_n^0, W_n^0 \dots$
$2nd\ stage$	$W_n^i, i = 0, N/4\ with \frac{N}{4} repeats$	$W_n^0, W_n^0, W_n^0 \dots W_n^{N/4}, W_n^{N/4}, W_n^{N/4} \dots$
$3rd\ stage$	$W_n^i, i = 0, \frac{N}{8}, \frac{N}{4}, \frac{3N}{8}\ with \frac{N}{8} repeats$	$W_n^0, W_n^0 \dots W_n^{\frac{N}{8}}, W_n^{\frac{N}{8}} \dots W_n^{\frac{N}{4}}, W_n^{\frac{N}{4}} \dots W_n^{\frac{3N}{8}}, W_n^{\frac{3N}{8}} \dots$
$\dots$	$\dots$	$\dots$
$(Log2(N) - 2)th\ stage$	$W_n^i, i = \left(0,1,2 \dots \frac{N}{8} - 1\right) * 4$ $with\ four\ repeats$	$W_n^0, W_n^0, W_n^0, W_n^0, W_n^1, \dots, W_n^2, \dots, \dots, W_n^{\frac{N}{2}-4}, \dots$
$(Log2(N) - 1)th\ stage$	$W_n^i, i = \left(0,1,2 \dots \frac{N}{4} - 1\right) * 2$ $with\ two\ repeats$	$W_n^0, W_n^0, W_n^1, W_n^1, W_n^2, W_n^2 \dots W_n^{\frac{N}{2}-2}, W_n^{\frac{N}{2}-2}$
$Log2(N)th\ stage$	$W_n^i, i = 0,1,2 \dots \frac{N}{2} - 1$ $with\ one\ repeat$	$W_n^0, W_n^1, W_n^2, W_n^3 \dots W_n^{\frac{N}{2}-1}$

By calculating the twiddle factors using Matlab, every stage's twiddle factor pattern can be formed and then implemented in VHDL. This automation process in Matlab greatly reduces the time for developing large input FFT. For example, an 64-input FFT would need 32 twiddle factors in each stage and there is a total of 6 stages, Matlab saves a lot of time by using software to form the twiddle factor codes in VHDL instead of writing them one by one by hand.

3.3.6 Revised butterfly architecture

As shown in Figure 31, the butterfly's output locations are slightly different from the unmodified butterfly's, since the Singleton approach is implemented in this study. The relationship between inputs (X_{m-1}) and outputs (X_m) of a butterfly are

$$X_m[p] = X_{m-1}[p] + W_N^r X_{m-1}[q]$$

$$X_m[q + N/2] = X_{m-1}[p] - W_N^r X_{m-1}[q]$$

A twiddle factor multiplier, TwiddleX, calculates the complex multiplication $W_N^r X_{m-1}[q]$. Figure 37 shows the overall twiddleX architecture, where W_N^r is a 32 bit complex number and $W_N^r X_{m-1}[q]$ is the product of the twiddle factor multiplier. Figure 38 shows twiddleX architecture in Quartus II RTL viewer. The overflow indicates whether there is an overflow occurring during multiplication either in real number multiplication or imaginary number multiplication.

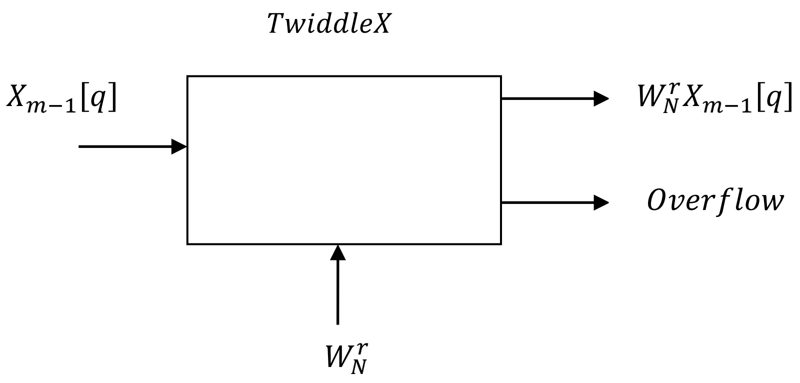


Figure 37: Overall twiddleX architecture

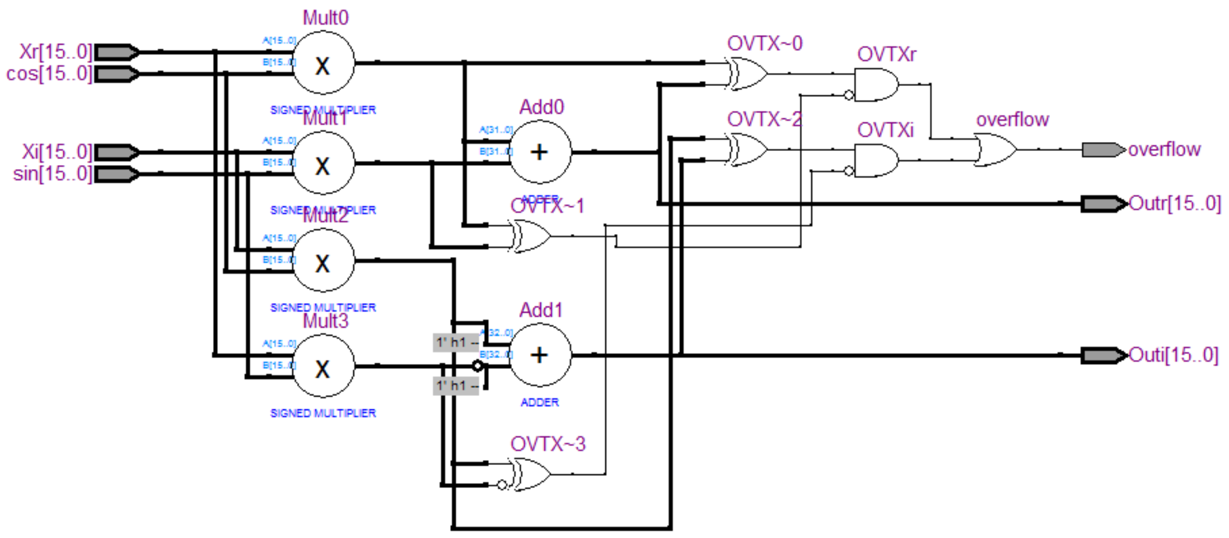


Figure 38: TwiddleX architecture in Quartus II RTL viewer

The twiddle factor multiplication, $W_N^r X_{m-1}[q]$, can be broken down into real and imaginary multiplication, where the W_N^r is supplied from the Wn_Rom. The complex multiplication is shown below:

$$X_{m-1}[q] * W_N^r$$
$$= \left(X_{real} + j * X_{imag}\right) * (W_{N_{real}}^r + j * W_{N_{imag}}^r)$$
$$= \left(X_{real} * W_{N_{real}}^r - X_{imag} * W_{N_{imag}}^r\right) + j * (X_{real} * W_{N_{imag}}^r + X_{imag} * W_{N_{real}}^r) \quad (3.3)$$

where X_{real} and X_{imag} are real and imaginary parts of $X_{m-1}[q]$, and ${W_N^r}_{real}$ and ${W_N^r}_{imag}$ are real and imaginary parts of W_N^r.

