DESIGN OF A HIGH SPEED OFDM TRANSMITTER AND RECEIVER

By
Foisal Ahmed

Post Graduate Diploma (PGD)
In
Information and Communication Technology

Institute of Information and Communication Technology
BANGLADESH UNIVERSITY OF ENGINEERING AND TECHNOLOGY
2014

The project titled “Design of a high speed OFDM Transmitter and Receiver” Submitted by Foisal Ahmed ID No: 0411311006, Session April, 2011 has been accepted as satisfactory in partial fulfillment of the requirement for the degree of Post Graduate Diploma in Information and Communication Technology held on the 26th February, 2014.

BOARD OF EXAMINERS

_____ Supervisor

1. Dr. Md. Liakot Ali
   Professor
   Institute of Information and Communication Technology
   BUET, Dhaka-1000.

_____ Member

2. Dr. Mohammad Jahangir Alam
   Professor
   Department of Electrical and Electronic Engineering
   BUET, Dhaka-1000.

_____ Member

3. Dr. Mohammad Shah Alam
   Assistant Professor
   Institute of Information and Communication Technology
   BUET, Dhaka-1000.

Declaration

I, do hereby solemnly declare that this research work fully or any part of it has not been submitted elsewhere for the award of any degree or diploma.

…….

Foisal Ahmed

ACKNOWLEDGEMENT

I would like to thank Almighty Allah for giving me the ability to complete this work successfully.

I pay my gratitude to my project supervisor, Dr. Md. Liakot Ali, Professor and Associate Director (Academics), Institute of Information & Communication Technology (IICT), BUET for his continuous support, careful guidance and direction throughout the project period.

I would like to thank to Prof. Dr. Md. Saiful Islam, Director, Institute of Information & Communication Technology (IICT), BUET for sharing his valuable experience and guidance.

Finally I acknowledge with thanks the help of all members of IICT and my classmates for their cordial cooperation and useful suggestion.

ABSTRACT

System–on-a-chip (SoC) is now a trend in digital design because it gives a lot of advantages over discrete electronic based product such as higher speed, lower power consumption, smaller size, lower cost etc. Reconfigurable platform such as FPGA, CPLD, and PLD etc. is now being used for designing and implementing SoC due to its low cost and high capacity and tremendous speed. In this project single chip Orthogonal Frequency Division Multiplexing (OFDM) transmitter and receiver has been designed using Verilog HDL. OFDM is a multi carrier modulation technique used in the various digital communication systems like 3G GSM, WiMAX, and LTE etc. The main advantage of this transmission technique is their robustness to channel fading in wireless communication environment. There are many applications of OFDM in communication such as digital audio Broadcasting, asynchronous digital subscriber line (ADSL) and high bit-rate digital subscriber line (HDSL) systems etc. In OFDM, two digital signal processing algorithm Fast Fourier transform (FFT) and Inverse Fast Fourier Transform (IFFT) are mainly involved. The 8-points IFFT/FFT decimation-in-frequency (DIF) with radix-2 algorithm have been analyzed and incorporated in the design. The design has been simulated on the FPGA platform under Altera's Quartus II environment. Simulation results show that each of the modules of the proposed OFDM is working as desired**.** The test output achieved from the simulation result of the OFDM has been verified with that of the MATLAB output.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Chapter 1

Introduction

1.1 Introduction

In this modern world wireless technology is becoming the main important things not only for communication but also in the field of industry day by day. In early 1990, a mobile phone was still expensive for a single man but today we see it at every hand even most teenagers have. They use it not only for calls but also for various multimedia and educational purposes. It has become possible only for the technological development in the field of digital communication and applied electronics. Now a day's most of the communication systems are digital like mobile communication, TV & Radio Communication, Wired and Wireless Internet, satellite communication and so on. Many of these communication systems use one of the most sophisticated techniques known as *orthogonal frequency division multiplexing* (OFDM) for its strong immunity against noise and other advantages.

1.2 Overview of Digital Communication System

Communication means to exchange information (voice, audio, video, text, picture etc.) from one place to another place. Communication system is regarding some things which involve these processes of information. The main parts of the communication system are transmitter, channel and receiver.

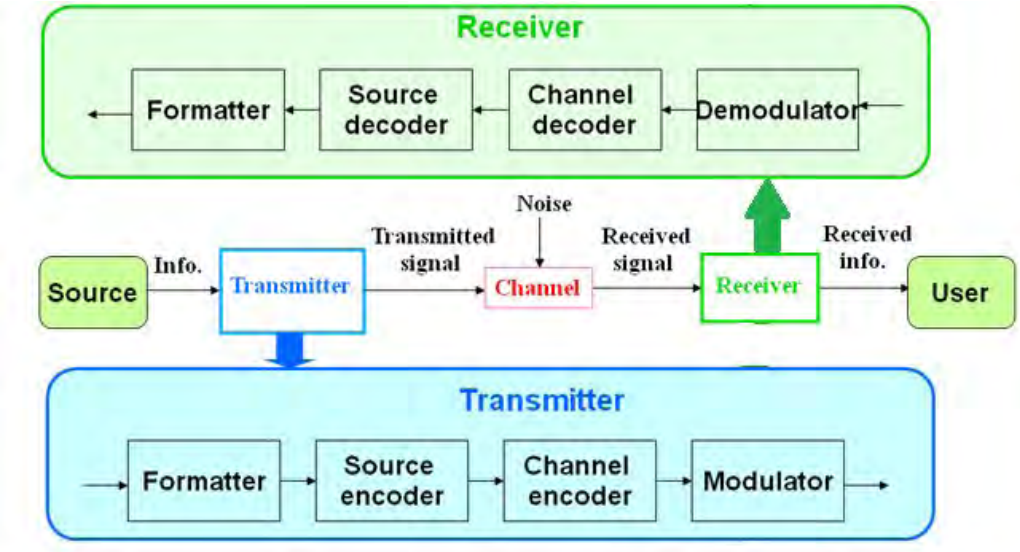


Figure 1.1 Digital Communication Systems

Figure 1.1 shows the block diagram of a basic communication system. The source of information is the messages that are to be transmitted to the other end in the receiver. A transmitter consists of formatter, source encoder, channel encoder and modulator. Firstly formatter transforms the signal from one type to another type suitable for the transmitter operation like transducer. Source

encoder is employed for an efficient representation of the information such that resources can be conserved. The main purpose of the channel encoder is for error correction and detection. The aim is to increase the redundancy in the data to improve the reliability of transmission. A modulation process converts the base band signal into band pass signal before transmission. When the signal is transferring through media (wired or wireless) it faces various noises which attenuate the signal amplitude and distort signal phase. Also, the signals transmitting through a channel also impaired by noise, which is assumed to be Gaussian distributed component.

In the receiver section, the opposite steps in the transmitter are occurred. The process in the receiving section is to be in such a way that same information could be recovered which is similar to the transmitted information.

1.3 Motivation of the project

Orthogonal Frequency Division Multiplexing (OFDM) is a multi carrier modulation technique used in the various digital communication systems like 3G GSM, WiMAX, and LTE etc. OFDM is based on a mathematical process called Fast Fourier Transform (FFT) and Inverse Fast Fourier Transform (IFFT), which enables number of channels to overlap without losing their individual characteristics (orthogonality) [1]. This is a more efficient use of the spectrum and enables the channels to be processed at the receiver more efficiently. When the data is transmitted at high data/bit rates over mobile radio channels then the channels may cause severe fading of transmitted signal when passed through channel and inter symbol interference (ISI).The focus of the fourth generation (4G) mobile system is on supporting higher data rates. Orthogonal Frequency Division Multiplexing (OFDM) is one of the promising techniques for 4G to mitigate ISI and fading in multi-path environment [2]. However, the main constraints nowadays for FFT processors used in wireless communication systems for OFDM are execution time and lower power consumption [3-4]. In conventional computer systems, the instructions are stored in the program memory and it is executed in sequential fashion. So inherently the whole process of operation becomes slow. FPGA technology is now an emerging technology which offers huge number of parallel processing in a single clock. The cost of the platform is also low. Due to availability of huge gate counts in a single chip, FPGA is now being used in various signal processing applications. It facilitates the implementation of the whole process in s single chip which in turn offers improved performance, lower power consumption, shorter time to market and higher reliability than the conventional computer platforms. Due to these attractive features, the proposed OFDM system has been implemented using the FPGA technology.

1.4 Project objectives

The project has the following objectives:

i. To design an OFDM base band transmitter and receiver using Verilog HDL
ii. To simulate the design using Quartus II simulator
iii. To verify the output with that generated using Mat lab Software
iv. To implement the design using DE2 Altera FPGA Education board

The outcome of this project will be a prototype OFDM base band transmitter and receiver..

1.5 Organization of the Project

The project is organized into five chapters, namely introduction, Fundamental concepts of OFDM and FPGA, Design of 8 point IFFT/ FFT for OFDM Transmitter/Receiver, Result of Verilog HDL Simulation and Conclusion.

Chapter 1 discusses the general idea of the project which covers the overview, project objective, project background and scope of the project.

Chapter 2 focuses the literature review of the OFDM system. The history and principle of the OFDM system, Fast Fourier Transform introduction, FPGA and Verilog programming basic idea is explained in this chapter.

Chapter 3 derives the Fast Fourier Transform and Inverse Fourier Transform algorithm using direct mathematical method. The equations are optimized for digital implementation.

Chapter 4 shows the Verilog simulation output. The results are presented in their sub-modules and then all the modules are combined to give the final output. Then, the Verilog output is compared with MATLAB simulation output.

Chapter 5 consists of conclusion, problems encountered in completing this project and suggestion to further improve this project.

Chapter 2

Fundamental Concept of OFDM and FPGA

2.1 Introduction

OFDM is a multi carrier high speed modulation technique used in the various digital communication systems like 3G GSM, WiMAX, and LTE etc. The ultimate goal of wireless communication technology is to provide universal personal and multimedia communication irrespective of mobility and location with high data rates. Orthogonal Frequency Division Multiplexing (OFDM) is one of the promising technique for 4G to mitigate ISI and fading in multi-path environment.

2.1.1 OFDM overview

The Orthogonal Frequency Division Multiplexing (OFDM) is developed to support high data rate and can handle multi carrier signals. Its specialty is that, it can minimize the Inter Symbol Interference (ISI) much more compared to other multiplexing schemes. It is more likely improved Frequency Division Multiplexing (FDM) as FDM uses guard band to minimize interference between different frequencies which wastes a lot of bandwidth but OFDM does not contain inter-carrier guard band which can handle the interference more efficiently than FDM. On top of that, OFDM handles multipath effect by converting serial data to several parallel data using Fast Fourier Transform (FFT) and Inverse Fast Fourier Transform (IFFT).

2.1.2 Data transmission

Data transmission in OFDM is high enough compared to FDM as OFDM follows multicarrier modulation. For this, OFDM splits high data bits into low data bits and sends each sub-stream in several parallel sub-channels, known as OFDM subcarriers. These subcarriers are orthogonal to each other and the each subcarrier bandwidth is much lesser than the total bandwidth. Inter Symbol Interference is reduced in OFDM technique as the symbol time T_s of each sub-channel is higher than the channel delay spread ς.

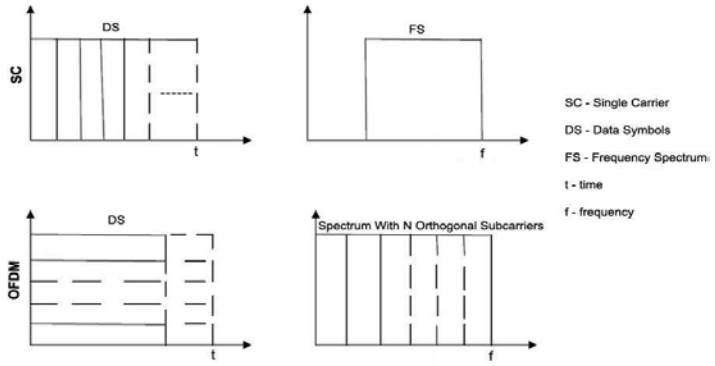


Fig 2.1: Time and Frequency diagram of Single and Multi-carrier signals

In the Fig 2.1, it is clear that OFDM resists the multipath effect by adopting smaller frequency bandwidth and longer period of time which leads to get better spectral efficiency.

2.1.3 Parameters

The implementation of OFDM physical layer is different for two types of WiMAX. For fixed WiMAX, FFT size is fixed for OFDM-PHY and it is 256 but for mobile WiMAX, the FFT size for OFDMA-PHY can be 128, 512, 1024 and 2048 bits [16]. This helps to combat ISI and Doppler spread. Other difference between OFDM-PHY and OFDMA-PHY is, OFDM splits a single high bit rate data into several low bit rate of data sub-stream in parallel which are modulated using IFFT whereas OFDMA accepts several users' data and multiplex those onto downlink sub-channel. Uplink multiple accesses are provided through uplink sub-channel. OFDM-PHY and OFDMA-PHY parameters are discussed briefly in the following subsection.

2.1.4 Sub-channelization

WiMAX divides the available subcarriers into several groups of subcarriers and allocates to different users based on channel conditions and requirement of users. This process is called sub-channelization. Sub-channeling concentrates the transmit power to different smaller groups of subcarrier to increase the system gain and widen up the coverage area with less penetration losses that cause by buildings and other obstacles. Without sub-channelization, the link budget would be asymmetric and bandwidth management would be poor [17]. Fixed WiMAX based OFDM-PHY permits a little amount of sub-channelization only on the uplink. Among 16 standard sub-channels, transmission can take place in 1, 2, 4, 8 or all sets of subchannels in the uplink of the Security Sub-layer (SS). SS controls the transmitted power level up and down depending on allotted sub-channels. When the allotted sub-channels increase for uplink users, the transmitted power level increases and when the power level decreases, it means the allotted sub-channels decreased. The transmitted power level is always kept below the maximum level. In fixed WiMAX, to improve link budget and the performance of the battery of the SS, the uplink sub-channelization permits SS to transmit only a fraction of the bandwidth usually below 1/16 allocated by the BS Mobile WiMAX's OFDMA-PHY permits sub-channelization in both uplink and downlink channels. The BS allocates the minimum frequency and sub-channels for different users based on multiple access technique. That is why this kind of OFDM is called OFDMA (Orthogonal Frequency Division Multiple Access). For mobile application, frequency diversity is provided by formation of distributed subcarriers. Mobile WiMAX has several distributed carrier based sub-channelization schemes. The mandatory one is called Partial Usage of Sub-Carrier (PUSC).

Another sub-channelization scheme based on unbroken subcarrier is called Adaptive Modulation and Coding (AMC) in which multiuser diversity got the highest priority. In this, allocation of sub-channels to users is done based on their frequency response. It is a fact that, contiguous sub-channels are best suited for fixed and low mobility application, but it can give certain level of gain in overall system capacity [16].

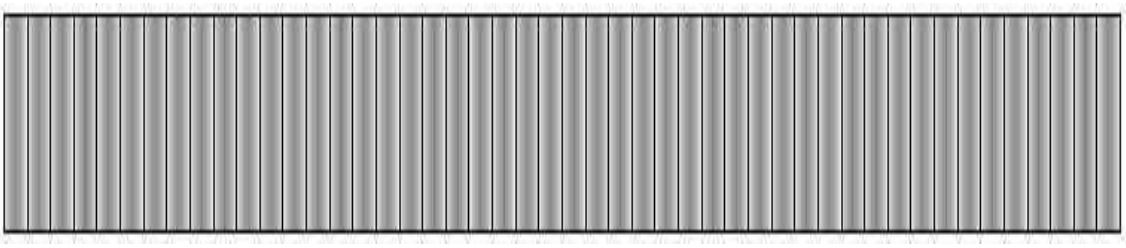


Fig 2. 2: Downstream transmission of OFDM spectrum

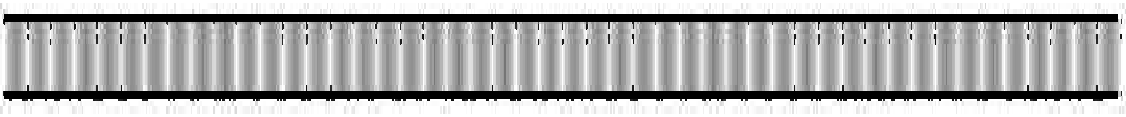


Fig 2. 3: Upstream transmission of OFDM spectrum

Figure 2.3 shows the upstream transmission of OFDM spectrum from a customer premises equipment (CPE) where the carriers are quarter in size compared to fig 2.2 downstream transmission from BS

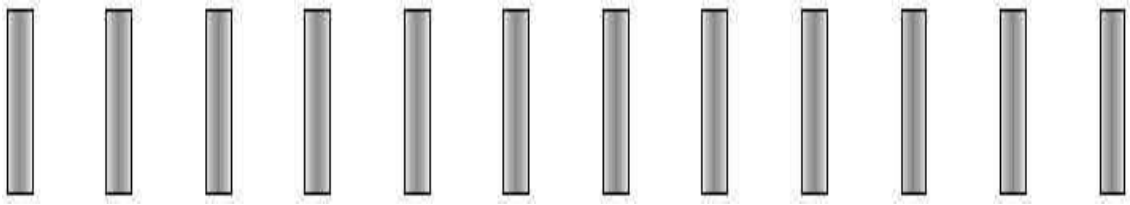


Fig 2. 4: Upstream transmission of OFDM spectrum from the CPE

Figure 2.4 illustrates the transmitted upstream OFDM spectrum from a CPE where the carriers are as same as BS in size and range but with small capacity. [13]

2.1.5 Basic OFDM implementation

Figure 2.5 shows the block diagram of basic OFDM communication system model. The Figure 2.5 indicates the step by step operation of each functional block. At the transmitter side, the input binary data stream obtained from the source. The retrieved binary data stream is passed through to the other required steps (source of encoder, channel coder) as the signal has to transmit.