$Singleton\ Butterfly$

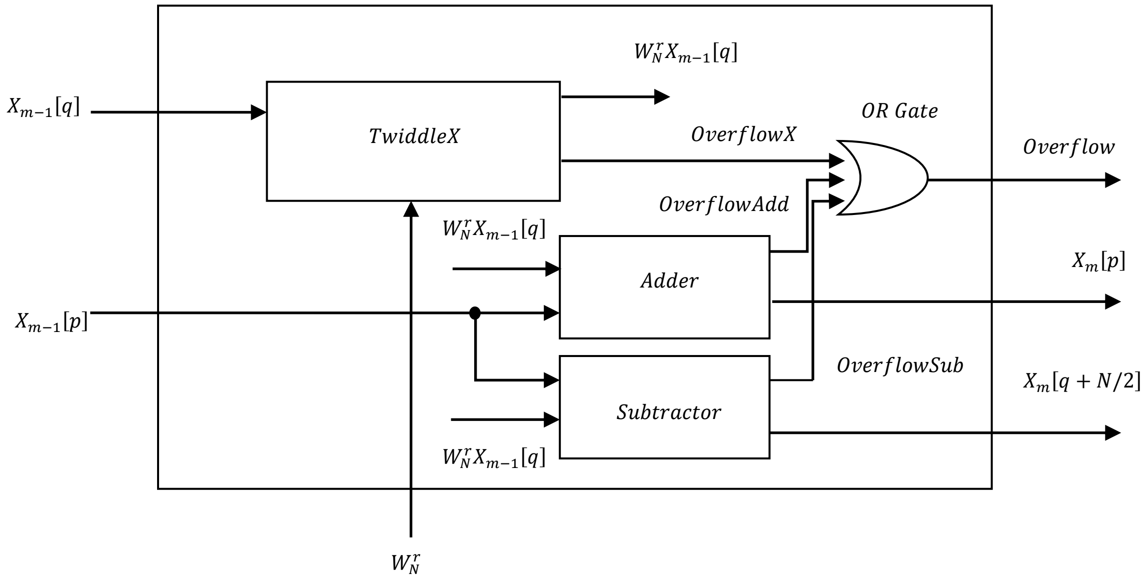


Figure 39: Singleton butterfly architecture

Figure 39 shows the internal structure of butterfly. In the butterfly unit, the TwiddleX unit calculates the product of the twiddle factor multiplication. The product, $W_N^r X_{m-1}[q]$, is then served as an input for a 32 bit adder and a 32 bit subtractor, while $X_{m-1}[p]$ serves as the other input. The output of the adder is $X_m[p]$, and the output of the subtractor is $X_m[q + N/2]$. Since overflow might occur during addition or subtraction in the adder or subtractor, overflow checking is required. It's important to track if there is overflow during these operations, as the data would be inaccurate if overflow occurs. To indicate if an overflow occurs in a butterfly, overflow in each sub-unit needs to be taken into consideration; so, a three input OR gate is used to check the overflow for the butterfly.

3.5 Modulation/Demodulator

Since baseband QPSK modulation is considered in this paper, two bits of data stream are modulated at each clock cycle. There are four possible values for the output of the modulation, as two bits can produce four different values as shown in table 4 below.

Table 4: Possible combination for 2 bits

Binary value
00
01
10
11

Gray code is used for the input stream since Gray code would greatly reduce the error rate in received signal. In Gray code, transitioning from neighboring symbols only makes a one-bit difference. The Gray code table is shown below.

Table 5: Gray code vs binary

Decimal	Gray	Binary
0	00	00
1	01	01
2	11	10
3	10	11

As shown in Table 5, the transition between ‘01’ and ‘10’, and the transition between ‘11’ and ‘00’ makes a two-bits difference in binary code. However, there is only a one-bit difference between neighboring symbol transitions in Gray code. This is a great advantage of Gray code over binary code and is the major reason Gray code is selected. Figure 40 shows the comparison between demodulations of Gray code and binary code

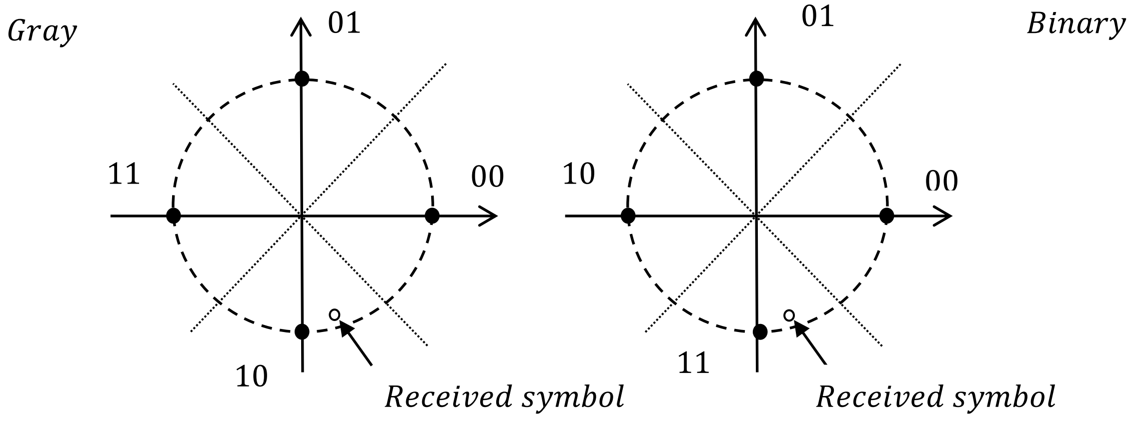


Figure 40: Comparison between demodulations of Gray code and binary code

As shown in Figure 40, it is supposed that the transmitted symbol is ‘00’, but that the received symbol falls outside of ‘00’ demodulated region. In Gray code, it will be demodulated as ‘10’, and it will be demodulated as ‘11’ in binary code. There is only a one-bit difference between the demodulated symbol and the transmitted symbol in Gray code, whereas both bits are demodulated incorrectly in binary code. Hence, Gray code would reduce the BER when comparing it to binary code.

The modulator has an internal ROM to store the modulated value for every symbol. These values are complex numbers, which are represented by 32-bit fixed numbers. Table 6 shows the modulator Rom for every symbol.

Table 6: Modulator ROM

Symbol	Modulated value
00	$\sqrt{E_s}\exp\left(j\frac{0\pi}{4}\right)$
01	$\sqrt{E_s}\exp\left(j\frac{2\pi}{4}\right)$
11	$\sqrt{E_s}\exp\left(j\frac{4\pi}{4}\right)$
10	$\sqrt{E_s}\exp\left(j\frac{6\pi}{4}\right)$

The modulator has a pretty straightforward structure. It takes two bits as inputs, and these inputs pass through modulator ROM to get the corresponding complex number output. The overall modulator architecture is shown in Figure 41, and Figure 42 shows the modulator architecture in Quartus II RTL viewer.