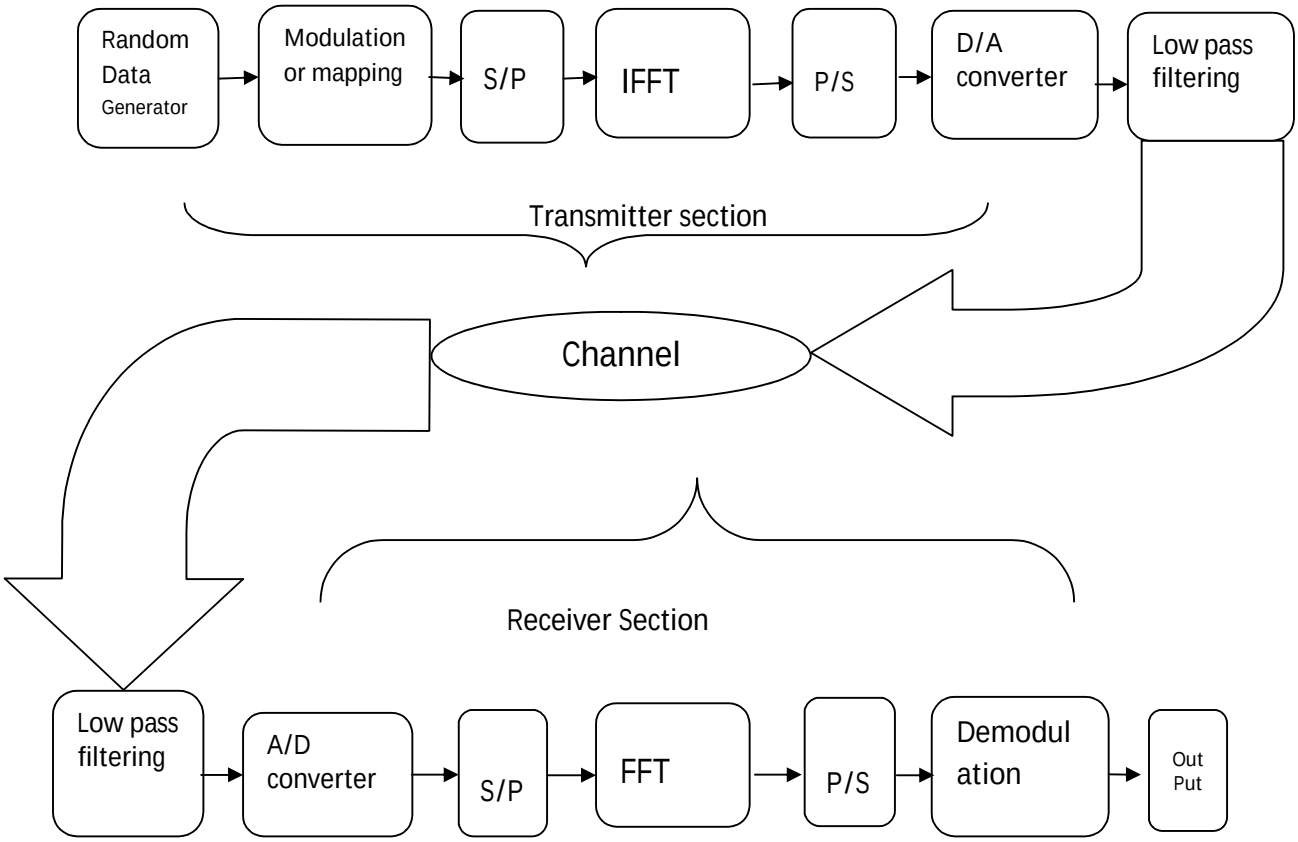


Figure-2.5: OFDM communication system Block Diagram

In OFDM system, the carriers are sinusoidal. Two periodic sinusoidal signals are called orthogonal when their integral product is equal to zero over a single period. Each orthogonal subcarrier has an integer number of cycles in a single period of OFDM system. To avoid inter channel interference these zero carriers are used as a guard band in this system.

The first step of the OFDM transmitter is to split up the data stream into K parallel sub-streams, and is modulated on its own subcarrier at frequency f_k in the complex baseband. Actually, we do not modulate the data stream, rather then we are just mapping the symbols for further operation. Different types of Digital modulation are available to do this as for example Amplitude shift keying (ASK), frequency shift keying (FSK), Phase Shift Keying (PSK), and Quadrature Amplitude modulation (QAM) etc. The data is being modulated depending on their size and on the basis of different modulation scheme like BPSK, QPSK, 16 QAM and 64 QAM. The

modulation has done on the basis of incoming bits by dividing among the groups of *i.* That is why there are 2^{I} points. The total number of bits represented according to constellation mapped of different modulation techniques. The size of *i* for BPSK, QPSK, 16QAM and 64 QAM is 1, 2, 4, and 16 respectively

The next block of the Figure 2.5 in the OFDM transmitter is IFFT part which is the major part of the OFDM transmitter. In OFDM Transmitter, IFFT (Inverse Fast Fourier Transform) used to create OFDM waveform or symbol with the help of modulated data streams. IFFT converts the data streams of frequency domain into time domain in the discrete form. Then again convert the signal parallel into serial.

Before transmit the OFDM signal in the wireless media, the signal must be converted into analog signal and pass it through the low pass filter. Then the OFDM signals transmit by the antenna to the multipath channel.

On the other hand in OFDM receiver end the FFT (Fast Fourier Transform)used to demodulate the data streams as time domain into frequency domain in the discrete from and take just the opposite operation step by step as well.

In wireless transmission the transmitted signals might be distort by the effect of echo signals due to presence of multipath delay for multichannel propagation which is called Inter-symbol Interference (ISI). To avoid Inter-symbol Interference (ISI) the Cyclic Prefix (CP) is inserted in OFDM system before each transmitted symbol. After performing Inverse Fast Fourier Transform (IFFT) the CP will be added with each OFDM symbol. The ISI is totally eliminated by the design when the CP length is greater than multipath delay. The time duration of the OFDM symbol is defined as symbol time, Tb. A copy of symbol period, Tg which is termed of Cyclic Prefix (CP) used to collect multipath where maintaining the orthogonality of the codes.

2.1.6 Fast Fourier Transform (FFT) and Inverse Fast Fourier Transform (IFFT)

The OFDM symbol threats the source symbols to perform frequency-domain into time domain. If we chose the N number of subcarriers for the system the basic function of IFFT receives the N number of sinusoidal and N symbols at a time. The output of IFFT is the total N sinusoidal signals and makes a single OFDM symbol.

The mathematical model of OFDM symbol defined by Inverse Discrete Fourier Transform IDFT which would be transmitted as given bellow:

$$x(n) = \frac{1}{N}\sum_{k=0}^{N-1} X(K)e^{j2\pi k/N}$$

x (n) represents the IDFT time domain output at the *n-th* spectral point where n ranges from *0 to N-1*. The *N* represents the number of sample points in the IDFT data frame. The quantity *X(k)*represents the *k-th* time sample, where *k* also ranges from *0 to N-1*. In general equation, *X(k)* could be real or complex.

And the corresponding Discrete Fourier Transform (DFT) operates on sample time domain signal which is periodic. The sequence x(n) gives a sequence X(k) defined only on the interval from *0 to N-1* .The equation for DFT is:

$$X(k) = \sum_{n=0}^{N-1} x(n)e^{-j2\pi k/N}$$

The DFT equation is:

$$X(k) = \sum_{n=0}^{N-1} x(n)W_N^{nK}$$

The factor $W_N^{nK}$is defined as:

$$W_N^{nK} = e^{-j2\pi k/N}$$

It is called twiddle factor. It is mainly a sinusoidal function written in the polar form.

By analyzing the first equation we found that the computation of each point of FFT requires the following computation, (N-1) complex multiplication, (N-1) complex addition (the first term in sum involves e^{j0}=1). Thus, to compute N points in FFT require N(N-1) complex multiplication and N(N-1) complex multiplication and N(N-1) complex addition.

And as the number of FFT/IFFT point (N) increases, the number of multiplications and addition required is significant because the multiplication function requires a relatively large amount of processing time even using computer. Thus, many methods for reducing the number of multiplication have been investigated over the last 50 years. The next section discussed in detail one of the method made popular by Cooly and Tukey.[8]

The twiddle factor is the combination of sine and cosine basis functions. By taking the advantages of the symmetric and periodic of the twiddle factors as shown:

$$W_N^{r+N/2} = -W_N^r$$
$$W_N^{r+N} = W_N^r$$

The twiddle factor for N=8 is calculated as shown in Table 2.1

Table 2.1 symmetric properties of W_8^m

M	W_8^m
0	+1
1	+0.7071-j0.7071
2	-j
3	-0.7071+j0.7071
4	-1
5	-0.7071+j0.7071
6	+j
7	0.7071-j0.7071

2.1.7 Analysis FFT and IFFT algorithm for OFDM

This is a "decimation in time" FFT, because the input value to the FFT is the time wave, x(t). The time wave could be a sound wave, for example. The only reason to use FFT rather than DFT is for speed. For example if we use 1024 sample

For a DFT $O\ (N^2)$ operations are required and for FFT only $O\ (N\ Log_2\ (N))$ operations are required, where N is the number of samples.
For example 1024 samples a straight DFT requires = 1024^2 = 1048576 operations.
For the same number of samples an FFT requires only 1024* Log_2 (1024) = 10240 operations

This means that a 1024 sample FFT is 102.4 times faster than the "straight" DFT. For larger numbers of samples the speed advantage improves. For example, for 4096 samples the FFT is over 340 times faster.

The first step to convert the FFT we should know the **"Danielson-Lanczos Lemma"** (D-L Lemma). This will require long equation writing, but it's a vital component of the FFT.

Here is the Danielson-Lanczos Lemma:

It is a DFT broken up into two summations of half the size of the original. The first summation is the "even terms", E, and the second is the "odd terms", O. W is the "twiddle factor", and understanding it is another key to understanding the FFT. Consider the DFT equation $X(k) = \sum_{n=0}^{N-1} x(n) W_N^{nK} \quad (Eqn..1)$

The DFT equation 1 can be divided into first half and last half in the following manner.

$$X(k) = \sum_{n=0}^{\frac{N}{2}-1} x(n) W_N^{nK} + \sum_{n=\frac{N}{2}}^{N-1} x(n) W_N^{nK} \qquad (Eqn..2)$$

$$X(k) = \sum_{n=0}^{\frac{N}{2}-1} x(n) W_N^{nK} + \sum_{n=0}^{N/2-1} x(n+N/2) W_N^{(n+\frac{N}{2})K} \qquad (Eqn..3)$$

$$X(k) = \sum_{n=0}^{\frac{N}{2}-1} x(n) W_N^{nK} + W_N^{\frac{Nk}{2}} \sum_{n=0}^{N/2-1} x\left(n+\frac{N}{2}\right) W_N^{nK} \quad (Eqn..4)$$

$$X(k) = \sum_{n=0}^{\frac{N}{2}-1} [x(n) + (-1)^k x\left(n+\frac{N}{2}\right)] W_N^{nK} \qquad (Eqn.5)$$

Now, consider k as even and odd separately. Let K=2r(even) and k=(2r+1) (odd).

$$X(2r) = \sum_{n=0}^{\frac{N}{2}-1} [x(n) + x(n+N/2)] W_N^{2nr} (Eqn..6)$$

$$X(2r+1) = \sum_{n=0}^{\frac{N}{2}-1} \left[x(n) + x\left(n+\frac{N}{2}\right)\right] W_N^{(2r+1)n} (Eqn..7)$$

Given

$$W_N^{2nr} = W_{N/2}^{nr}$$

$$W_N^{(2r+1)n} = W_{\frac{N}{2}}^{nr} \; W_N^n$$

Equation 6 and Equation 7 can be simplified into:

$$X(2r) = \sum_{n=0}^{\frac{N}{2}-1} [x(n) + x(n+N/2)] W_{N/2}^{nr}$$

The above two equations are recognizeed as (N/2)-point DFT. The (N/2) –point DFT can be subdivided until only two points are left in each DFT. The result flowgraph also called **"Butterfly Diagram"** for this method for 8-point data shown in figure 2.6**.** The Butterfly diagram is a diagrammatic representation of an FFT algorithm where the outputs were subdivided to obtain. It is referred to as decimation -in – frequency (DIF) FFT algorithm or radix-2 algorithm.

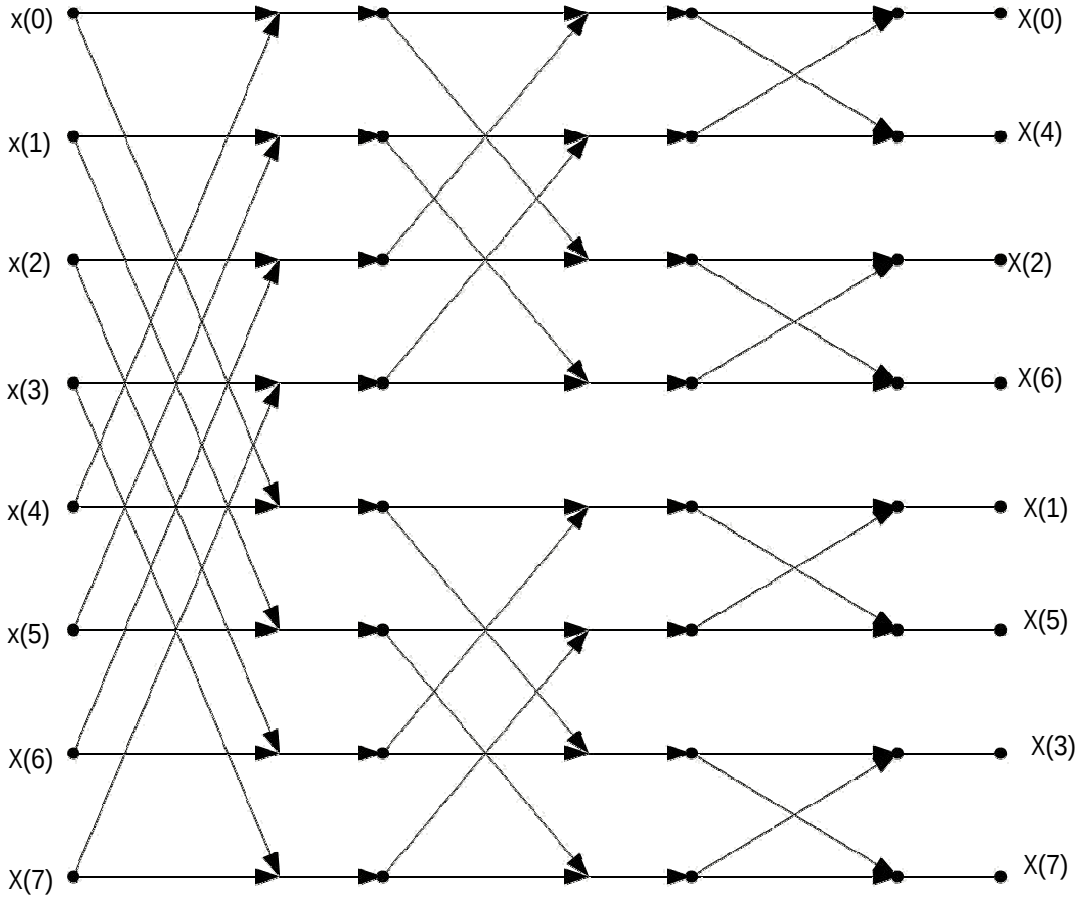


Figure 2.6: 8-Point DIF FFT flow chart

From Figure 2.6, it is clear that the final output for 8-point FFT is carried out in three stages and each stage requires 4 multiplications and hence a total of 12 complex multiplications are

required. So we can say that for N point sequence there is $\log_2 N$ stages and each stage requires $N/2$ complex multiplication hence a total of $N/2 \log_2 N$ complex multiplications are required. Number of additions required is $N \log_2 N$. Table 2.2 shows the comparison on the computation basis between the direct method and decimation-in-frequency.

Table 2.2 Computation of DFT in direct method and decimation-in-frequency algorithm

Direct Method			Decimation –in-frequency FFT	
N	Complex Multiplications $N(N-1)$	Complex Additions $N(N-1)$	Complex Multiplications $N/2 \log_2 N$	Complex Additions $N \log_2 N$
4	12	12	4	8
8	56	56	12	24
16	240	240	32	64
32	992	992	80	160

Hence, from the Table 2.2, definitely we can say that DIF FFT algorithm achieves considerable reduction in the computation of DFT

2.1.8 Pros and Cons of OFDMA

There are certain pros and cons of OFDMA which made mobile WiMAX to choose OFDMA as its multiplexing schemes. It would be clear if we just have a look description below.

Pros

1. Can fights much better against multipath with less computational complexity and more robustness than other techniques.
2. OFDMA permits portions of spectrum to be used to transmit data by different users.
3. Divides channels into narrow band flat fading sub-channels.
4. More resistant to frequency selective flat fading.
5. Easy to filter out noise.
6. Shorter and constant delay.
7. Cyclic prefix helps to eliminate Inter Symbol Interference and Inter Frequency Interference.
8. Channel equalization is simpler than single carrier channel.
9. Computational efficient as it uses FFT and IFFT.
10. It degrades its performance gracefully when the system delay reaches its highest limit.

11. Provides benefit of transmitting low power for low data rate users.
12. Narrowband interference can be reduced through spectral efficiency.
13. Suitable for coherent demodulation.
14. Accepts simultaneous low data rates from multiple users .
15. Provides different channel quality to different users depending on the requirement and condition of the channel.

Cons

1. Strong synchronization is required between users and FFT receiver.
2. For synchronization purpose, OFDMA technique uses pilot signals.
3. Very much sensitive to phase noise and frequency offset.
4. Delay in co-channel interference is more complex in OFDMA than CDMA.
5. Inefficient power consumption as FFT algorithm and FEC is constantly active.
6. If very few sub-carriers are allotted to each user, the diversity gain of OFDM and frequency Selective fading might be vanished. Much complex adaptive sub-carrier channel assignment compared to CDMA Power control system.

2.1.9 Applications of OFDM

In the beginning, the implementation of the OFDM was very complicated task, so their applications were very limited. But at present, OFDM has been adopted as the new European digital audio broadcasting (DAB) standard and for terrestrial digital video broadcasting (DVB) [16]. In fixed-wire applications, OFDM is employed in asynchronous digital subscriber line (ADSL) and high bit-rate digital subscriber line (HDSL) systems. It has been proposed for power line communications systems as well due to its resilience to dispersive channel and narrow band interference.

Beside these applications , OFDM is also used in Digital Audio Broadcasting (DAB) , Terrestrial Digital Video Broadcasting (DVB) , Magic WAND (Wireless ATM Network Demonstrator) , IEEE802.11a/HiperLAN2 and MMAC Wireless LAN etc.

2.2 Hardware Description Language (HDL)

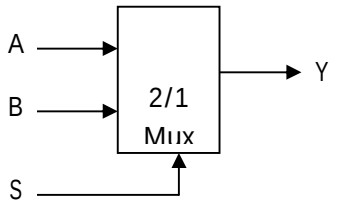
The impact of digital electronics in our daily life is so much now a days, the current period is named as digital age. Digital solutions are being observed in all areas such as telecommunications, consumer electronics, controls, data manipulations etc. To design a digital system, once the conventional approach such as hand-draw, schematic based design technique

etc. was the only choice to the designer. However the scenario has been changed. Now millions of transistors are being integrated on a single chip integrated circuit (IC) where the conventional design technique is impossible to be used. It points towards a new approach for designing today's complex digital system and that new approach is designing VLSI chip using Hardware Description Language (HDL).

2.2.1 Types of HDL Coding Style

There are three types of Verilog coding style as follows:

- **RTL Verilog Code:** Digital circuit can be represented in different ways such as gate level representation, transistor level representation etc. RTL, acronym of Register transfer Level (RTL), is also another type of representation style for digital circuit. The reason why it is named as RTL is that any complex digital system can be partitioned into different modules where each module is basically consists of registers and gates. Information is stored in the registers and specific operation is performed using the information, and then it is transferred among the registers. Hence the said representation style is known as RTL. For example, we may define an addition event: X=A+B; It means that the sum of the content of register A and register B is transferred to the storage device X. Figure 2.7 shows the example of RTL code of a 2/1 Mux.




```
module mux_21(A, B, S, Y);
input A, B, S;
output Y;
reg Y;
always @ (S or A or B) begin
        if (S= =1) Y=A;
        else Y=B;
endmodule
```

Figure 2.7: (a) Block diagram of aA 2/1 Mux (b) Example of RTL Code Showing a 2/1 Mux

- **Structural Verilog Code:** It describes the components and interconnections present in a design. Electronic Design Automation (EDA) tool compiles and synthesizes the RTL code of a design and produces the netlist of the design in the form of structural code. Table 2.3 shows the example of structural code for a circuit of a 2/1 Mux as shown in Figure 2.8.

Table 2.3 : Example of Structural Code Showing a 2/1 Mux

```
module mux (A, B, S, Out);
input A, B, S;
output Out;
wire M, N, P;
and g1 (N, B, S);
and g2 (P, M, A);
not g3 (M, S);
or g4 (Out, N, P);
endmodule
```

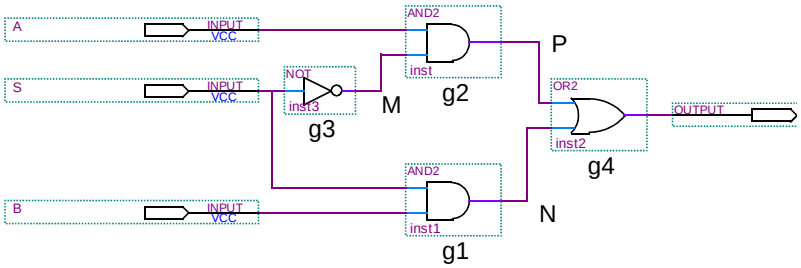


Figure 2.8: Logic diagram of a 2/1 Mux

Behavioral Verilog code: Behavioral code describes the functionality and behavior of the functional block diagram of a design. It is often used for system level modeling and simulation of a design. Table 2.4 shows an example of behavioral Verilog code (using fork and join). Usually many of the commands in behavioral code are not synthesizable (for example, wait command as shown in Table 2.4) and the code length of a design is much smaller than to write for RTL code of the same design because it is only for simulation purpose.

Table 2.4: Example of Behavioral Verilog Code

```
fork
  // start of fork block 1
  begin
    wait (y != 0);
    a = y;
  end
  // start of fork block 2
  begin
    wait (z != 0);
    b = z;
  end
join
```

2.2 Synthesis Process in Verilog HDL

Verilog HDL is a hardware description language that allows designers to model a circuit at different level of abstraction, ranging from the gate level, RTL level, and behavioral level to the algorithmic level. Synthesis process is to construct a gate-level net list from a model of a circuit described in Verilog HDL. The synthesis process is described in diagram below. [7]

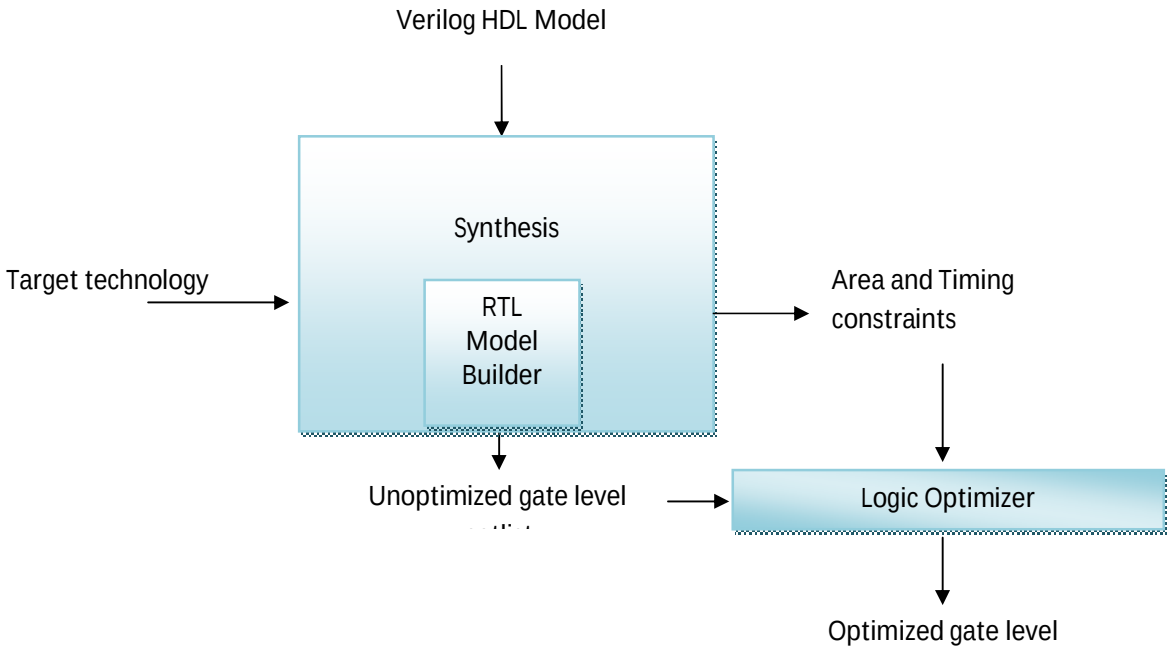


Figure 2.9: Synthesis Process in Verilog HDL Environment.

A synthesis program may alternately generate a RTL net list, which is consists of register-transfer level blocks such as flip-flops, arithmetic-logic-units, and multiplexers interconnected by wires. All these are performed by RTL module builder. This builder is to build or acquire from a library predefined components, each of the required RTL blocks in the user-specified target technology.

The above synthesis process produced an unoptimized gate level net list. A logic optimizer can used the produced net list and the constraint specified to produce an optimized gate level net list. This net list can be programmed directly into a FPGA chip.

2.3 Field Programmable Gate Arrays (FPGA)

By modern standards, a logic circuit with 20000 gates is common. In order to implement large circuit, it is convenient to use a type of chip that has a large logic capacity. A field-programmable gate arrays (FPGA) is a programmable logic device that support implementation of relatively large logic circuits. FPGA is different from other logic technologies like CPLD and SPLD because FPGA do not contain AND or OR planes. Instead, FPGA consists of logic blocks for implementing required functions. [10]

A FPGA contain 3 main types of resources: logic blocks, I/O blocks for connecting to the pins of the package, and interconnection wires and switches. The logic blocks are arranged in a two-dimensional array, and the interconnection wires are organized as horizontal and vertical routing channels between rows and columns of logic blocks. The routing channels contain wires and programmable switches that allow the logic blocks to be interconnected in many ways. FPGA can be used to implement logic circuits of more than a few hundred thousand equivalent gates in size. Equivalent gates is a way to quantify a circuit's size by assume the circuit is to be built using only simple logic gates and then estimate how many of these gates are needed.

Each logic block in a FPGA typically has a small number of inputs and one output. The FPGA products on the market feature different types of logic blocks. The most commonly used logic block is a lookup table (LUT), which contains storage cells that are used to implement a small logic function. Each cell is capable of holding a single logic value, either 0 or 1. The stored value is produced as the output of the storage cell. LUT of various sizes may be created, where the size is defined by the number of inputs.

For a logic circuit to be realized in a FPGA, each logic function in the circuit must be small enough to fit within a single logic block. In practice, a user's circuit is automatically translated into the required form by using CAD tools. When a circuit is implemented in an FPGA, the logic blocks are programmed to realize the necessary functions and the routing channels are programmed to make the required interconnections between logic blocks.

FPGA is configured by using the in-system programming (ISP) method; the FPGA can be programmed while the chip is still attached to its circuit board. The storage cells in the LUTs in an FPGA are volatile, which means that they lose their stored contents whenever the power supply for the chip is turned off. Hence the FPGA has to be programmed every time power is

applied. Of this, a small memory chip that holds its data permanently, called a programmable read-only memory (PROM) is included on the circuit board that houses the FPGA. The storage cells in the FPGA are loaded automatically from the PROM when power is applied to the chips.

2.3.1 FPGA Implementation of a Design

Design of a digital system can be realized into hardware using FPGA device. It reduces the time to market, reduces the cost and offers the design flexibility. There are many FPGA vendors such as Altera, Xilinx, Latice, Philips etc. The vendors provide the Electronic Design Automation (EDA) software for compilation, synthesis and simulation. Figure shows flow diagram of a design to be realized into FPGA hardware. Once the sub-modules of a design are identified, each of the modules is designed, compiled and synthesized using FPGA vendor provide software. Then functional simulation is performed upon each module. The correct simulation result ensures the proper functionality of a design. Once the simulation result of all the sub modules are as desired then they are integrated and simulated again. Then for hardware realization, suitable FPGA device is selected for the design, inputs and outputs are assigned to specific pins of the FPGA, It is again compiled and synthesized. After that timing simulation of the design is performed to ensure that the design functions in real time. Then the design is downloaded into the FPGA. [9]

Design Flow for FPGA Implementation

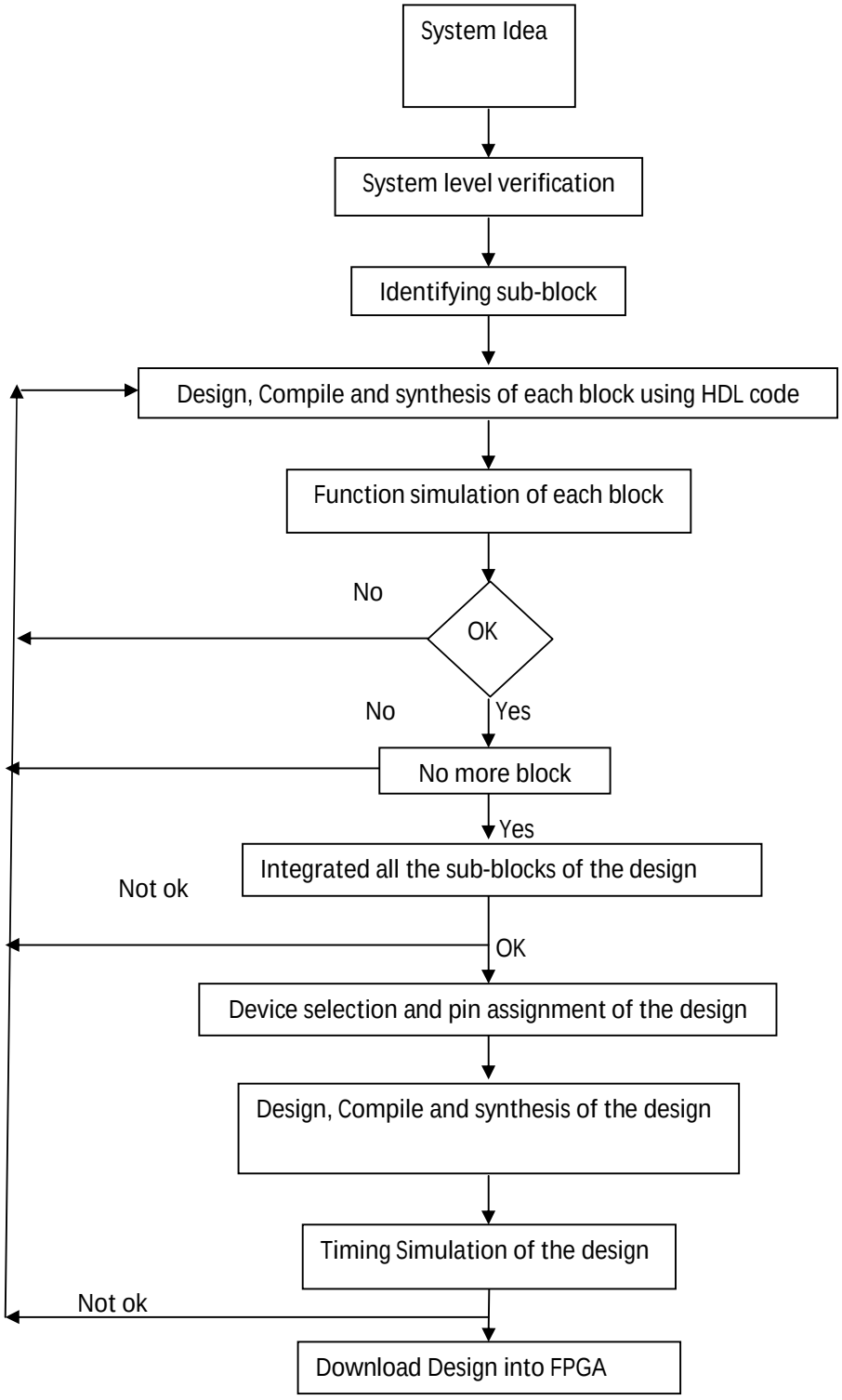


Figure 2.10: Digital system design and implementation using FPGA

Chapter 3

Design of an 8-point IFFT/FFT for OFDM Transmitter /Receiver

3.1 Introduction

This chapter focuses the technique and method for designing the core processing block in an OFDM transmitter. The computational time of FFT is faster than that of DFT because the number of multiplications and additions operation in FFT is less compared to DFT method as shown in Table 2.2. The computational time between FFT and IFFT is assumed to be same because of their operations which are almost similar except for scaling and conjugation of the twiddle factor. There are two techniques to implement the OFDM transmitter, namely structural method and direct computation method. Both methods will be discussed in the following section.

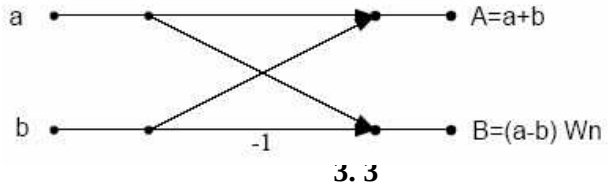
3.2 Proposed technique of an 8-point Inverse Fast Fourier Transform (IFFT) and Fast Fourier Transform (FFT)

The core processing block in an OFDM transmitter is the Inverse Fast Fourier Transform. The IFFT can be implemented using two methods, structural method or direct mathematical method.

In our total project work, we consider here only 8-point IFFT /FFT calculations for simplicity. Otherwise if want to use 16, 32, 64…..2^n (n= 4, 5, 6…) point IFFT /FFT calculations, it would increase much complexity of our project work.

3.2.1 Implementing structural method of an 8-point IFFT

Structural method implements a single butterfly computation.



3. 3

Figure 3.1 Single Butterfly Flow Chart in IFFT

The single butterfly computation is implemented in the data path unit for IFFT operation. A control unit controlling the data path and determine the stage of operations. The control unit coordinates the appropriate pairs of inputs into the butterfly computation and the output pairs are store in the memory. Each pair of random input bits will undergo multiples of butterfly computation in stage 1. Assume the input string bits are x0, x1, x2, x3, x4, x5, x6, and x7 respectively, Stage 1 computation will store its result in certain memory location, assume memory M.

At stage 2, the result in memory M is feed into butterfly computation in pairs. The control unit acts as a selector to select the correct input for the butterfly computation in every stage. The output from Stage 2 is stored in the same memory location at M.

For 8-point IFFT, the process ends at Stage 3. The output of the Stage 3 is divided by 8 and hence the final output is the computed Inverse Fast Fourier Transform.

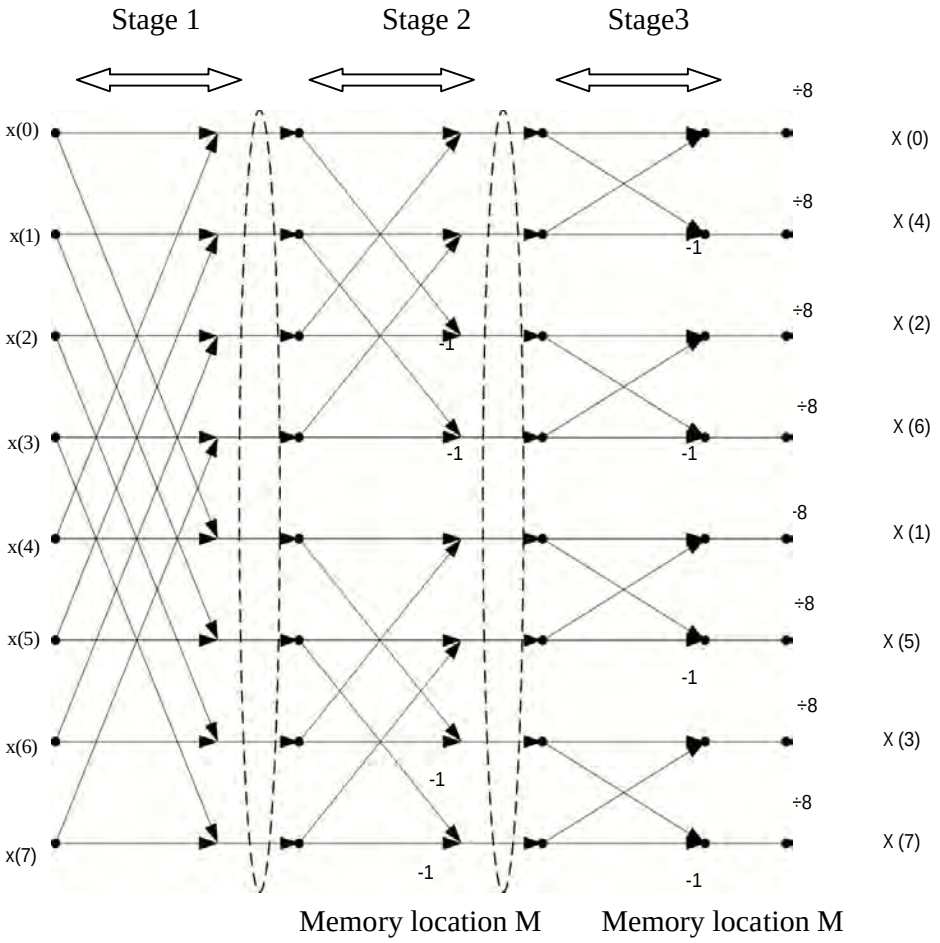


Figure 3.2 Structural Implementation of IFFT

Figure 3.2 shows that there are 3 stages in an 8-point IFFT where Stage 1 accepts the input data directly.

3.2.2 Expanding Direct Method of an 8-point IFFT

To make simplicity of the implementation we can use direct method where the final output is derived from the input directly. In a structural method, the single butterfly and summation has to be carried out 12 times for an 8-point IFFT. The multiplication and summation has to be carried out, although some twiddle factor has value of 0 or 1. This introduces redundancy in the implementation.

For example, the implementation of structural approach is $X= (0)\ a+ (1)\ b$, where in the direct mathematical approach, the implementation is simply $X=b$. Multiplication of the twiddle factor is skipped to avoid redundancy in and reduce computation time. Thus, this method is optimized.