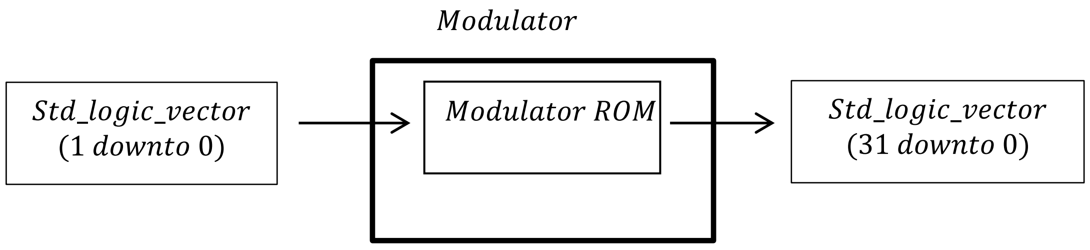


Figure 41: Overall modulator architecture

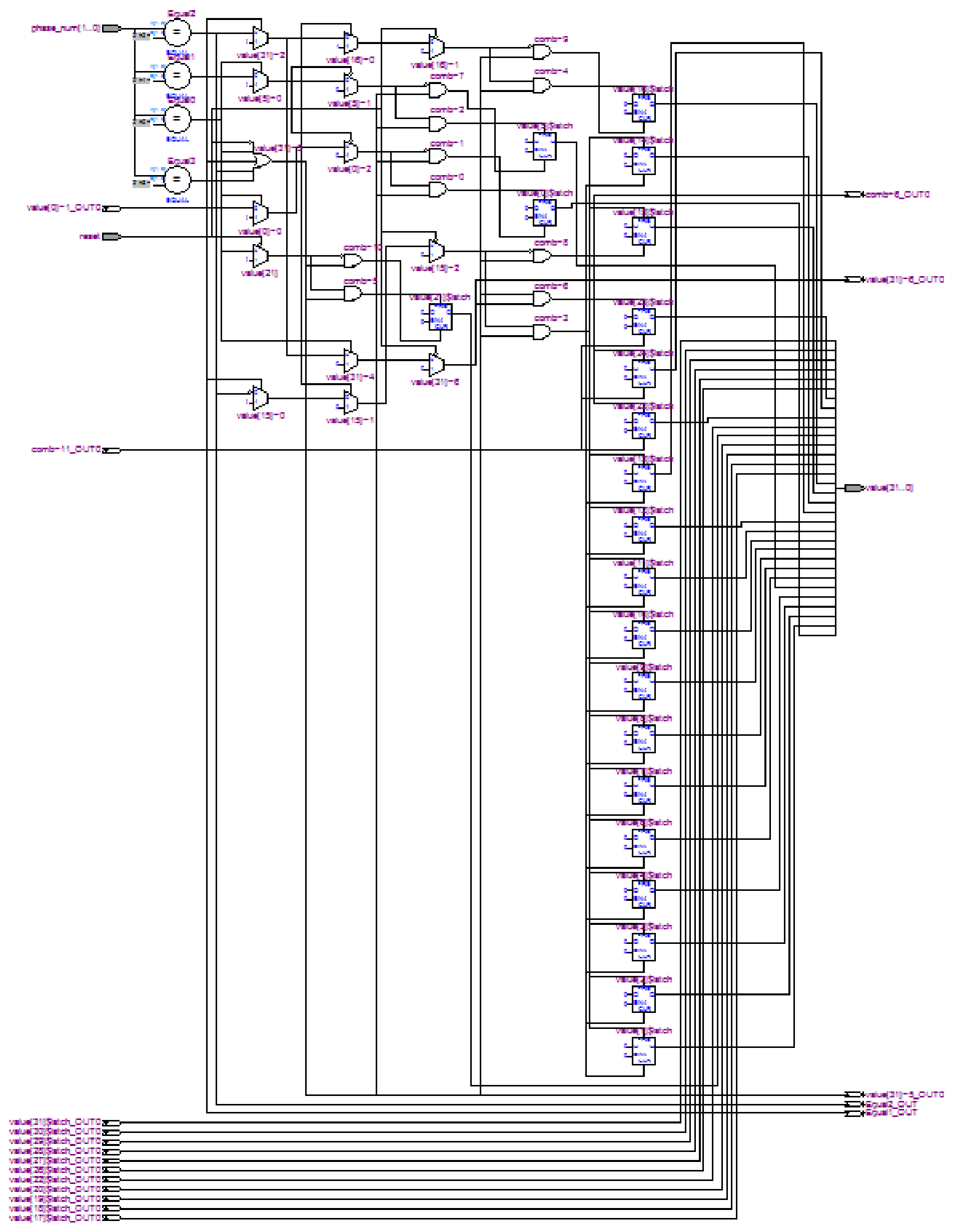


Figure 42: Architecture of modulator in Quartus II RTL viewer

Once the symbol is received by the demodulator, the symbol will be demodulated based on the shortest Euclidean distance between the received symbol and possible transmitted symbols. Figure 43 shows an example of demodulation.

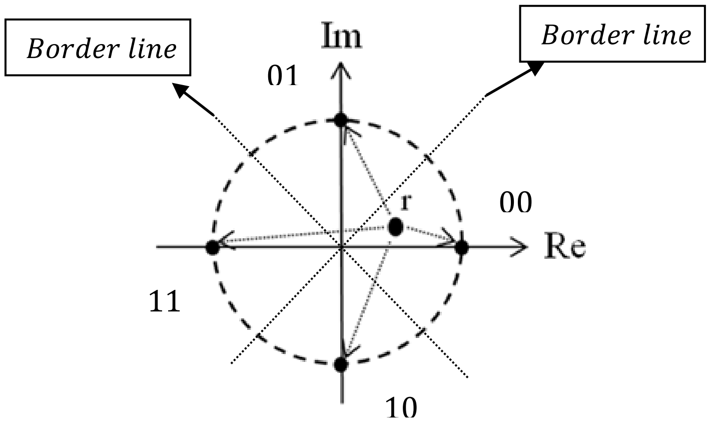


Figure 43: An example of demodulation on QPSK

Since the received signal falls in the region of ‘00’, which means it has shortest Euclidean distance to symbol ‘00’, it will be demodulated as ‘00’. If the received signal falls between the border lines, the shortest Euclidean distance to one of the symbols is unique. However, if the received signal falls right on one of the border lines, the shortest Euclidean distance from the received signal is the same to the two symbols. In the extreme case, the received signal might fall on the intersection of two border lines, which is the origin; the shortest Euclidean distance would be the same to every symbol. To prevent the difficulty if they happen, these situations need to be handled separately as shown in Table 7. Figure 44 shows the demodulator unit in Quartus II RTL viewer.

Table 7: Look up table for demodulator

Received signal	*Demodulated symbol*
$Real > 0\ and\ Real > abs(Imag)$ $Real > 0\ and\ Real = Imag$ $Real = 0\ and\ Imag = 0$	00
$Imag > 0\ and\ Imag > abs(Real)$ $Imag > 0\ and\ Imag = -Real$	01
$Real < 0\ and\ Real < -abs(Imag)$ $Real < 0\ and\ Real = Imag$	10
$Imag < 0\ and\ Imag < -abs(Real)$ $Imag < 0\ and\ Imag = -Real$	11