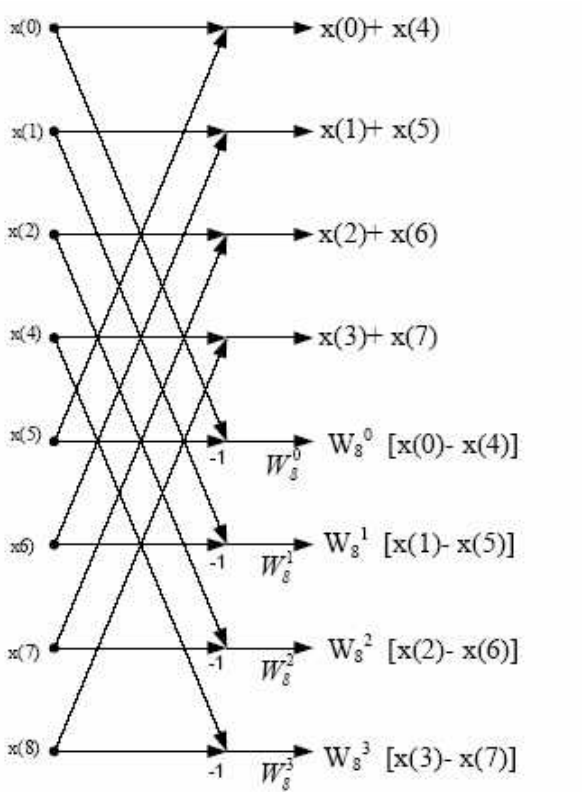


Figure 3.3 Stage 1 Computation Flow Chart of an 8-point IFFT Computation

The Figure 3.3 shows the computation in Stage 1. It is shown the even samples and odd samples are processed separately. The outputs of Stage 1 are feed as the inputs of the Stage 2. Stage 2 computation take place and this process repeats at the final stage, Stage 3.

The output of Stage 1 is connected to the input of Stage 2. The complexity of the output equations increases as the Stage number increases because twiddle factor computations are involved. The twiddle factor includes multiplication and additions operations.

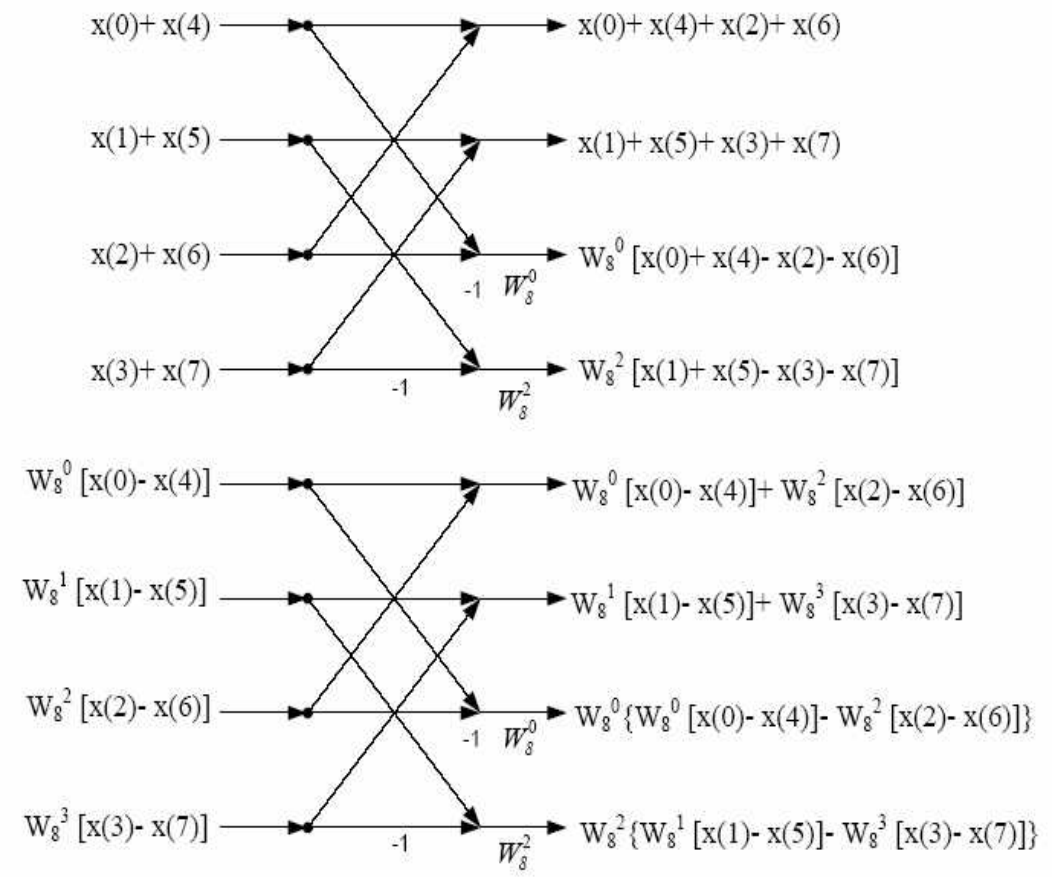


Figure 3.4 Stage 2 Computation Flow Chart of an 8-point IFFT Computation

Figure 3.4 shows the Stage 2 computation. The even inputs are grouped together and summed up in pairs. The other inputs are multiplied with their respective twiddle factor. Each of these inputs will undergo butterfly operation. Some of the outputs will have to multiply again with the twiddle factor. The outputs of the Stage 2 are fed into Stage 3. At Stage 3, the butterfly computations are repeated. The computations complexity is increased.

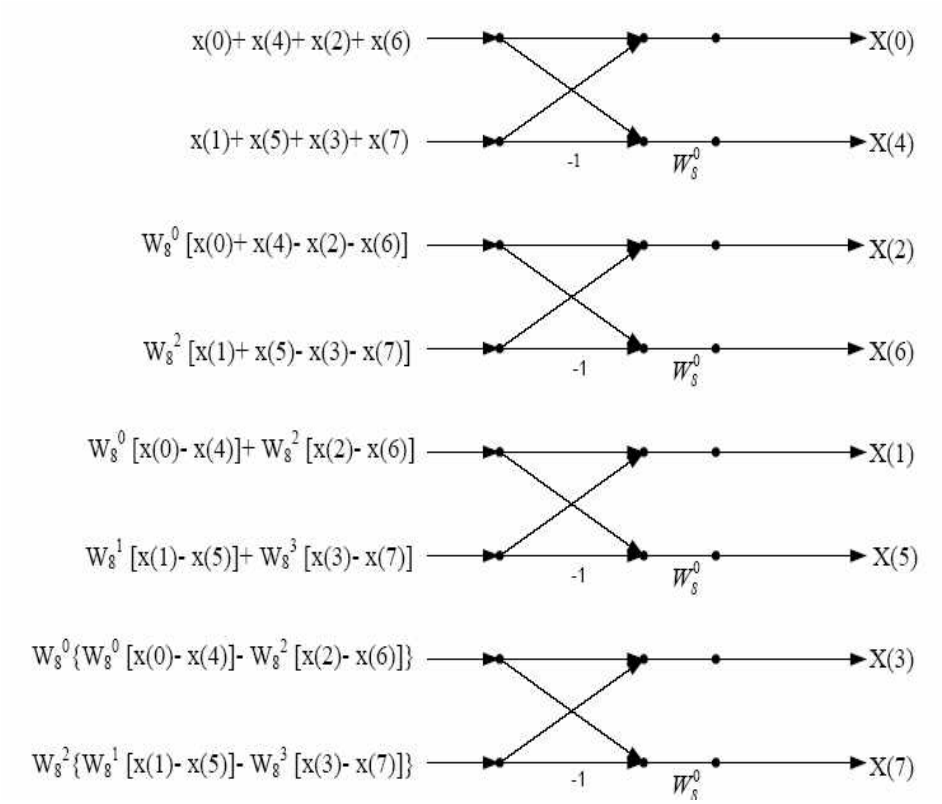


Figure 3.5: Stage 3 Computation Flow Chart of an 8-Point IFFT Computation

The final output equations derived from Figure 3.5 is shown as below (before divide 8):

$$X(0)=x(0)+ x(4)+ x(2)+ x(6)+ x(1)+ x(5)+ x(3)+ x(7)$$
$$X(4)=W_8^0\ [x(0)+ x(4)+ x(2)+ x(6)- x(1)- x(5)- x(3)- x(7)]$$
$$X(2)=W_8^0\ [x(0)+ x(4)- x(2)- x(6)]+ W_8^2\ [x(1)+ x(5)- x(3)- x(7)]$$
$$X(6)=W_8^0\ \{ W_8^0\ [x(0)+ x(4)- x(2)- x(6)]-W_8^2\ [x(1)+ x(5)- x(3)- x(7)]\}$$
$$X(1)=W_8^0\ [x(0)- x(4)]+ W_8^2\ [x(2)- x(6)]+ W_8^1\ [x(1)- x(5)]+ W_8^3\ [x(3)- x(7)]$$
$$X(5)= W_8^0\ \{W_8^0\ [x(0)- x(4)]+ W_8^2\ [x(2)- x(6)]- W_8^1\ [x(1)- x(5)]+ W_8^3\ [x(3)- x(7)]\}$$
$$X(3)= W_8^0\ \{W_8^0\ [x(0)- x(4)]- W_8^2\ [x(2)- x(6)]\}+ W_8^2\{W_8^1\ [x(1)- x(5)]- W_8^3\ [x(3)- x(7)]\}$$
$$X(7)= W_8^0\ \{W_8^0\{W_8^0\ [x(0)- x(4)]- W_8^2\ [x(2)- x(6)]\}- W_8^2\{W_8^1\ [x(1)- x(5)]- W_8^3\ [x(3)- x(7)]\}\}$$

By using Verilog HDL the final equations are implemented to make an 8-point IFFT processor which is shown Table 3.1. Using direct method the equations have been optimized for an efficient implementation. The twiddle factors are represented by 8 bit binary number. There will be slight error percentage in this design due to the approximation of the twiddle factor values.

Table 3.1 Final equations for an 8-point IFFT processor

X(0) = x(0)+ x(4)+ x(2)+ x(6)+ x(1)+ x(5)+ x(3)+ x(7)
X(4) = x(0)+ x(4)+ x(2)+ x(6)- x(1)- x(5)- x(3)- x(7)
X(2) = x(0)+ x(4)- x(2)- x(6)+ jx(1)+ jx(5)- jx(3)- jx(7)
X(6) = x(0)+ x(4)- x(2)- x(6)- jx(1)- jx(5)+ jx(3)+ jx(7)
X(1) = x(0)- x(4)+ jx(2)- jx(6)+ 0.7071x(1)+ j0.7071x(1) - 0.7071x(5)- j0.7071x(5) - 0.7071x(3)- j0.7071x(3) + 0.7071x (7) + j0.7071x (7)
X(5) = x(0)- x(4)+ jx(2)- jx(6)- 0.7071x(1)- j0.7071x(1) + 0.7071x(5)+ j0.7071x(5) + 0.7071x (3) + j0.7071x (3) - 0.7071x (7) - j0.7071x (7)
X(3) = x(0)- x(4)- jx(2)- jx(6)- 0.7071x(1)+ j0.7071x(1) + 0.7071x(5)- j0.7071x(5) + 0.7071x(3)- j0.7071x(3) - 0.7071x(7)+ j0.7071x(7)
X(7) =x(0)- x(4)- jx(2)- jx(6)+ 0.7071x(1)- j0.7071x(1) - 0.7071x(5)+ j0.7071x(5) - 0.7071x(3)+ j0.7071x(3) + 0.7071x(7)- j0.7071x(7)

3.3 Implementation of an 8-point IFFT processor

Now we implement the 8-point IFFT processor which involved few modules. All these modules are joined together to produce an 8 point IFFT processor. Figure 3.6 shows an 8 point IFFT block diagram and their interconnections.

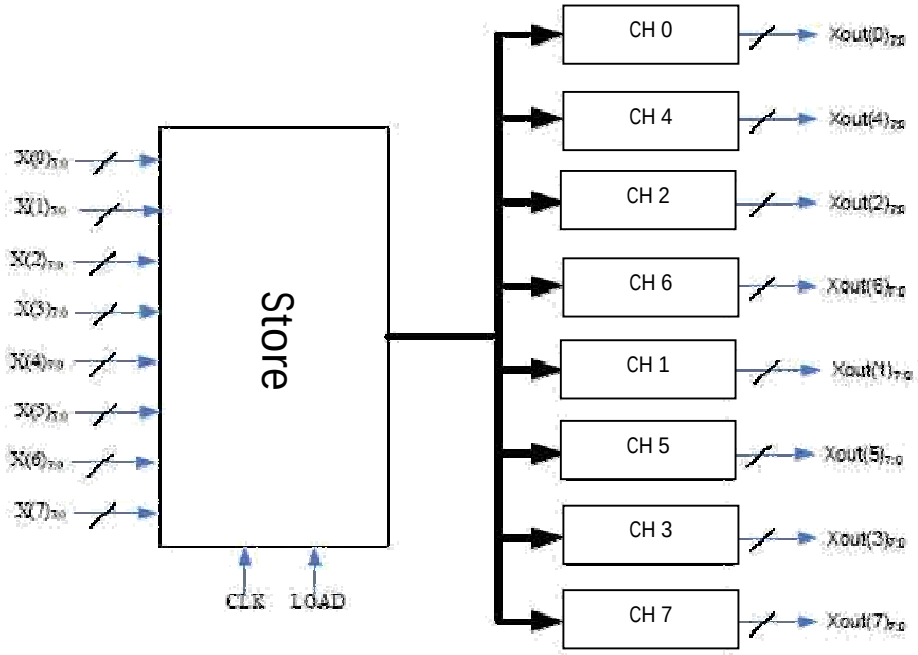


Figure 3.6 Block diagram of an 8 point IFFT processor

Figure 3.6 shows the full functional block diagram of an 8 point IFFT processor where the inputs are passed into the design to store the input data synchronously at every positive edge triggered. Then, the path module shows the arithmetic computation for each respective output.

From table 3.1, for the IFFT algorithm, the even and odd outputs are computed separately in two main groups. The odd output blocks computation is more complex compared to the even group computation. The odd output computations are represented by CH 1, CH 3, CH 5 and CH 7. The even output computations are CH 0, CH 2, CH 4 and CH 6.

From the Table 2.1 we see that, in the even outputs, the twiddle factor at the output equations has been simplified.

In the sub-modules, few digital circuitries are implemented. The most important components are adder, subtractor and unsigned divider. Multiplexers are used to approximate the decimal values to the nearest integer. They are also used to convert the summation to unsigned numbers which are connected to the divider. If the signed bit is '1', then the quotient value will be converted into unsigned number. Conversion of signed and unsigned numbers is not required for positive summation values.

The input range is from -15 to 15 to avoid overflow from occurring in Xout(0). The maximum summation (before division) value which can be supported ranges from 128 to +127. Any value which exceeds this range will contribute to overflow problem.

3.3.1 Store module of an 8-point IFFT processor

The first step of an 8-point IFFT processor is to store the input for other sub-module. This module passes the inputs to the sub-modules that do the IFFT computations. The Store module consists of 8 D type flip flop registers. The outputs of this block are 8 lines of 8 bit output which are connected to Channel 0, Channel 1, Channel 2, Channel 3, Channel 4, Channel 5, Channel 6 and Channel 7.

3.3.2 Channel 0 and Channel 4 module of an 8-point IFFT processor

The purpose of Channel 0 and Channel 4 is to compute and display the result of these computations. The outputs are Xout(0) and Xout(4) respectively. The arithmetic operation for Xout(0) is summation. The Xout(4) arithmetic involves summation, subtraction and division.

$$X(0)=x(0)+ x(4)+ x(2)+ x(6)+ x(1)+ x(5)+ x(3)+ x(7)$$
$$X(4)=x(0)+ x(4)+ x(2)+ x(6)- x(1)- x(5)- x(3)- x(7)$$

3.3.3 Channel 2 and Channel 6 module of 8 point IFFT processor

The work of Channel 2 and Channel 6 is to compute and display the result of these computations. The outputs are Xout(2) and Xout(6) respectively. The arithmetic operation for Xout(2) and Xout(6) involves real and imaginary operation. They are performed separately. The arithmetic operation involves summation, subtraction and division. The twiddle factor for this output is either j or –j which contributes to the imaginary component for this path.

X(2)=x(0)+ x(4)- x(2)- x(6)+ jx(1)+ jx(5)- jx(3)- jx(7)
X(6)=x(0)+ x(4)- x(2)- x(6)- jx(1)- jx(5)+ jx(3)+ jx(7)

3.3.4 Channel 1, Channel 3, Channel 5, Channel 7 modules of an 8 point IFFT processor

The function of Channel 1, Channel 3, Channel 5 and Channel 7 is to compute and display the result of these computations. The outputs are Xout(1), Xout(3), Xout(5) and Xout(7) respectively. The arithmetic operations for all of these modules involve real and imaginary operation. They are performed separately. The arithmetic operation involves summation, subtraction and division.

The twiddle factor for all these modules consists of a real and an imaginary value of sine 45 degree or cos 45 degree. The output of the twiddle factor is approximated to 0.7071 in our proposed design.

The output equations implemented are as below:

X(0) = x(0)+ x(4)+ x(2)+ x(6)+ x(1)+ x(5)+ x(3)+ x(7)
X(4) = x(0)+ x(4)+ x(2)+ x(6)- x(1)- x(5)- x(3)- x(7)
X(2) = x(0)+ x(4)- x(2)- x(6)+ jx(1)+ jx(5)- jx(3)- jx(7)
X(6) = x(0)+ x(4)- x(2)- x(6)- jx(1)- jx(5)+ jx(3)+ jx(7)

X(1) = x(0)- x(4)+ jx(2)- jx(6)+ 0.7071x(1)+ j0.7071x(1) - 0.7071x(5)- j0.7071x(5) - 0.7071x(3)- j0.7071x(3) + 0.7071x (7) + j0.7071x (7)
X(5) = x(0)- x(4)+ jx(2)- jx(6)- 0.7071x(1)- j0.7071x(1) + 0.7071x(5)+ j0.7071x(5)

+ 0.7071x (3) + j0.7071x (3) - 0.7071x (7) - j0.7071x (7)

X(3) = x(0)- x(4)- jx(2)- jx(6)- 0.7071x(1)+ j0.7071x(1) + 0.7071x(5)- j0.7071x(5) + 0.7071x(3)- j0.7071x(3) - 0.7071x(7)+ j0.7071x(7)

X(7) =x(0)- x(4)- jx(2)- jx(6)+ 0.7071x(1)- j0.7071x(1) - 0.7071x(5)+ j0.7071x(5) - 0.7071x(3)+ j0.7071x(3) + 0.7071x(7)- j0.7071x(7)

3.4 Implementation of an of an 8-point Fast Fourier Transform (FFT) processor

In the preceding section, we developed the algorithm for 8-point IFFT by using direct method. Now we develop 8-point FFT processor. The equation for FFT is similar to IFFT equation except for the negative sign in the twiddle factor and the scaling factor. Thus, the algorithm developed for the IFFT in the previous section can be used for FFT algorithm development with minor modification.