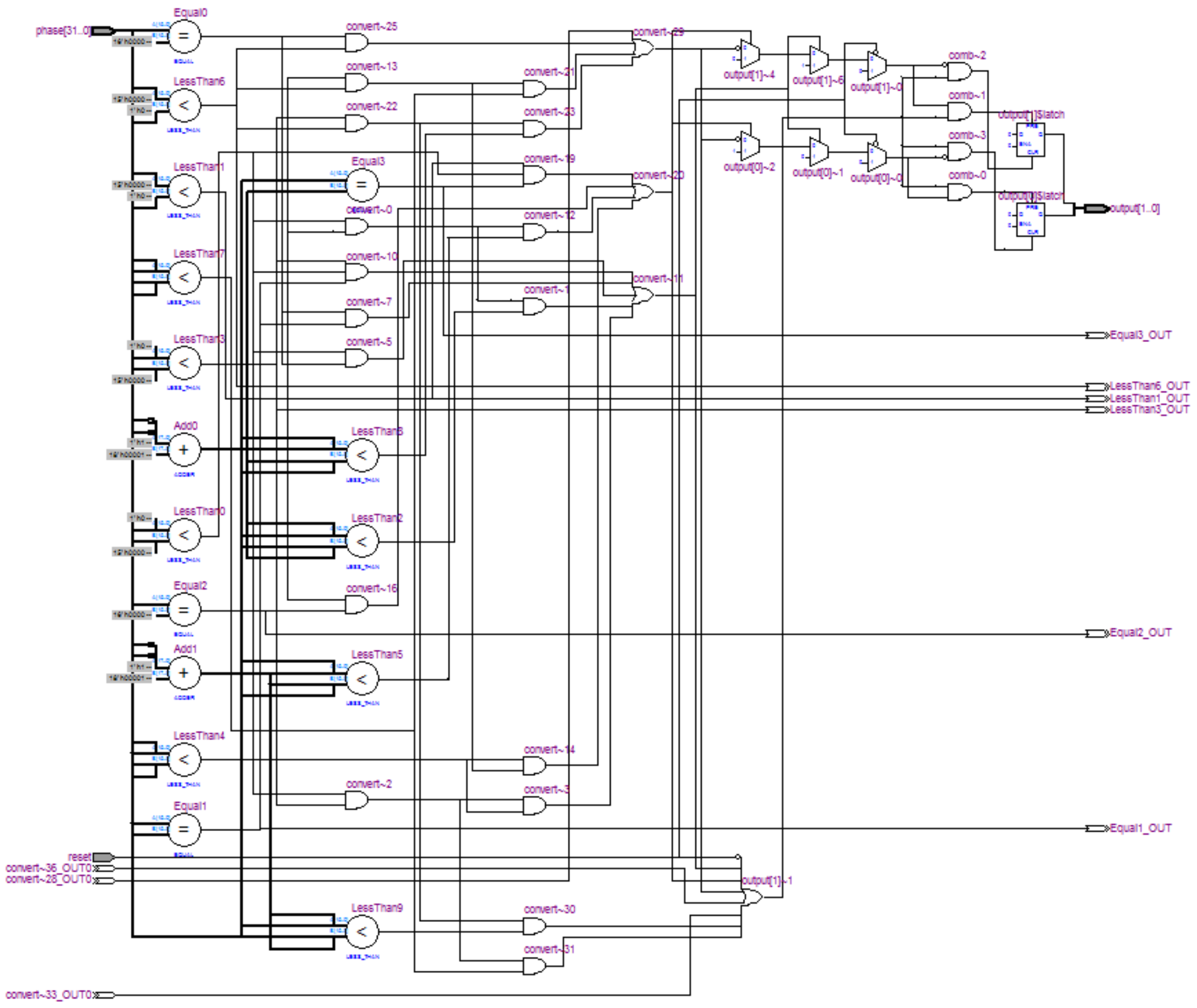


Figure 44: Demodulator in Quartus II RTL viewer

3.6 RRC filter

The RRC filter can be designed by using the Matlab function-‘rcosine’. In this study, a 61-tap RRC filter with a 0.5 roll-off factor (α) is designed. Figure 45 shows the matlab simulation of the RRC filter coefficient.

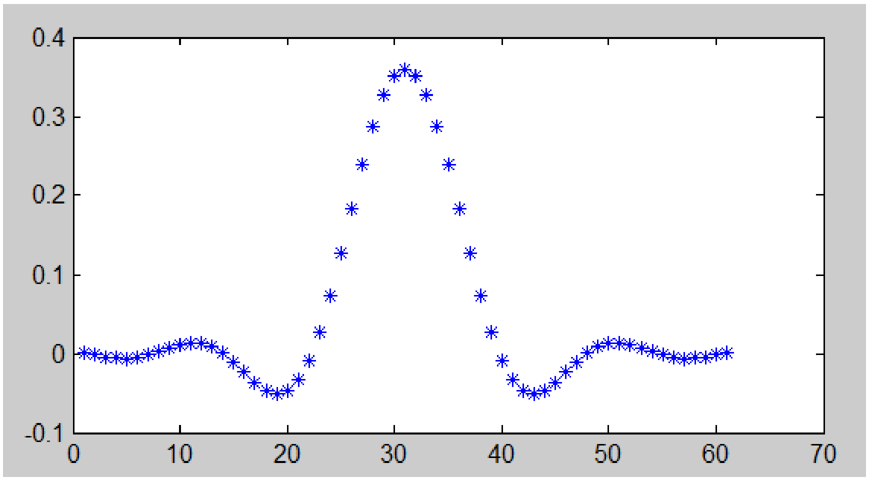


Figure 45: 61-tap RRC filter with α=0.5

Each tap of the RRC filter shown in Figure 44 is a coefficient of the RRC filter. After transforming these coefficients into 16-bit fixed points in Matlab, they can be implemented into VHDL. For example, RRC_0 serves as the first point when the data goes through the filter. Each data will multiply every point in the filter once at a time. Figure 46 shows the overall RRC filter architecture.

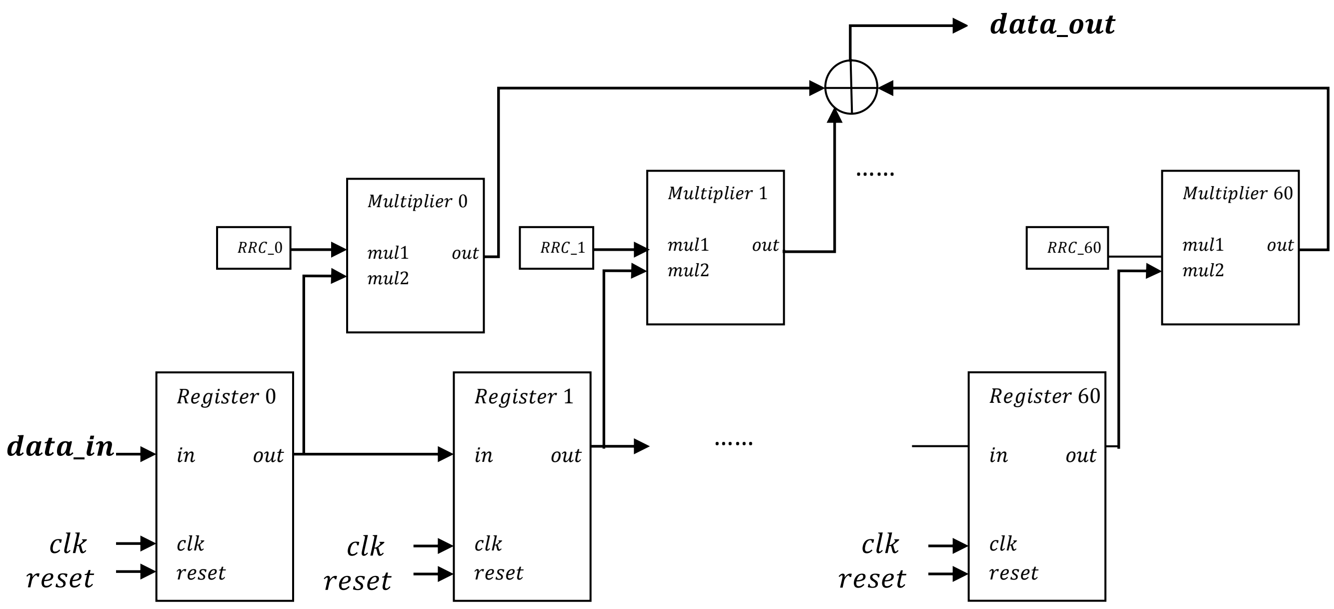


Figure 46: Overall RRC filter architecture

The transmitter has two RRC filters, one for real numbers and the other one for imaginary numbers. Since a 16-bit fixed number is used to represent a real number or an imaginary number, this would be sufficient enough to preserve the data precision. The output of the RRC filter is the summation of all outputs of multipliers. The output of the RRC filter is zero before the reset is turned on, since outputs of registers and multipliers are all zeros and the summation of zeros results in zero. After the reset is turned on, data is first stored into a 16-bit register in the RRC filter called “Register 0” at the rising clock edge. The output of “Register 0” is one of the inputs for Multiplier 0, the product of the RRC_0 coefficient and the “Register 0” output would be output of the RRC filter since the outputs of the other multipliers are all zeros. As the data passes through each register, the data is filtered by the RRC. The clock used in the RRC filter has ten times the frequency rate when comparing it to the clock in other units in the transmitter or the receiver. Supposing the symbol period is Ts for the other unit in the transmitter or the receiver, the symbol period of the RRC is $Ts/10$. The input data for the RRC is padded with zeros between time intervals as shown in Figure 47.