The FFT equation is shown below.

$$\boldsymbol{X(k)} = \sum_{n=0}^{N-1} x(n) W_N^{nK}$$

3.4.1 Expanding direct method of an 8 point FFT

In the Figure 3.6, it is shown that there are 3 stages in an 8-point FFT. Stage 1 accepts the input data directly. The Figure 3.7 shows the computation in Stage 1.

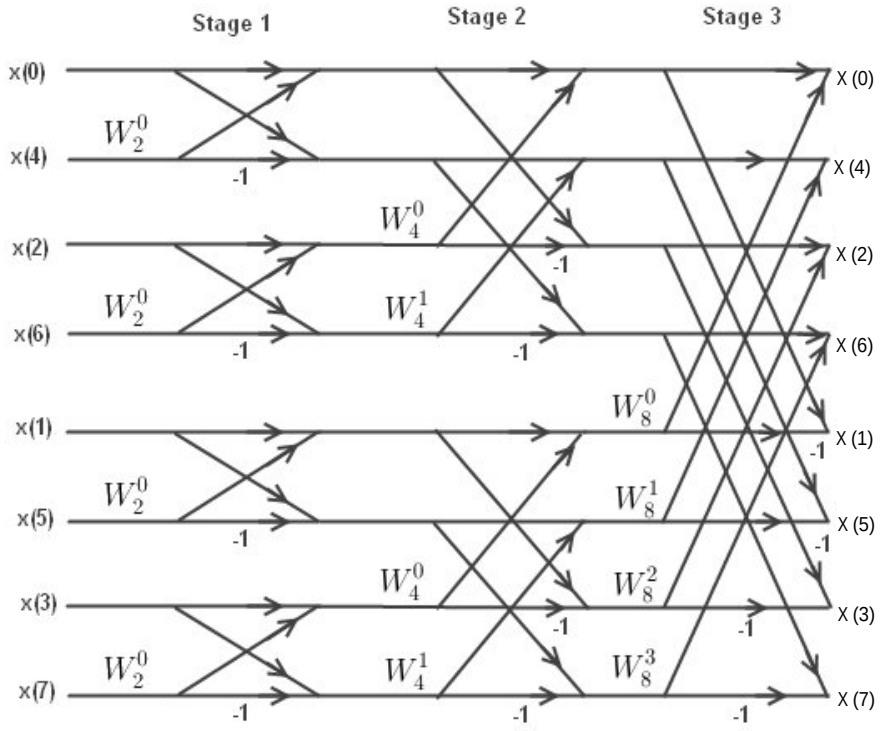


Figure 3.7 8-point FFT flow chart

Table 3.2 symmetric properties of

M	
0	+1
1	+0.7071+j0.7071
2	+j
3	-0.7071-j0.7071
4	+1
5	-0.7071-j0.7071
6	-j
7	0.7071+j0.7071

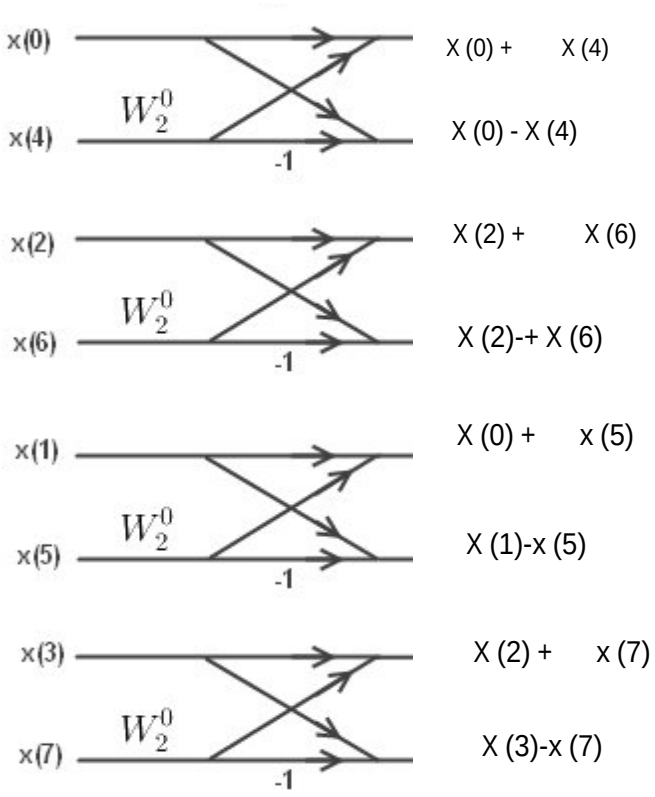


Figure 3.8: Stage 1 computation flow chart of an 8-point FFT Computation.

Figure 3.8 shows the even samples and odd samples which are processed separately. The outputs of Stage 1 are feed as the inputs of the Stage 2. Stage 2 computation take place and this process repeats at the final stage, Stage 3.

The output of Stage 1 is connected to the input of Stage 2. The complexity of the output equations increases as the Stage number increases because twiddle factor computations are involved. The twiddle factor includes multiplication and additions operations.

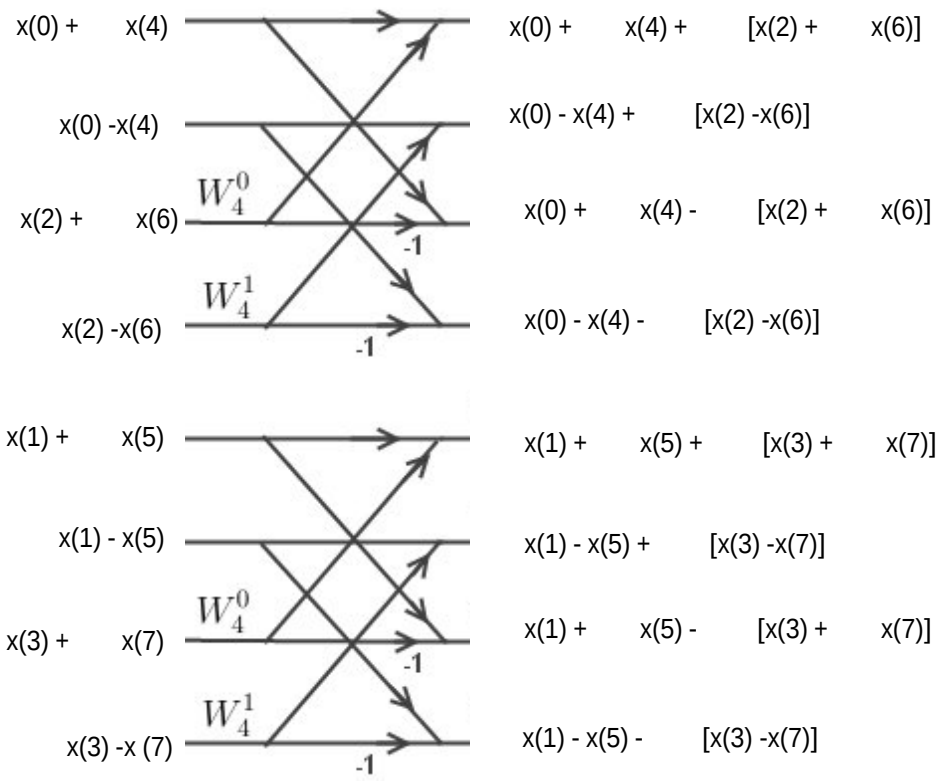


Figure 3.9 Stage 2 Computation Flow Chart of an 8-point FFT Computation

Figure 3.9 shows the Stage 2 computation. The even inputs are grouped together and summed up in pairs. The other inputs are multiplied with their respective twiddle factor. Each of these inputs will undergo butterfly operation. Some of the output will have to multiply again with the twiddle factor. The outputs of the Stage 2 are fed into Stage 3.

x(0) + x(4) + {x(2) + x(6)}

x(0) - x(4) + [x(2) -x(6)]

x(0) + x(4) - [x(2) + x(6)]

x(0) - x(4) - [x(2) -x(6)]

x(1) + x(5) + {x(3) + x(7)}

x(1) - x(5) + [x(3) -x(7)]

x(1) + x(5) - [x(3) + x(7)]

x(1) - x(5) - {x(3) -x(7)}

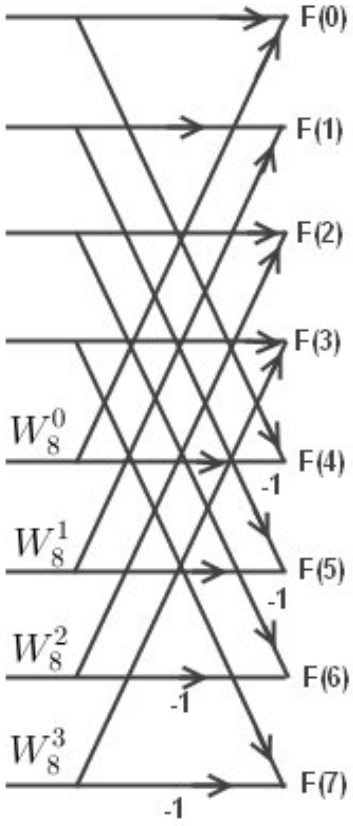


Figure 3.10 Stage 3 computation flow chart of an 8-point FFT Computation

At Stage 3, the butterfly computations are repeated and the computations complexity is increased. Therefore from Figure 3.10 which is the stage 3 computation, we get the final output.

The final output equations derived from Figure 3.10 is shown as below:

X(0) = x(0) + x(4) + [x(2) + x(6)] + [x(1) + x(5) + {x(3) + x(7)}]

X(4) = x(0) - x(4) + [x(2) -x(6)] + [x(1) - x(5) + {x(3) -x(7)}]

X(2) = x(0) + x(4) - [x(2) + x(6)] + [x(1) + x(5) - {x(3) + x(7)}]

X(6) = x(0) - x(4) - [x(2) -x(6)] + [x(1) - x(5) - {x(3) -x(7)}]

X(1) = x(0) + x(4) + [x(2) + x(6)] - [x(1) + x(5) + {x(3) + x(7)}]

X(5) = x(0) - x(4) + [x(2) -x(6)] - [x(1) - x(5) + {x(3) -x(7)}]

X(3) = x(0) + x(4) - [x(2) + x(6)] - [x(1) + x(5) - {x(3) + x(7)}]

Table 3.1 shows the final equations that are implemented using Verilog HDL to produce an 8-point IFFT processor. The equations have been optimized for an efficient implementation. The twiddle factors are represented b 8 bit binary number. There will be slight error percentage in this design due to the approximation of the twiddle factor values.

Table 3.3 Final equations for an 8-point FFT processor

X(0) = x(0) + x(4) + x(2) + x(6) + x(1) + x(5) + x(3) + x(7)
X(4) = x(0) - x(4) -j[x(2) -x(6)] +[0.7071-0.7071j] [x(1) - x(5) - j {x(3) -x(7)}]
X(2) = x(0) + x(4) - x(2) - x(6) - j [x(1) + x(5) - x(3) - x(7)]
X(6) = x(0) - x(4) +j [x(2) -x(6)] + [0.7071-j0.7071][x(1) - x(5) +j {x(3) -x(7)}]
X(1) = x(0) + x(4) + x(2) + x(6) –[x(1) + x(5) + x(3) + x(7)]
X(5) = x(0) - x(4) -j[x(2) -x(6)] - [0.7071-0.7071j] [x(1) - x(5) - j {x(3) -x(7)}]
X(3) = x(0) + x(4) - x(2) - x(6) +j [x(1) + x(5) - x(3) - x(7)]
X(7) = x(0) - x(4) +j [x(2) -x(6)] - [0.7071-j0.7071][x(1) - x(5) +j {x(3) -x(7)}]

Chapter 4

Result of Verilog HDL Simulation

4.1 Introduction

In the previous chapter we have designed the core processing block of IFFT and FFT for an OFDM transmitter and receiver. We showed there are two techniques to implement the design and we use direct method of an 8-pont IFFT and FFT for simplicity.

This chapter discusses the results and simulation which are carried by Altera Quartus -II simulator with various input samples. Each of the input samples contains 4-bits of input. The accuracy of the output is also verified to the output from MATLAB simulation. The result is divided into 2 different sections, for FFT processor and IFFT processor. The output from each of the modules is shown and followed to the overall output.

4.2 FFT Processor Result

In this sub section, the output of FFT processor is presented. This module is combined of eight sub-modules.

4.2.1 Store Module Simulation Result for FFT Processor

This module is to pass the input data at each positive clock edge to the different modules of FFT processor with the condition that the load signal is active high. One clock signal is required to pass the data in. The result is shown in Figure 4.1

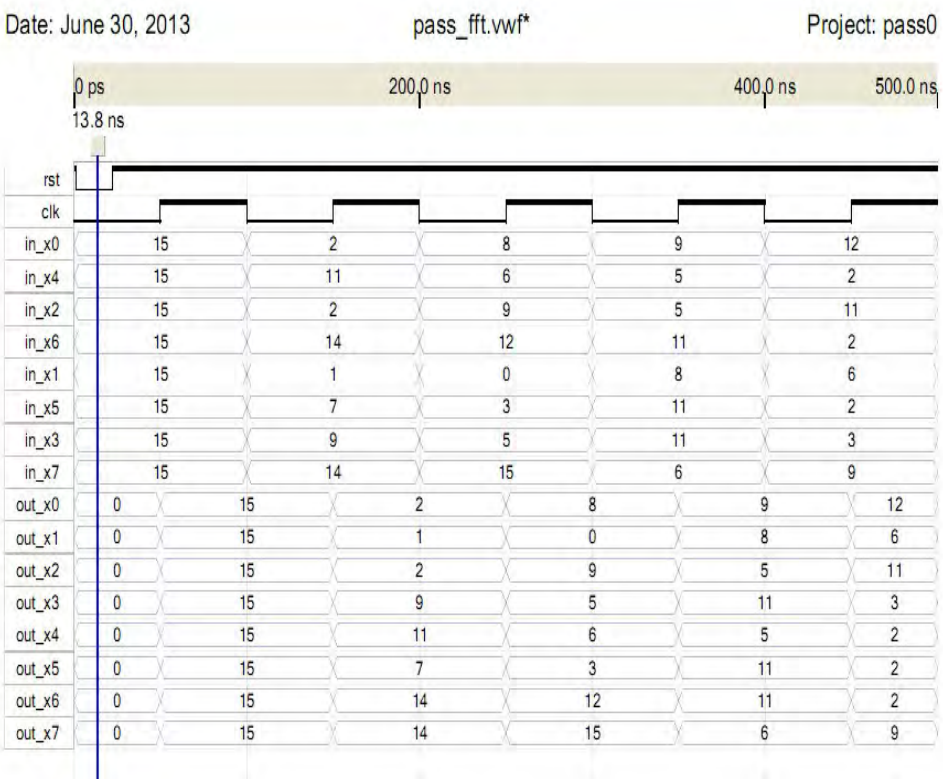


Figure 4.1 Store Module Simulation Output for FFT Processor

The above Figure 4.1 shows that each data in the corresponding input is successfully pass to the output buffer out_x0,out_x1,out_x2,out_x3,out_x4,out_x5,out_x6, and out_x7, respectively. This simulation and other rest of simulations we use five (5) different 8-input data (ASCII) which is provided for corresponding 8-point FFT operation at every positive clock.

4.2.2 Channel 0 and Channel 4 Module Simulation Result for FFT Processor

These modules implement almost the identical mathematical operation except the mathematical operators are different. The equations are shown in Chapter 3. There is no imaginary component present at the output. The result is shown in Figure 4.2 and Figure 4.3.

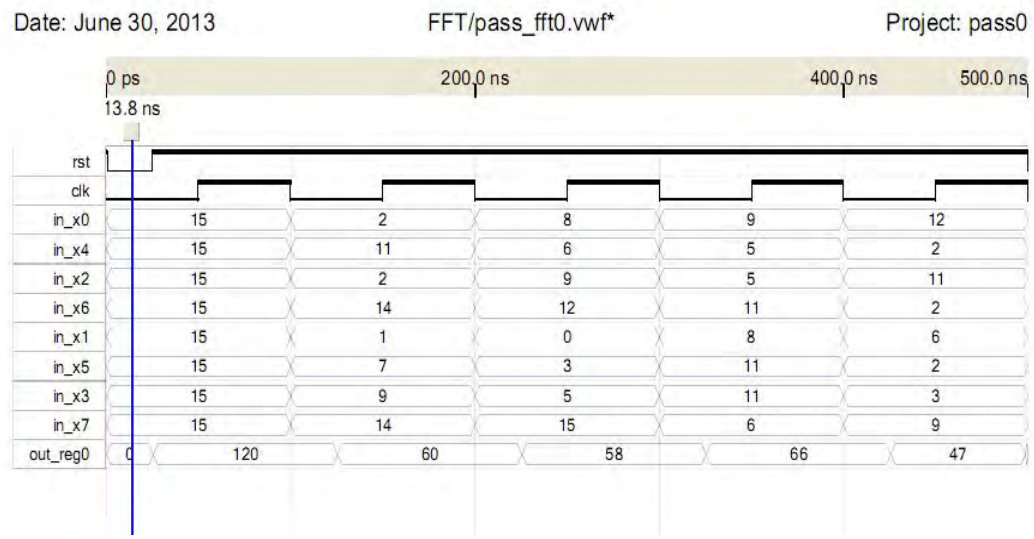


Figure 4.2: Channel 0 Module Simulation Output for FFT Processor

Figure 4.2 indicates the first output **out_reg0** by calculating eight(8) input data in the 8-point FFT operation.

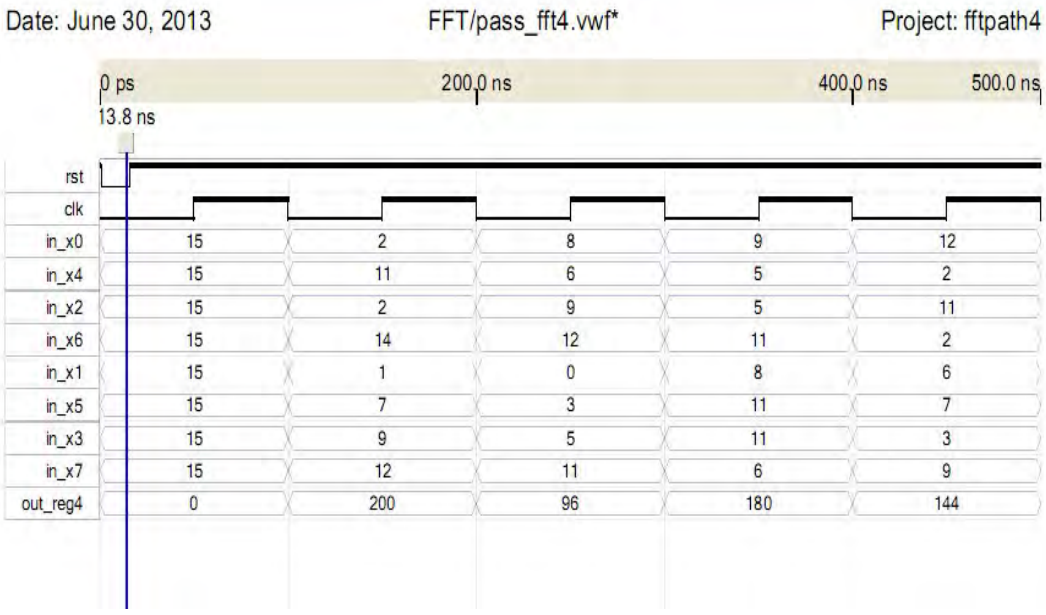


Figure 4.3: Channel 4 Module Simulation Output for FFT Processor

Figure 4.3 shows the fifth output **out_reg4** by calculating eight (8) input data in the 8-point FFT operation.

4.2.3 Channel 2 and Channel 6 Module Simulation Result for FFT Processor

These modules implement almost the identical mathematical operation except the mathematical operators are different. The equations are shown in Chapter 3. There is a imaginary component present at the output. Thus, Channel 2 and Channel 6 have more complex mathematical expressions rather than Channel 1 and Channel 4. The result is shown in Figure 4.4 and Figure 4.5

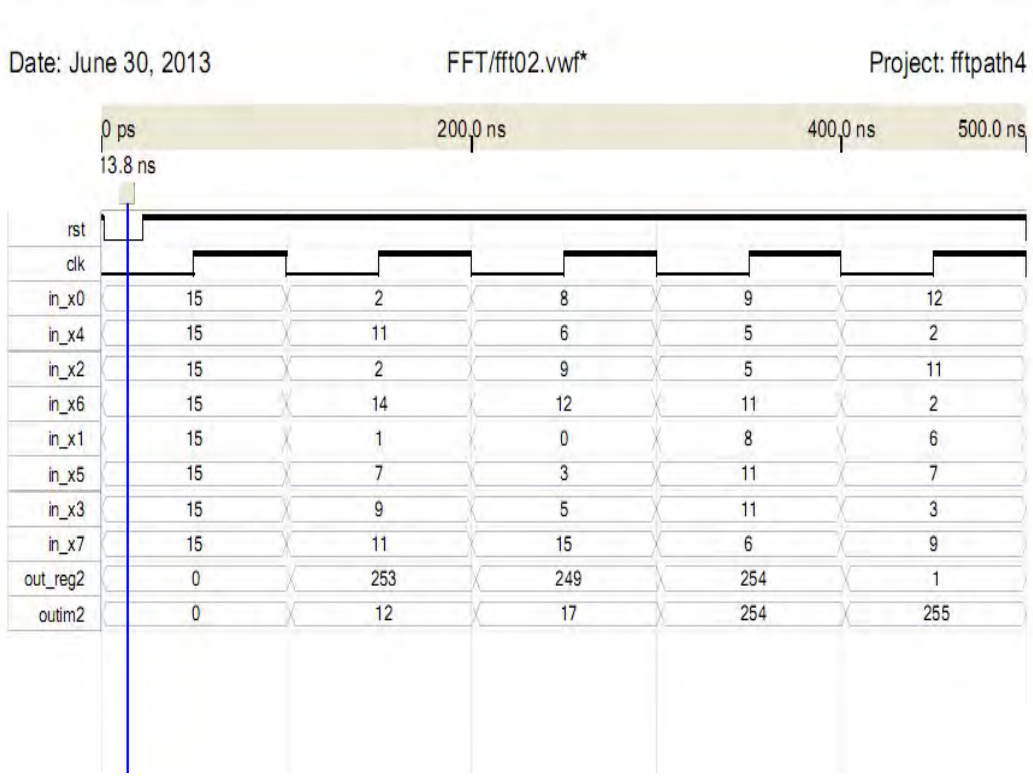


Figure 4.4: Channel 2 Module Simulation Output for FFT Processor

Figure 4.4 shows the third output which consists of one real output **out_reg2** and one imaginary output ***outim2*** by calculating eight (8) input data in the 8-point FFT operation.

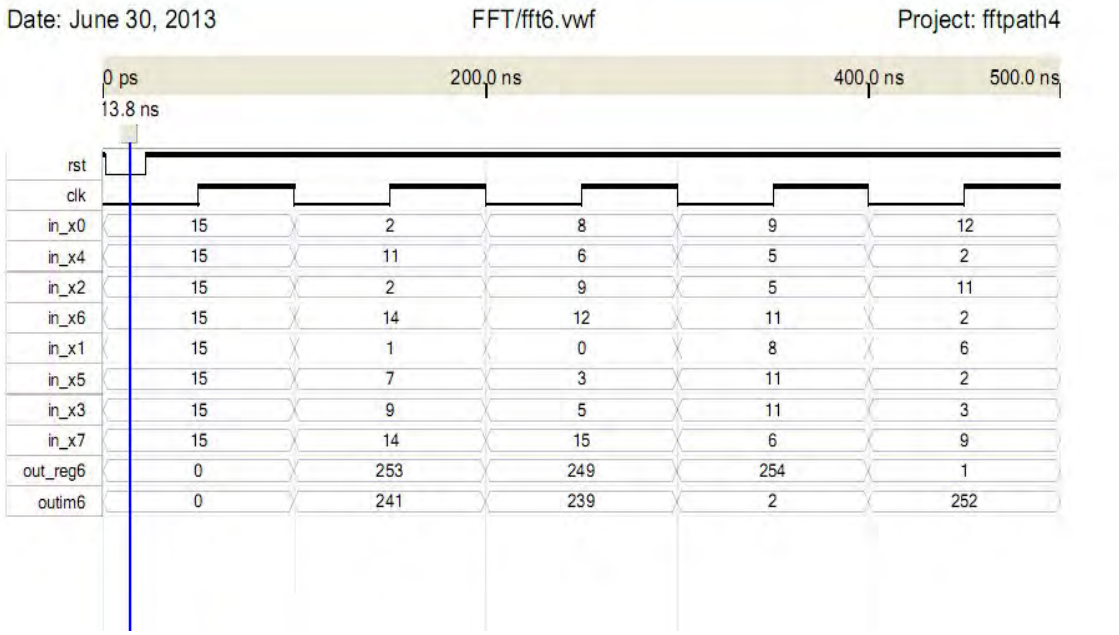


Figure 4.5: Channel 6 Module Simulation Output for FFT Processor

Figure 4.5 shows the seventh output which consists of one real output **out_reg6** and one imaginary output ***outim6*** by calculating eight (8) input data in the 8-point FFT operation.

4.2.4 Channel 1, Channel 3, Channel 5 and Channel 7 Module Simulation Result for FFT Processor

These blocks are the most complicated among all the modules in the FFT processors because it involves number of mathematical operators, like, addition, subtraction, and multiplication. The outputs contain real and imaginary components. The imaginary components are resulted from the twiddle factor which involves sin 45 degree and cos 45 degree. This value is approximated to 0.70703125 which is equivalent to 0.10110101 in binary form. The result is shown in Figure 4.6, Figure 4.7, Figure 4.8 and Figure 4.9.

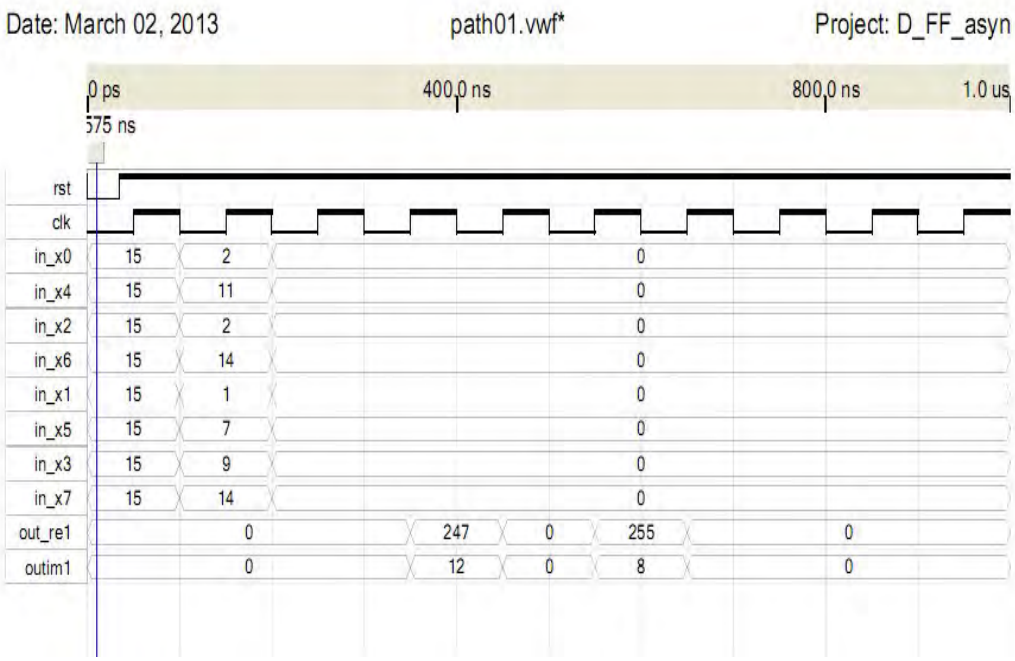


Figure 4.6: Channel 1 Module Simulation Output for FFT Processor

Figure 4.6 shows the second output which consists of one real output **out_reg1** and one imaginary output***outim1***by calculating eight (8) input data in the 8-point FFT operation.

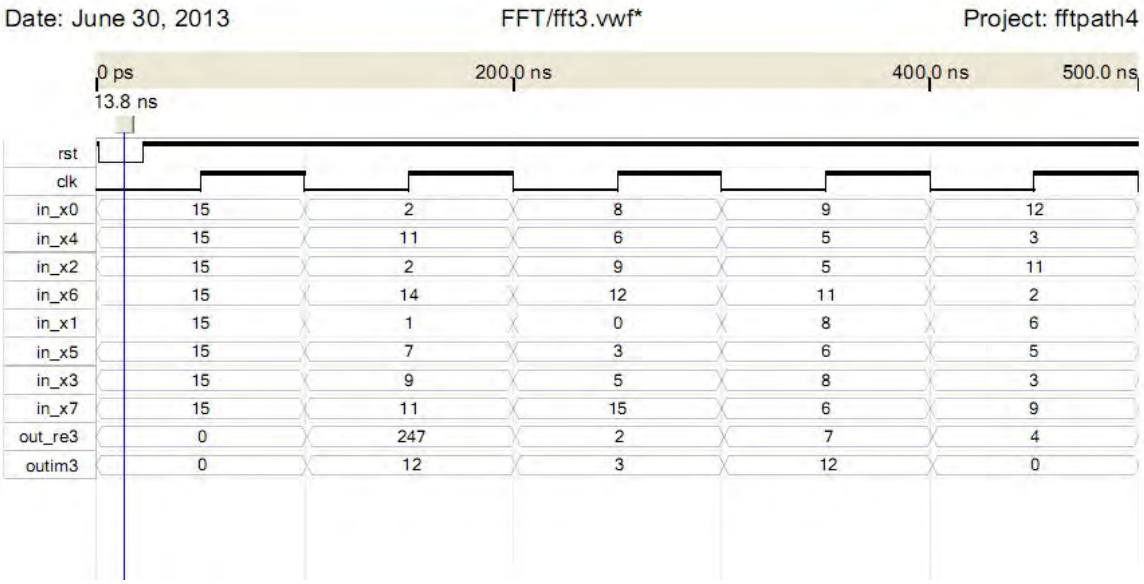


Figure 4.7: Channel 3 Module Simulation Output for FFT Processor

Figure 4.7 shows the fourth output which consists of one real output **out_reg3** and one imaginary output ***outim3*** by calculating eight (8) input data in the 8-point FFT operation.

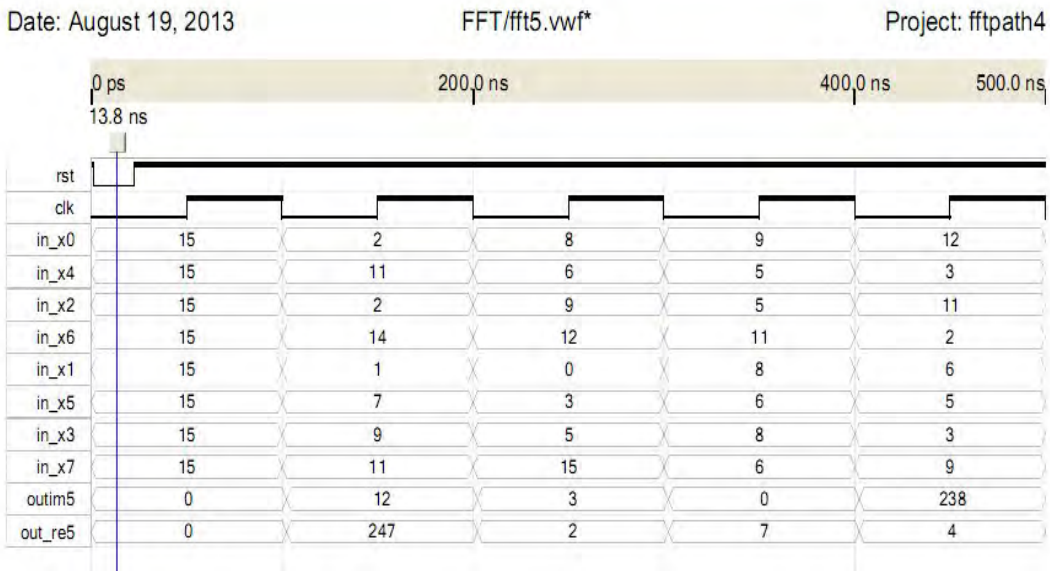


Figure 4.8: Channel 5 Module Simulation Output for FFT Processor

Figure 4.8 shows the sixth output which consists of one real output **out_reg5** and one imaginary output ***outim5*** by calculating eight (8) input data in the 8-point FFT operation.

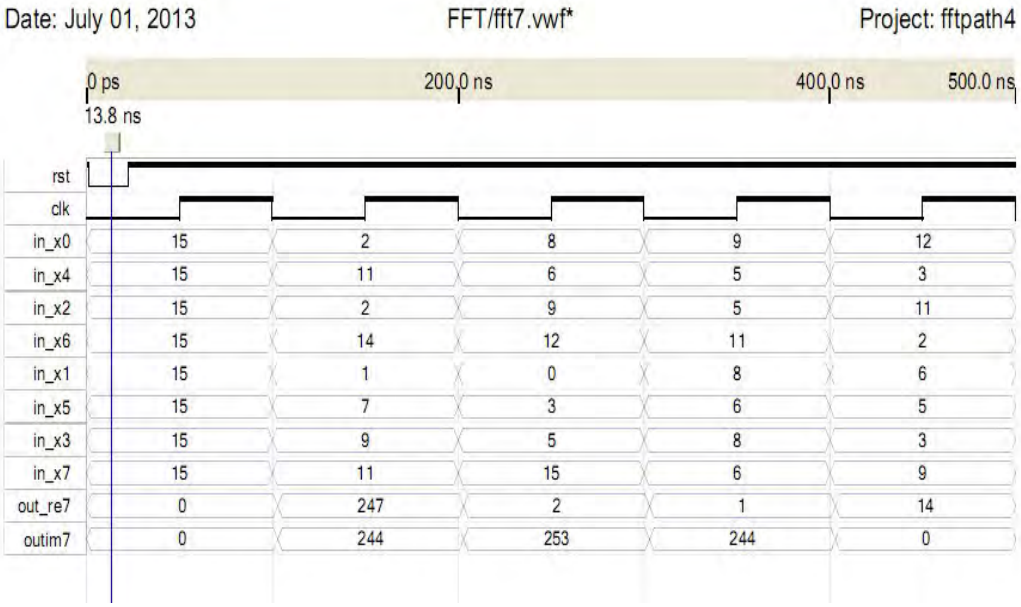


Figure 4.9: Channel 7 Module Simulation Output for FFT Processor

Figure 4.9 shows the eighth output which consists of one real output **out_reg7** and one imaginary output ***outim7*** by calculating eight (8) input data in the 8-point FFT operation.

4.2.5 8-Point FFT Simulation Result

Now the total simulation result obtained by combining all modules that has been presented previously step by step are shown here. The result is shown in Figure 4.10

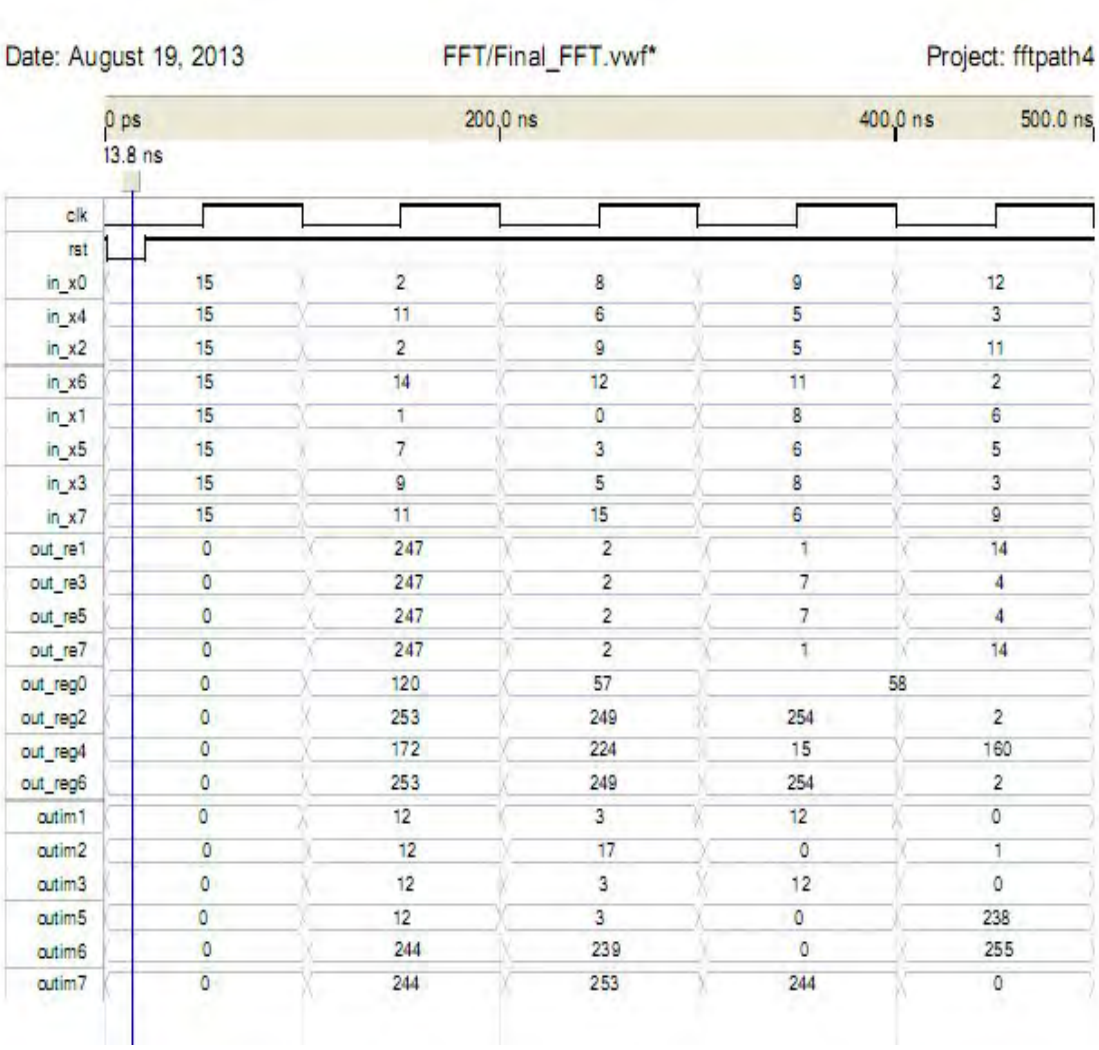


Figure 4.10: FFT Processor output

Figure 4.10 is the final simulation of the 8-point FFT processor where we see that when we give data through the eight input point it will give eight output data respectively. The simulation also shows that Channel 0 and Channel 4 give only real output and rest of paths give both real and imaginary value. So, this simulation results is very close to the mathematical calculation and further we will compare these data with MATLAB simulation.