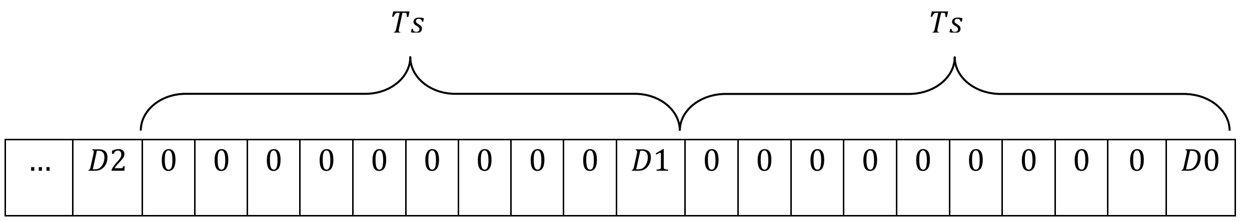


Figure 47: Input data format for RRC filter

As the data $D1$ passes through the filter, if $D0$ is still in one of the register in the filter, the output would be the summation of two multiplier outputs with one multiplier having $D0$ as one of its inputs and the other multiplier having $D1$ as one of its inputs.

The receiver has two RRC filters too, one for real numbers and the other for imaginary numbers. Once the signal is received, it is sampled at a rate of $Ts/10$. These sampled signals are then passed through the receiver's RRC filters. After sampling the signal at the output of the RRC at Ts, the sampled output will serve as inputs for the FFT.

Synchronization between the transmitter and the receiver is one of the most important things necessary to ensure data accuracy. If a synchronization error occurs, the result would be affected greatly over time. In this study, two wires, one the clock and the other the reset, are used to make the synchronization between transmitter and receiver.

3.7 Serial to parallel converter

Since the modulator output is in serial form and the input for the IFFT is in parallel form, a serial to parallel conversion is required. Figure 48 shows architecture of a serial to parallel converter.

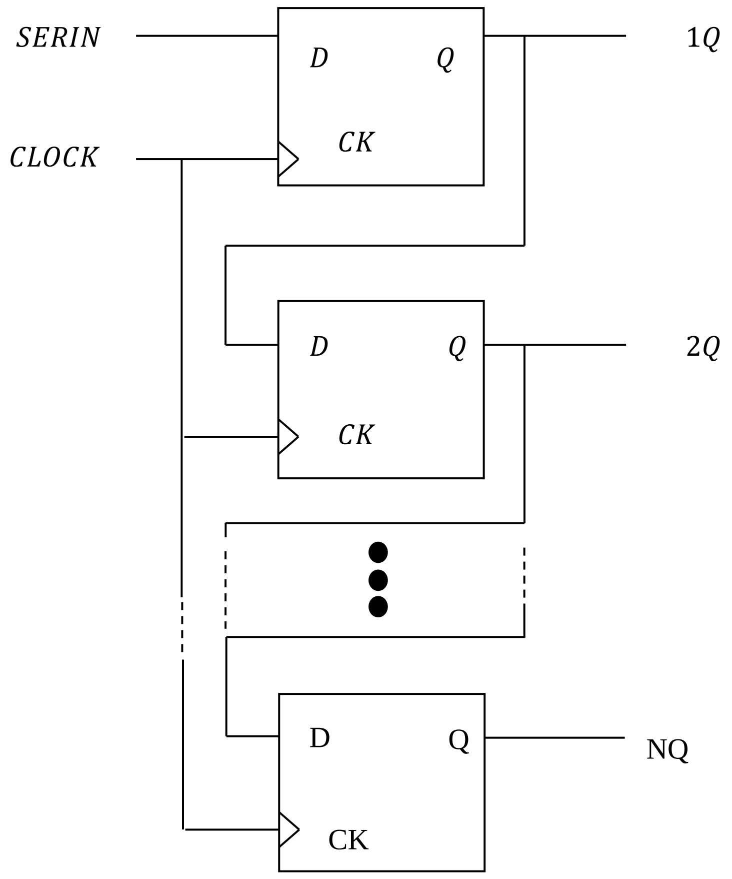


Figure 48: Serial to parallel converter

In the serial to parallel converter, a series of D flip-flops is used to output parallel data. The truth table of a D flip-flop is shown in Table 8. These D flip-flops are connected in a series, the output of previous D flip-flop serves as the input of the next D flip-flop. In this way, serial data are stored in each of D flip-flops. The outputs of these D flip-flops form the parallel data for the FFT unit.

Table 8: Truth table of D flip-flop

$Clock$	D	$Qnext$
$Rising\ edge$	0	0
$Rising\ edge$	1	1
$Non - Rising$	X	Q

3.8 Parallel to serial converter

The parallel to serial converter converts the parallel data from the FFT unit to serial data for the demodulator. Figure 49 shows the architecture of a parallel to serial converter.