4.3 IFFT Processor Result

Due to the scaling factor (1/N), the mathematical operation of IFFT processor is more complex than FFT. In the digital domain, this translates a division operation.

This section presents the simulation result in a similar fashion as the earlier.

4.3.1 Store Module Simulation Result for IFFT Processor

This module performs the same function as the Pass module in the FFT processor. This module is to pass the input data at each positive clock edge to the different modules of IFFT processor with the condition the load signal is active high. One clock signal is required to pass the data in. The result is shown in figure 4.11

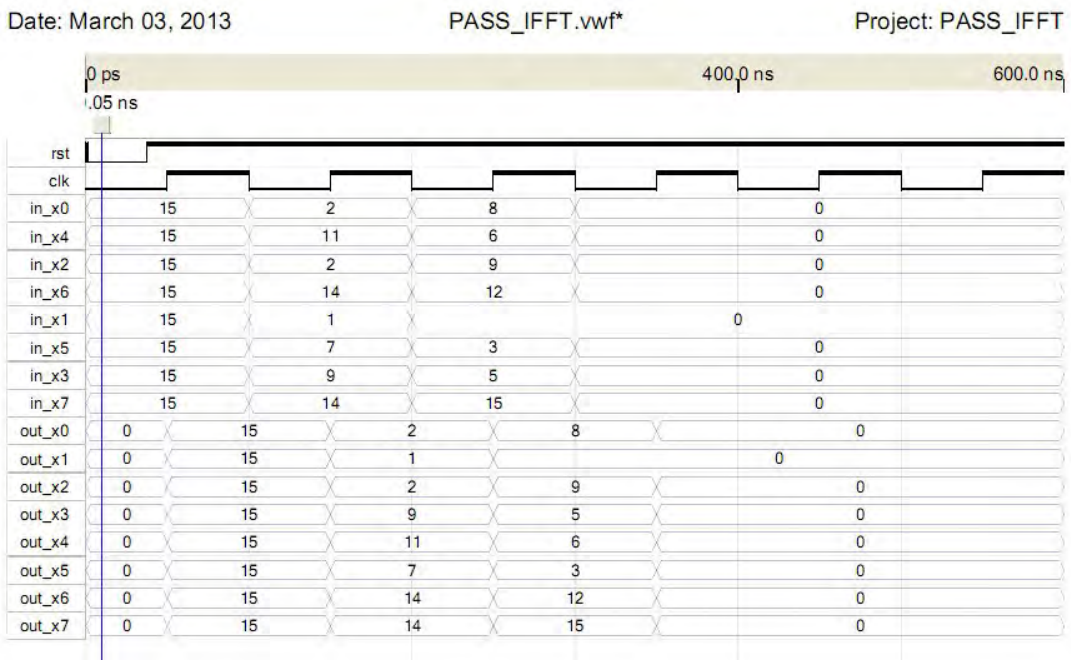
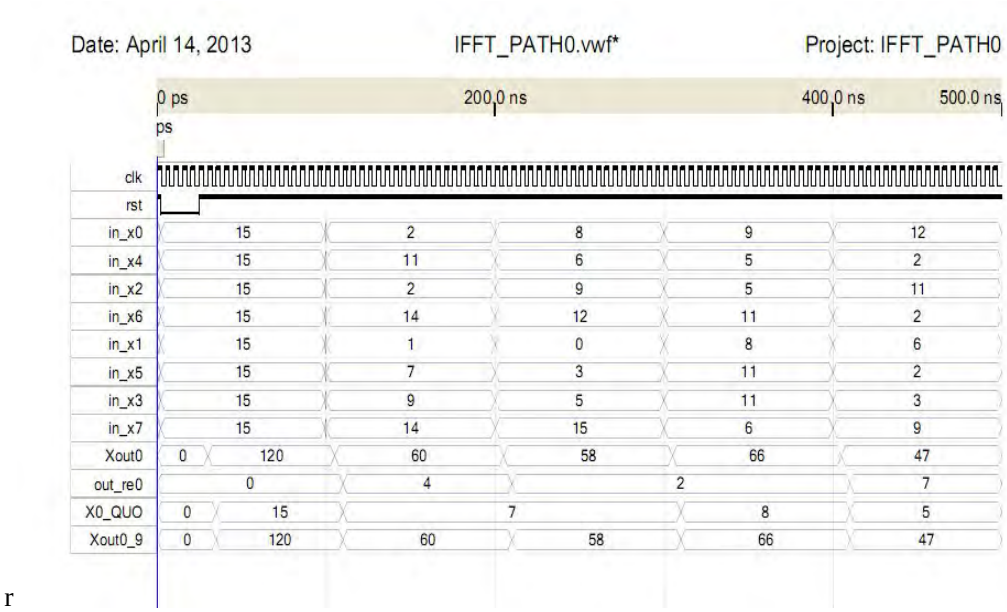


Figure 4.11 Store Module Simulation Output for IFFT Processor

Figure 4.11 shows that each data in the corresponding input is successfully pass to the output buffer out_x0,out_x1,out_x2,out_x3,out_x4,out_x5,out_x6, and out_x7, respectively. This simulation and other rest of simulations, we use five (5) different 8-input data (ASCII) which is provided for corresponding 8-point IFFT operation at every positive clock.

4.3.2 Channel 0 and Channel 4 Module Simulation Result for IFFT Processor

These modules implement almost the similar mathematical operation except the mathematical operators are different. The equations are shown in chapter 3. There is no imaginary component present at the output. The result is shown in figure 4.12 and figure 4.13.



r

Figure 4.12 Channel 0 Module Simulation Output for IFFT Processo

Figure 4.12 indicates the first output **out_re0** by calculating eight (8) input data in the 8-point IFFT operation.

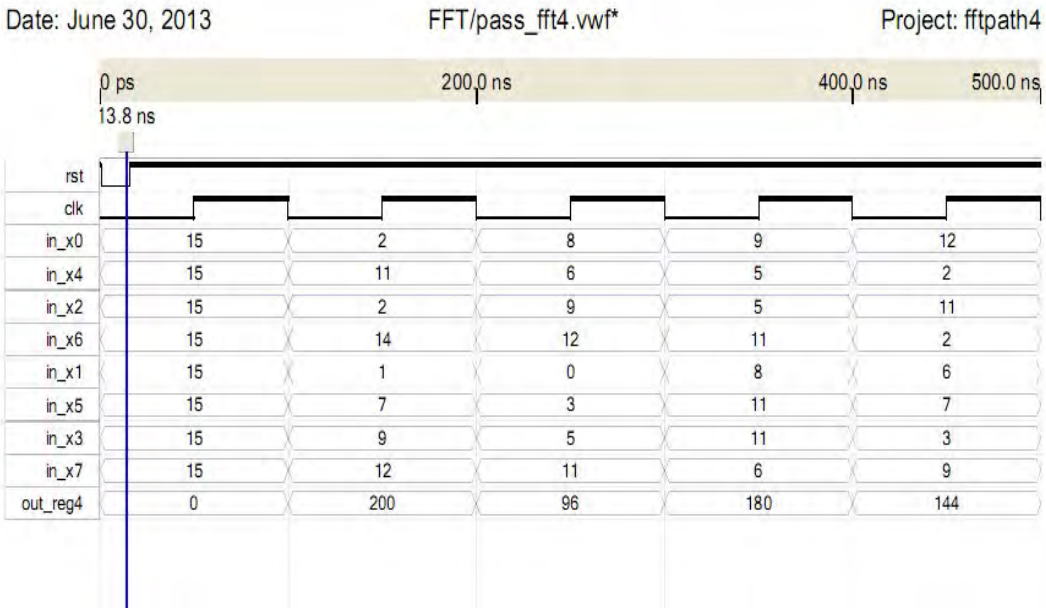


Figure 4.13 Channel 4 Module Simulation Output for IFFT Processor

Figure 4.13 indicates the fifth output **out_reg4** by calculating eight (8) input data in the 8-point IFFT operation.

4.3.3 Channel 2 and Channel 6 Module Simulation Result for IFFT Processor

These modules implement almost the identical mathematical operation except the mathematical operators are different. The equations are shown in Chapter 3. There is imaginary component present at the output. Thus, Channel 2 and Channel 6 have more complex mathematical expressions. The result is shown in Figure 4.14 and Figure 4.15

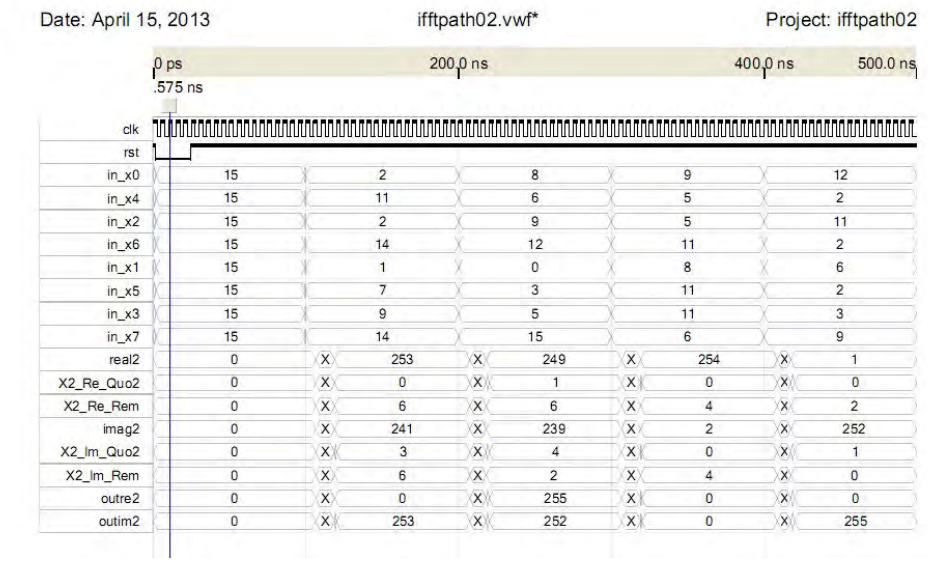


Figure 4.14 Channel 2 Module Simulation Output for IFFT Processor

Figure 4.14 shows the third output which consists of one real output **out_re2** and one imaginary output ***outim2*** by calculating eight (8) input data in the 8-point IFFT operation.

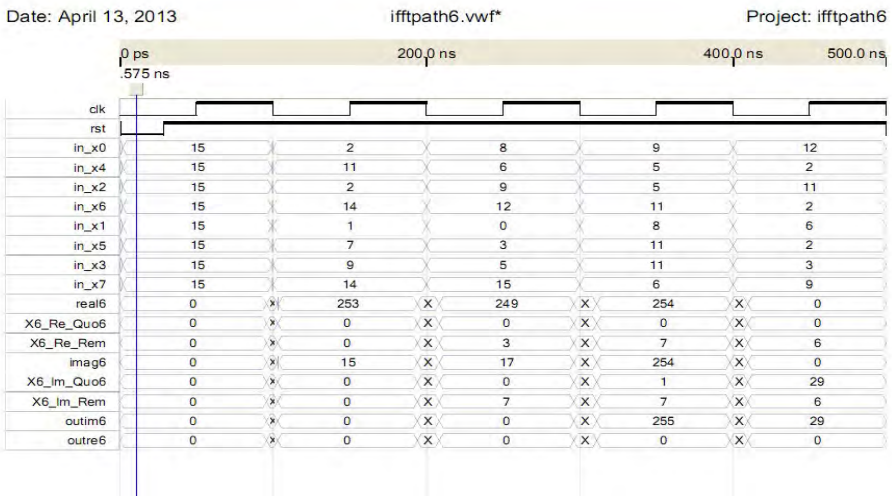


Figure 4.15 Channel 6 Module Simulation Output for IFFT Processor

Figure 4.15 shows the sixth output which consists of one real output **out_reg6** and one imaginary output ***outim6*** by calculating eight (8) input data in the 8-point IFFT operation.

4.3.4 Channel 1, Channel 3, Channel 5 and Channel 7 Module Simulation Result for IFFT Processor

These blocks are the complex among all the modules in the IFFT processor because it involves a number of mathematical operators, like, addition, subtraction, and multiplication. The outputs contain real and imaginary components. The imaginary component is resulted from the twiddle factor which involves sin 45 degree and cos 45 degree. This value is approximated to 0.70703125 which is equivalent to 0.10110101 in binary form. The result is shown in Figure 4.16, Figure 4.17, Figure 4.18 and Figure 4.19.

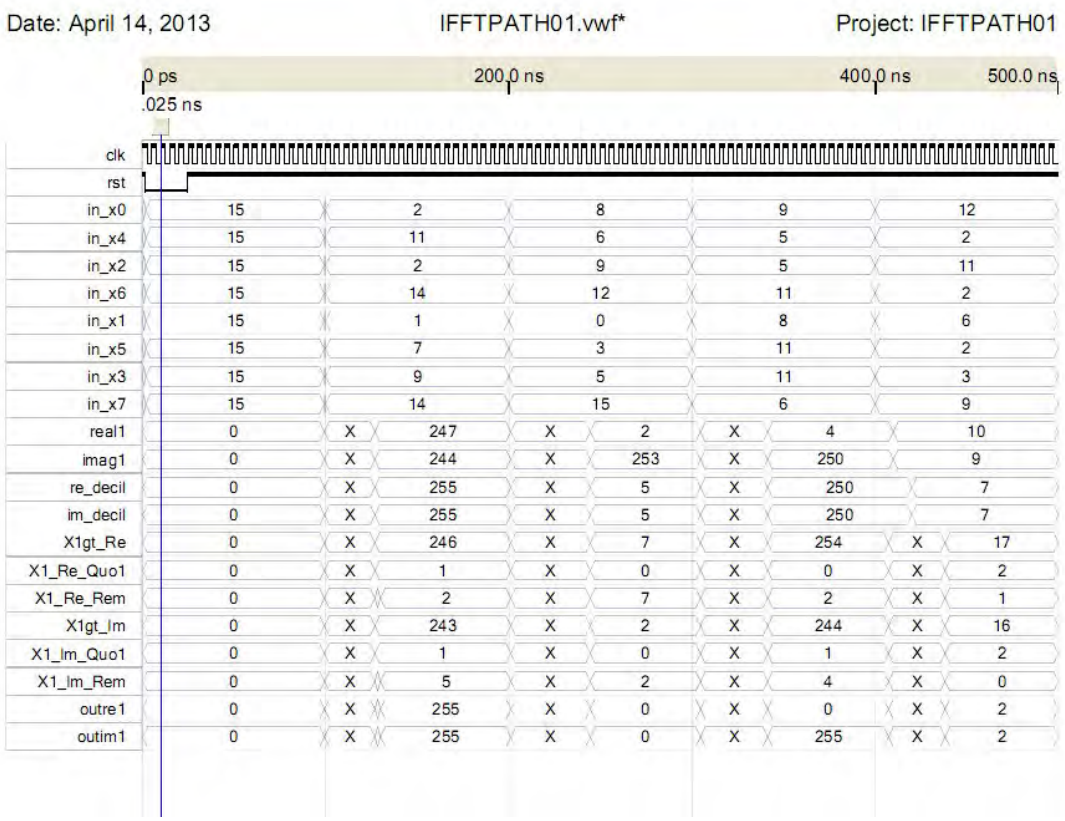


Figure 4.16 Channel 1 Module Simulation Output for IFFT Processor

Figure 4.15 shows the second output which consists of one real output **out_re1** and one imaginary output ***outim1***by calculating eight (8) input data in the 8-point IFFT operation.

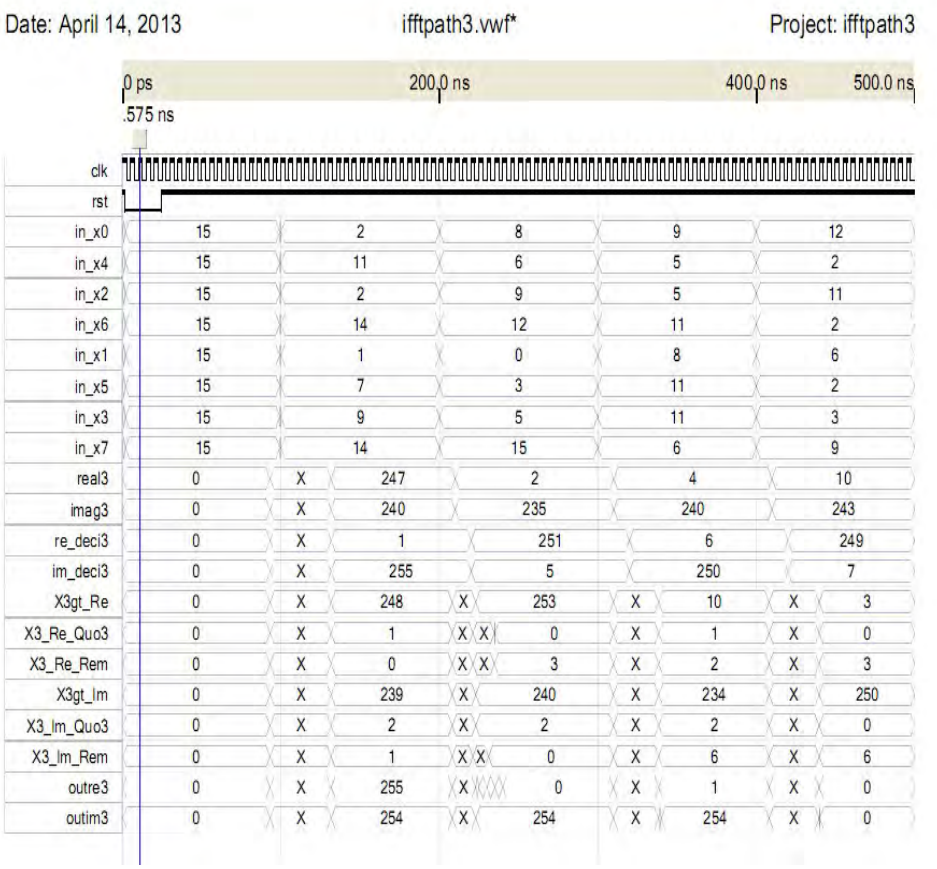


Figure 4.17 Channel 3 Module Simulation Output for IFFT Processor

Figure 4.17 shows the fourth output which consists of one real output **out_reg3** and one imaginary output ***outim3*** by calculating eight (8) input data in the 8-point IFFT operation.