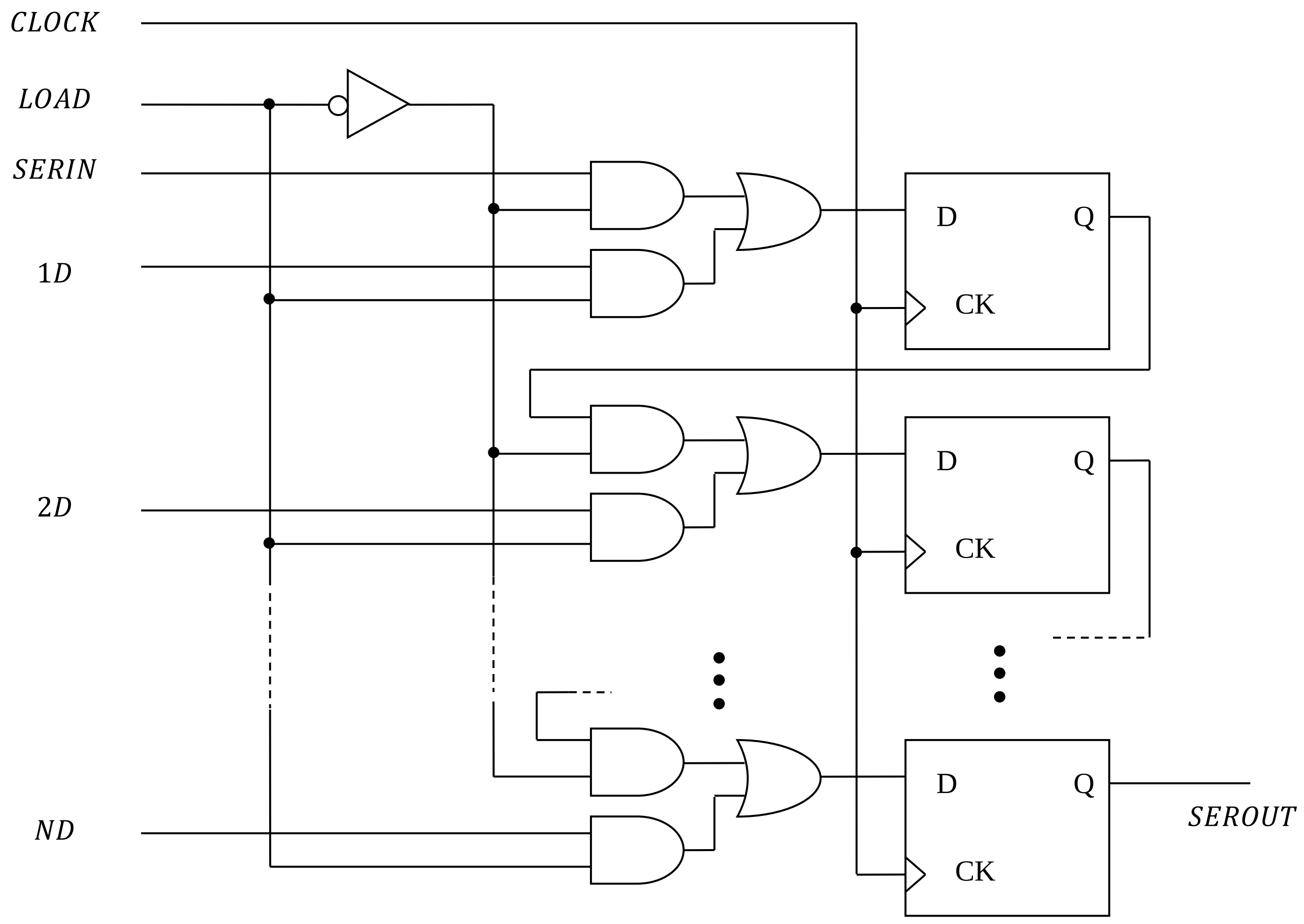


Figure 49: Parallel to serial converter

In a parallel to serial converter, parallel data $1D$ through ND are first loaded into the D flip-flops by setting $load$ = ‘1’. Once the data is loaded into the D flip-flops, the data starts the shift to the next D flip-flop at the rising clock edge by setting $load$ = ‘0’. The last D flip-flop outputs the serial data at the clocking rising edge. After N clocks, the parallel to serial data conversion is completed.

3.9 Diagrams of the entire system

Since an OFDM system is composed of a transmitter and a receiver, two FPGA boards are used for implementation. One is used to implement as a transmitter, the other as a receiver. A Sel signal in the transmitter is used to output a selected IFFT output on the LCD display. A

Sel signal in the receiver is used to output a selected FFT output on the LCD display. The LCD display is used to examine the data of OFDM system in FPGAs by comparing it with the matlab simulation results. Figure 50 and Figure 51 show the OFDM transmitter and receiver in Quartus II RTL viewer.

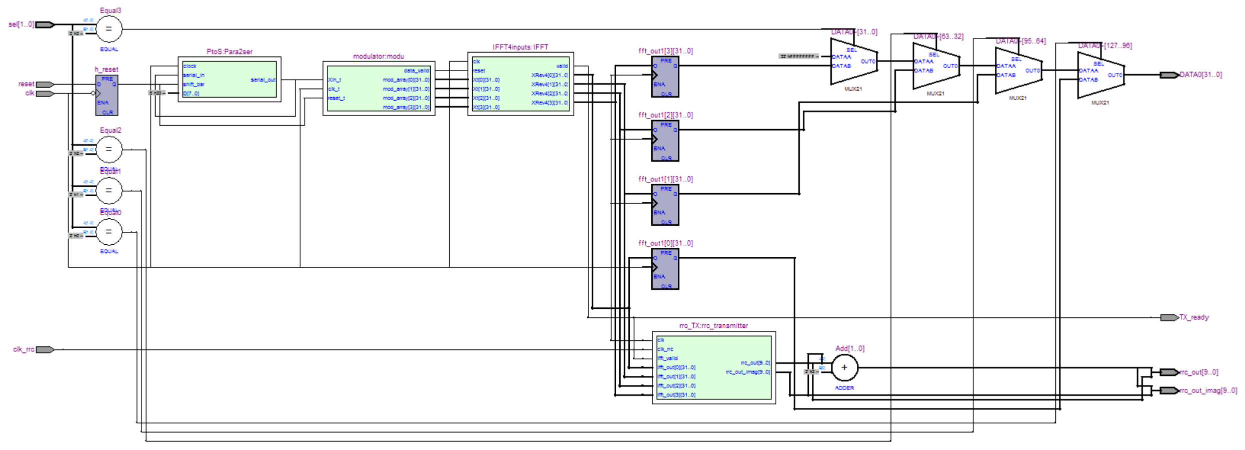


Figure 50: OFDM transmitter in Quartus II RTL viewer

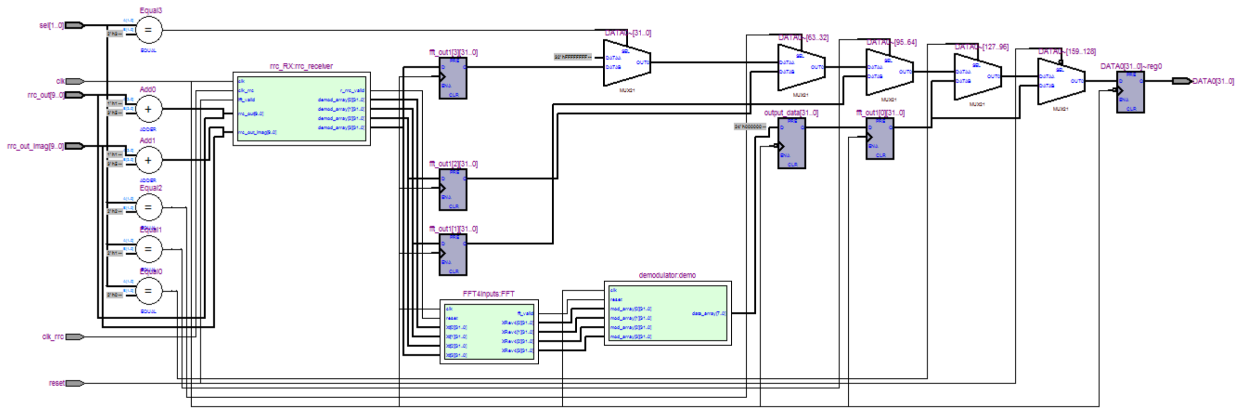


Figure 51: OFDM receiver in Quartus II RTL viewer

Chapter 4. Results and Discussions

4.1 Matlab simulation

Matlab is a powerful programming environment for algorithm development, data analysis, visualization, and numerical computation. By using Matlab to simulate communication theory, it can quicken the process over traditional programming languages, such as C, C++, and Java.

In the Matlab simulation for OFDM, the OFDM has a 128-input IFFTs and QPSK modulator for the transmitter, a 128-input FFT and QPSK demodulator for the receiver, and the wireless channel between transmitter and receiver is assumed to be AWGN and Raleigh fading. The simulation result of OFDM is shown in Figure 52.

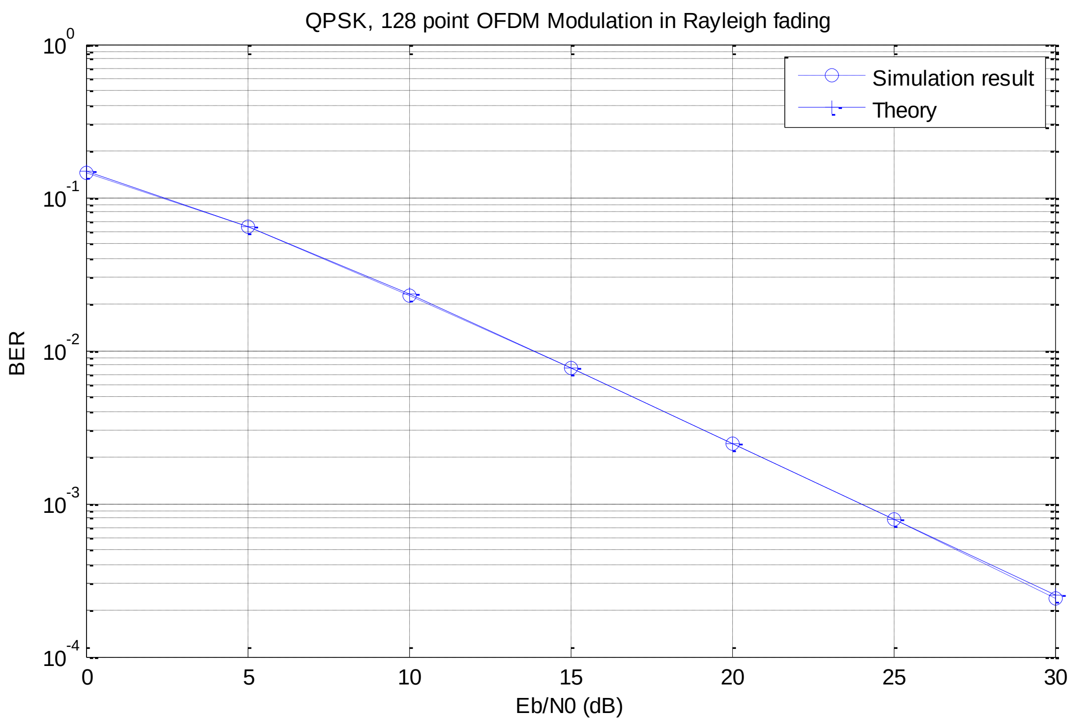
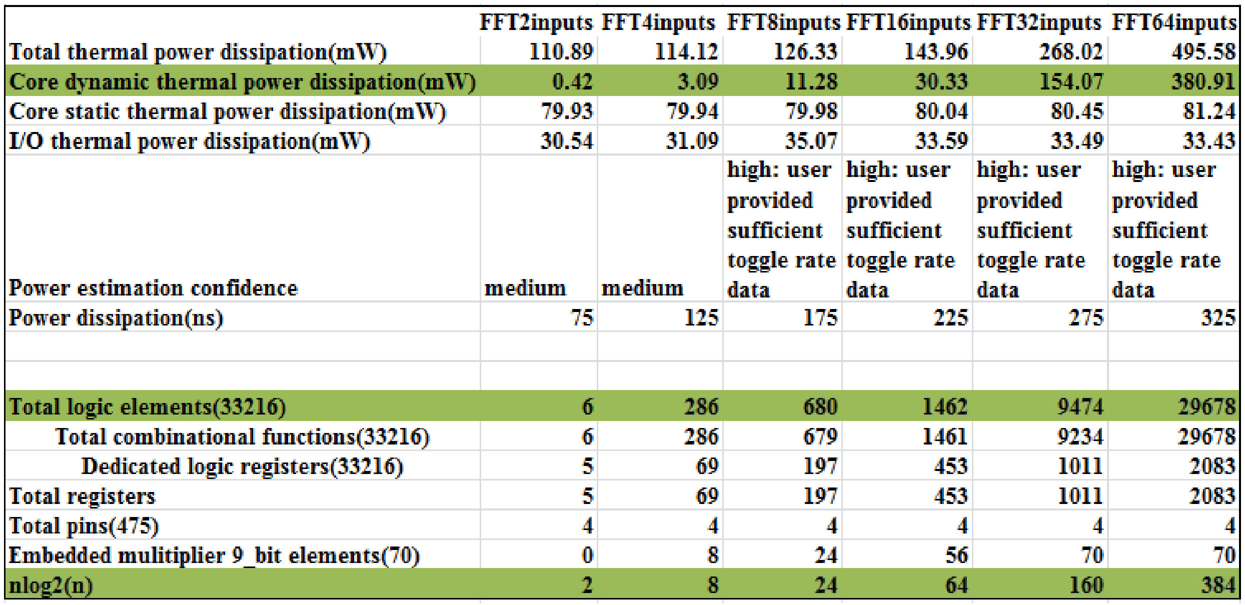


Figure 52: Performance of OFDM

4.2 ModelSim simulation

ModelSim is a great simulation environment to simulate FPGA designs. Since ModelSim simulate the gate level designs without implementing it on the actual board, it improves the development life span by shortening the simulation time. Since the FFT unit is the most complex unit in the design, it would be beneficial to figure out the relationship among FFT input number, complexity, and power dissipation level. Table 9 below indicates the relationships between these three elements.

Table 9: Relationships among FFT input number, complexity, and power consumption



	FFT2inputs	FFT4inputs	FFT8inputs	FFT16inputs	FFT32inputs	FFT64inputs
Total thermal power dissipation(mW)	110.89	114.12	126.33	143.96	268.02	495.58
Core dynamic thermal power dissipation(mW)	0.42	3.09	11.28	30.33	154.07	380.91
Core static thermal power dissipation(mW)	79.93	79.94	79.98	80.04	80.45	81.24
I/O thermal power dissipation(mW)	30.54	31.09	35.07	33.59	33.49	33.43
Power estimation confidence	medium	medium	high: user provided sufficient toggle rate data	high: user provided sufficient toggle rate data	high: user provided sufficient toggle rate data	high: user provided sufficient toggle rate data
Power dissipation(ns)	75	125	175	225	275	325
Total logic elements(33216)	6	286	680	1462	9474	29678
Total combinational functions(33216)	6	286	679	1461	9234	29678
Dedicated logic registers(33216)	5	69	197	453	1011	2083
Total registers	5	69	197	453	1011	2083
Total pins(475)	4	4	4	4	4	4
Embedded mulitiplier 9_bit elements(70)	0	8	24	56	70	70
nlog2(n)	2	8	24	64	160	384

As shown in Table 9 above, increasing increments in FFT inputs result in more core dynamic power thermal dissipation and total logic elements. In order to compare each parameter more visually, $Nlog_2(N)$ is used to serve as a base line in the Figure 53 below. Normalized total logic elements and core dynamic thermal power dissipation (mW) served to compare them with $Nlog_2(N)$.

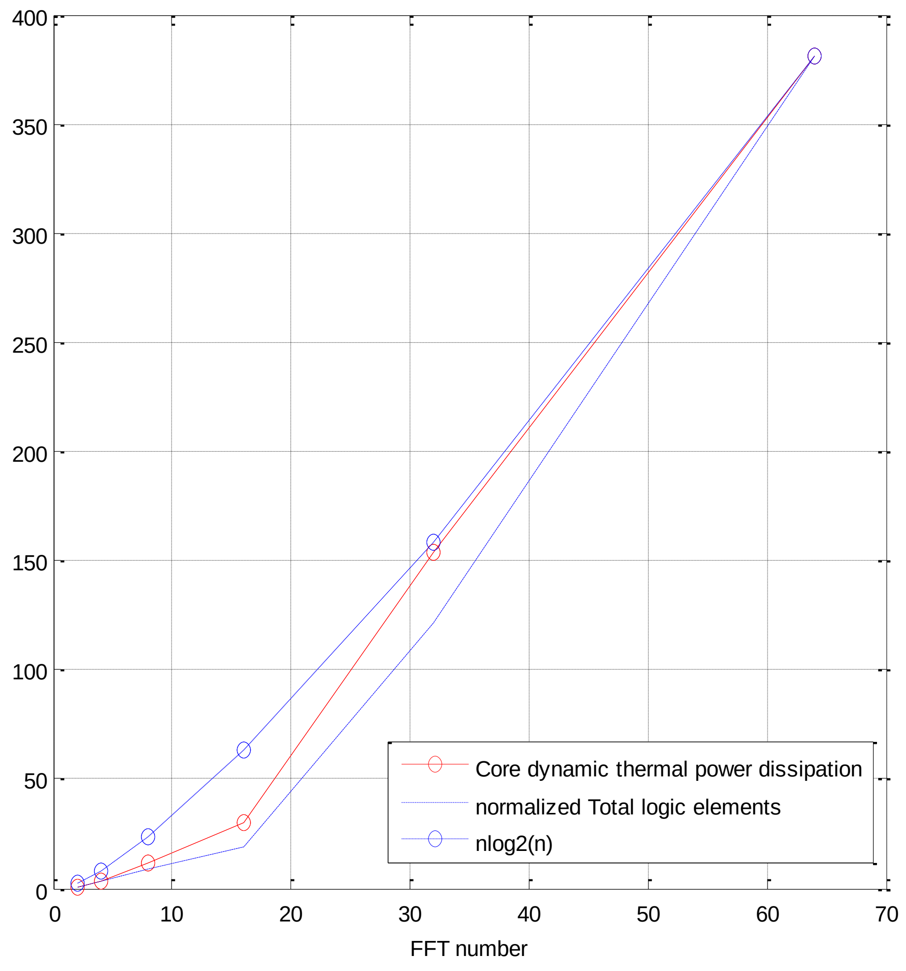


Figure 52: FFT complexity vs power consumption

Based on the Figure 53, it is obvious that dynamic thermal power dissipation and normalized total logic elements match with the $Nlog_2(N)$ slope.

Since fixed point is used in the implementation, data might experience some precision loss during calculation. It would be important to check the error percentages for FFT, as it is the largest unit in the design. By comparing different fixed point FFT inputs in Modelsim with FFT floating point computation results, it is found that the accuracy level is down to 3 digits. Since 16 digits are used to represent a fixed point number, the error percentage of 3 digits over 16 digits is

$$\frac{2^3}{2^{16}} * 100\% = 0.0122\%$$

As shown in the calculation result above, the error percentage is pretty small when compared to 16 full digits. However, the percentage of error varies from comparisons at different FFT numbers. Based on the observation, when the compared floating number is small, its fixed point result would possibly have a larger percentage error.

4.3 FPGA implementation

In FPGA implementation, actual logic gate circuits are formed and simulated on the hardware. The whole simulation process takes longer when compared to ModelSim simulation, but it is the better method to examine the design. In the implementation, two FPGA boards are used. One is used as a transmitter, and the other one is used as receiver. In order to check data from FPGA, LCD on FPGA is used to display the desired output to compare with ModelSim. Some points are sampled from FFT outputs and displayed in Figure 54 below and compared with the ModelSim simulation output. This Figure also displays the final output of FPGA.

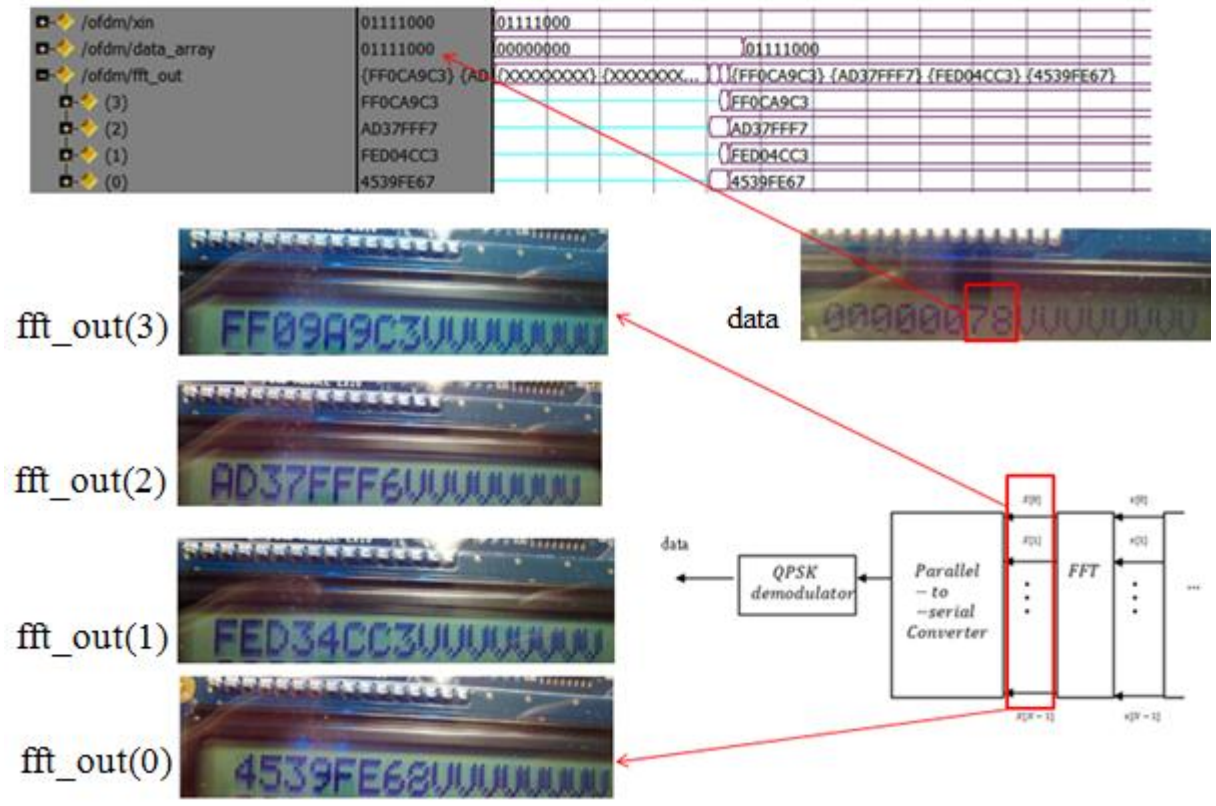


Figure 54: Comparison of FPGA outputs and ModelSim outputs

After comparing each FFT output from FPGA and ModelSim, it is obvious that the FFT outputs from the implementation are pretty much identical to ModelSim. Also, the OFDM input '78' is being demodulated correctly.

Chapter 5. Conclusions and Future Work

Through theoretical background studies for OFDM, OFDM is being simulated in Matlab and implemented on hardware. OFDM is robust to ICI and ISI. Through this paper, various number input FFTs are implemented on FPGAs. Complexity and total power consumption of these FFTs are examined by constructing graphs and tables. Each OFDM building block is implemented on FPGAs and examined. Future works will include channel estimations and perform maximum ratio combining (MRC) on oversampled OFDM (OOFDM) receivers, and use third party hardware to transmit signals wirelessly.

References

[1] Jingxian Wu and Yahong Zheng, "Oversampled orthogonal frequency division multiplexing in doubly selective fading," *IEEE Trans. Commun.*, vol. 59, pp. 815-822, Mar. 2011.

[2] Stefan Parkvall, Erik Dahlman, Anders Furuskär, Ylva Jading, Magnus Olsson, Stefan Wänstedt, and Kambiz Zangi. "LTE Advanced – Evolving LTE towards IMT-Advanced," in *Proc. The 68th IEEE Vehicular Technology Conference*, Sept. 2008.

[3] Volnei A. Pedroni, *Circuit design with VHDL*. MIT Press, 2004.

[4] Alan V. Oppenheim, Ronald W. Schafer, and John R. Buck, Discrete-time Signal Processing, *Prentice Hall*, 1999.

[5] Andrea Goldsmith, *Wireless Communications*. Cambridge University Press, 2005.

[6] Jingxian Wu. “ELEG5693 Wireless Communication.” Internet: http://comp.uark.edu/~wuj/teaching/eleg5693/eleg5693.html, Dec. 7, 2008 [Feb. 20, 2012].

[7] James Parkerson. “CSCE2114-Digital Design.” Internet: http://www.csce.uark.edu/~jparkers/CSCE2114-spring2012/DE2_Introduction_box.pdf. [Feb. 25, 2012].

[8] ALTERA. “DE2 Development and Education Board.” Internet: http://www.altera.com/education/univ/materials/boards/de2/unv-de2-board.html. [Feb. 27, 2012]

[9] Nasser Kehtarnavaz, Real-Time Digital Signal Processing: Based on the TMS320C6000, *Newnes*, 2004.