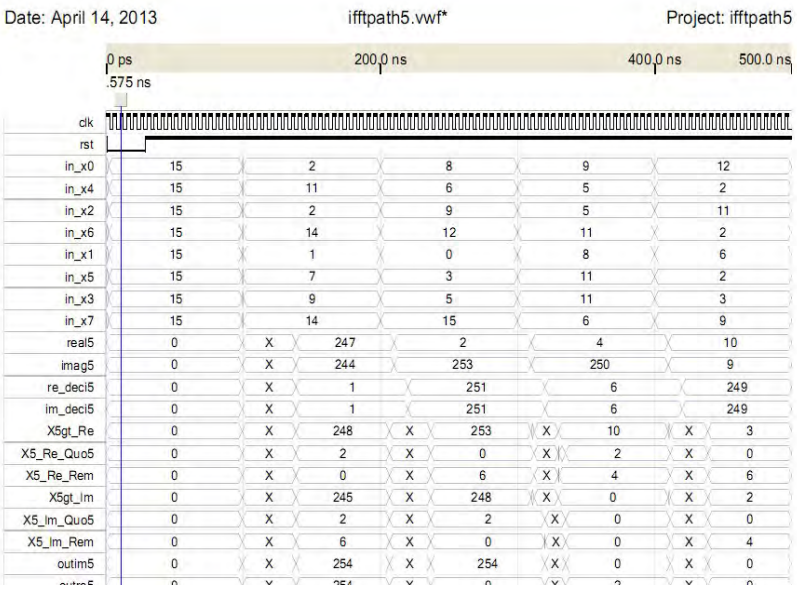


Figure 4.18 Channel 5 Module Simulation Output for IFFT Processor

Figure 4.18 shows the fourth output which consists of one real output **out_reg5** and one imaginary output ***outim5*** by calculating eight (8) input data in the 8-point IFFT operation.

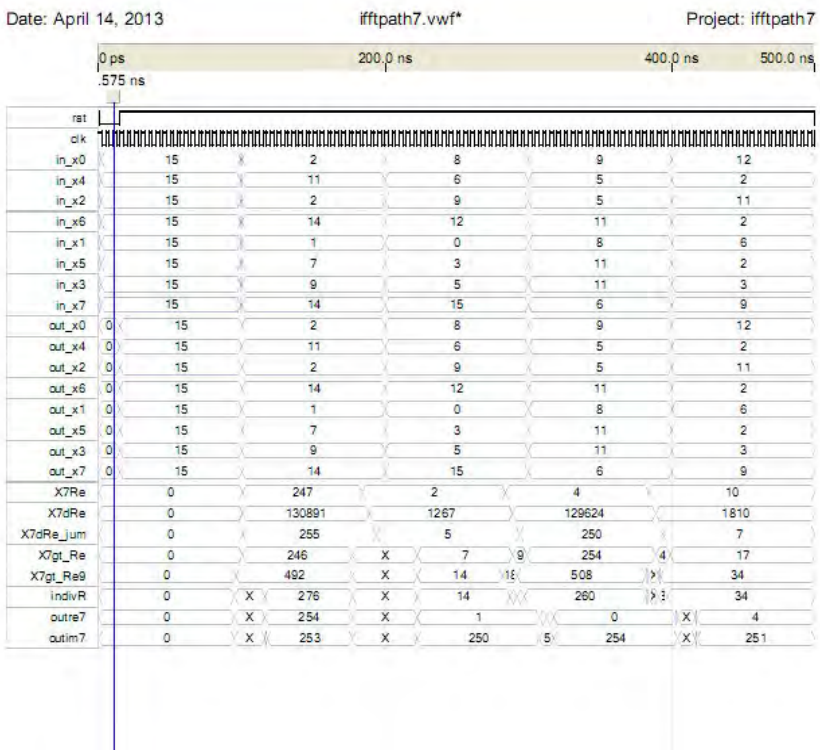


Figure 4.19 Channel 7 Module Simulation Output for IFFT Processor

Figure 4.19 shows the eighth output which consists of one real output **out_reg7**and one imaginary output ***outim7*** by calculating eight (8) input data in the 8-point IFFT operation.

4.3.5 8-Point IFFT Simulation Result

This part illustrates the overall simulation result obtained by combining all the modules at the IFFT section that has been presented earlier. The result is shown in Figure 4.20.

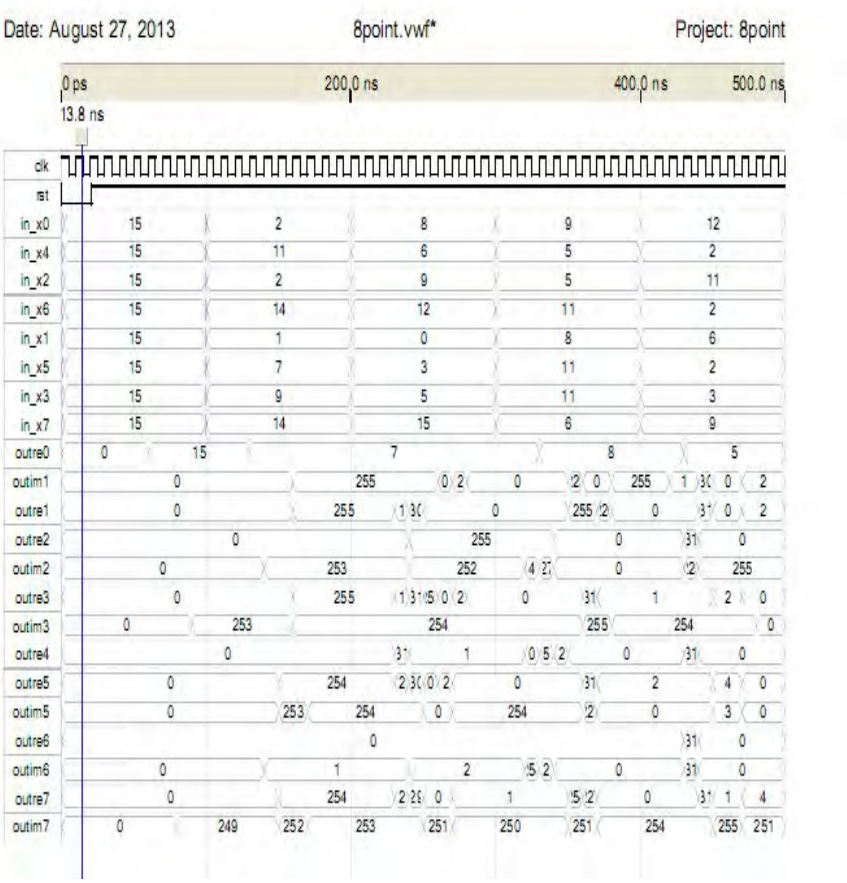


Figure 4.20: IFFT processor output

Figure 4.20 is the final simulation of the 8-pointIFFT processor where we see that when we give random data through the eight input point it will give eight output data respectively. The simulation also shows that Channel 0 and Channel 4 give only real output and rest of Channel give both real and imaginary value. So, this simulation results is very close to the mathematical calculation and further we will compare these data with MATLAB simulation.

4.4 Resources Used:

The design of the FFT and IFFT processor are compiled and simulated separately using Altera Quartus -II simulator software. The following resources are used by the module, when compiled and simulated using an Altera device of family Cyclone II, are as follows:

Resource	**Usage**
Device Family:	Cyclone II
Device:	EP2C35U484C8
Total Logic elements:	745 (out of 33216) for FFT and 1146 for IFFT
Total combinational function:	641 (out of 33216) for FFT and 967 for IFFT
Dedicated log registers:	349 for FFT and 632 for IFFT
Total registers:	349 and 632 for IFFT
Total Pin:	178 (out of 362) for FFT and 179 for IFFT
Total memory bits:	0
Embedded Multiplier 9 -bit elements	3

4.5 Verification of Verilog HDL Simulation Output

The result presented in this chapter has to be verified. The Verilog HDL output and the MATLAB simulation output using same arbitrary input number are compared to measure the accuracy of the result. As shown in Table 4.1 and Table 4.2, the accuracy of the Verilog HDL simulation output has been rounded to the nearest integer because of only 8 bit is used to represent the output value.

Table 4.1: Comparison between Quartus II Vs MATLAB output for 8-point FFT

Sl No	**Input**	**Quartus II output**		**MATLAB output**	
---	---	Real	Imaginary	Real	Imaginary
1	2	60	0	60	0
2	1	247	12	246	19
3	2	253	15	253	15
4	9	247	12	248	252
5	11	224	0	254	0
6	7	247	12	248	4.22
7	14	253	241	253	241
8	14	247	244	246	237

Table 4.2: Comparison between Quartus II Vs MATLAB output for 8-point IFFT

Sl No	**Input**	**Quartus II output**		**MATLAB output**	
---	---	Real	Imaginary	Real	Imaginary
1	2	7	0	7.50	0
2	1	255	255	255	254
3	2	255	253	255	254
4	9	255	253	255	255
5	11	253	1	255	0
6	7	254	253	255	251
7	14	0	1	0	1.875
8	14	254	249	255	2.472

Table 4.1 and Table 4.2 show the eight output values of FFT and IFFT operations obtained from Quartus-II and MATLAB simulator respectively using the same input values to the both simulators. The values are close to each other. The output of the Twiddle factor is approximated to 0.7071 in our proposed design whereas in MATLAB it is little bit more than this value. Again in our design any decimal value after summation is approximated to integer '1' when it is greater than 0.5 whereas MATLAB considers the exact value of the summation.

Chapter 5

Conclusion

5.1 Conclusion

In this modern era, OFDM (Orthogonal Frequency Division Multiplexing) is one of the best technologies which support the 3G and 4G standard of wireless communication. OFDM is mainly used in the WiMAX, LTE, WLAN etc. and providing tremendous speed. The aim of this project is the implementing the core processing blocks of an OFDM system, namely the Fast Fourier Transform (FFT) and Inverse Fast Fourier Transform (IFFT).

The core processing blocks of an OFDM system i.e. the FFT and the IFFT processors were successfully designed and implemented using common pass module and multiple path modules. Direct mathematical method was adopted as it was found to be an efficient and optimized method implementation which is based on butterfly operation instead of the structural. At the end functional simulation was performed from Altera Quartus II to generate the design net list file and translate the design into the target Altera FPGA device. The output of the functional simulation has been compared with the data calculated from pre synthesis MATLAB simulations and found that there is showing slightly error percentage due to the approximation of the twiddle factor. If we take the large value of the twiddle factor, then the percentage of error could be reduced and FFT / IFFT processor will show the better performance.

5.2 Future Work

The Author suggests the following research that could be carried out in the future.

1. In this work, FFT/IFFT block is implemented using FPGA device, in future by adding some resources like serial to parallel converter, channel coder and A/D converter it could be made complete OFDM modem.
2. The FFT/IFFT block can be implemented to the higher order like 16/32/64 point to have the better accuracy.

References:

[1] Dai,L.,Wang,Z.,and Yang, Z.,"Time-Frequency Training OFDM with High Spectral Efficiency and Reliable Performance in High Speed Environments", IEEE Journal in Communications, Vol. 30 (4), 2012.

[2] Ouarzazi, B., Berbineau, M., Dayoub, I. and Rivenq, A,M., " Channel estimation of OFDM system for high data rate communication on mobile environment" IEEE Proceedings,2009 , pp 425-429.

[3] Weidong, Li., "Studies on implementation of lower power FFT processors," Linköping Studies in Science and Technology, Thesis No. 1030, ISBN 91-7373-692-9, Linköping, Sweden, Jun. 2003.

[4] He S. and Torkelson M., "A new approach to pipeline FFT processor," In Proc. of the 10th Intern. Parallel Processing Symp. (IPPS), pp. 766– 770, Honolulu, Hawaii, USA, April 1996.

[5] Li, W., Wanhammar, L.,"Complex multiplication reduction in FFT processor," SSoCC'02, Falkenberg, Sweden, Mar. 2002.

[6] Ali, L.,Sidek, R., Aris, I.,Mohd,Ali.M. A. and Suparjo, B. S., "Challenges and Directions for IC testing", Integration, the VLSI Journal, Elsevier Science, Vol 37(1), pp. 17-28, Netherland, 2004.

[7] Brown, S. and Vranesic, "Fundamentals of Digital Logic with Verilog Design", MacGraw-Hill, Boston, USA, 2008.

[8] Cooley,J.W.,and Tukey, J.W.,"An algorithm for the machine calculation of complex Fourier series," Math. Computation, vol.19, pp. 297-301, 1965.

[9] http://www.altera.com; last access on 15 December, 2013.

[10] Cofer,R. C. and Harding, "Rapid System Prototyping with FPGAs: Accelerating the Design Process", Newres, Amsterdam, 2006.

[11] Obermaisser,R.,Kopetz,H.,and Paukovits,C., "A Cross-Domain Multiprocessor System-on-a-Chip for Embedded Real-Time Systems", IEEE Transaction on Industrial Informatics, Vol. 6(4), 2010 .

[12] Chang, R. W. ,"Synthesis of Bandlimited Orthogonal Signals for Multichannel Data Transmission," Bell System Tech. J., pp. 1775-1796, Dec, 1966.

[13] White paper, “Orthogonal Frequency Division Multiplexing (OFDM) Explained”, Magis Network, Ins. 2001.

[14] Weinstein ,S. B. and Ebert, P.M., “Data transmission by frequency division multiplexing using the discrete Fourier transform,” IEEE Transactions on Communication Technology, vol. COM-19, pp. 628-634, October 1971.

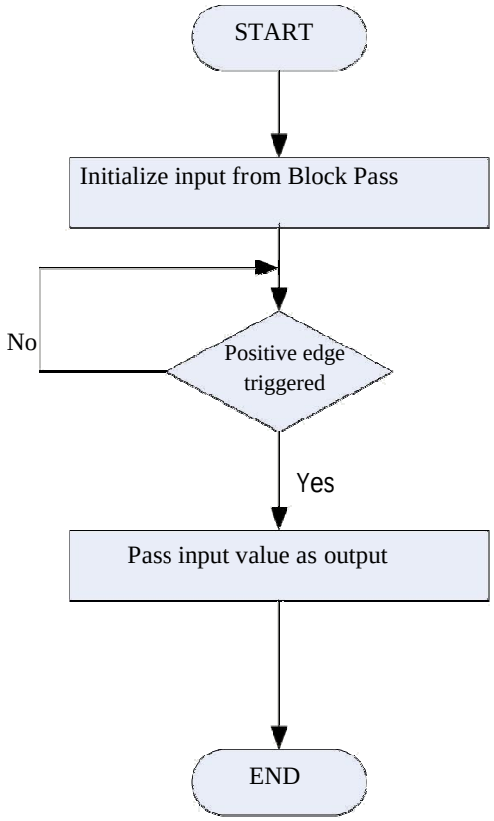
[15] Bhasker, J., “A VHDL Synthesis Primer”, 2 Edition, Star Galaxy Publishing, 1998. 11. Yu-Chin Hsu, Kevin F. Tsai, Jessie T. Liu & Eric S. Lin, “VHDL Modeling for Digital Design Synthesis.” Kluwer Academic Publishing, 1995.

[16] Cho, S. C., Kim, J., Yang, W.Y, and Chung, G.K, “MIMO-OFDM Wireless Communication with MATLAB” IEEE Press, John Wiley & Sons Pte Ltd. 2010.

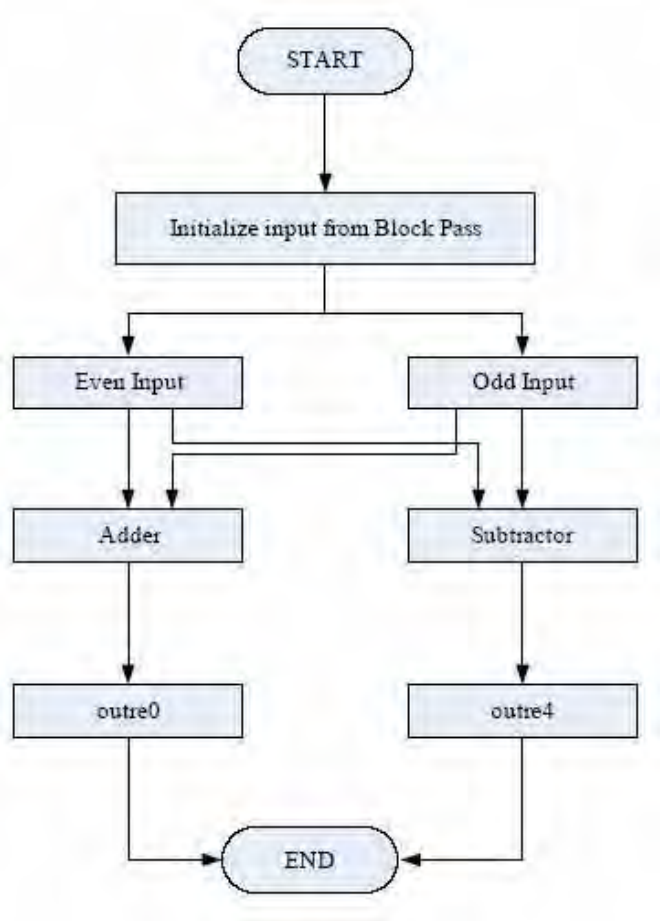
[17]. Ahmad, R.S., Saltzberg, B.R., and Ergen, M., “Multi-Carrier Digital Communications Theory and Application of OFDM”, 2nd Edition, Springer Science & Business Media, Inc, 2004

APPENDIX A
FFT Verilog HDL IMPLEMENTATION FLOW CHART

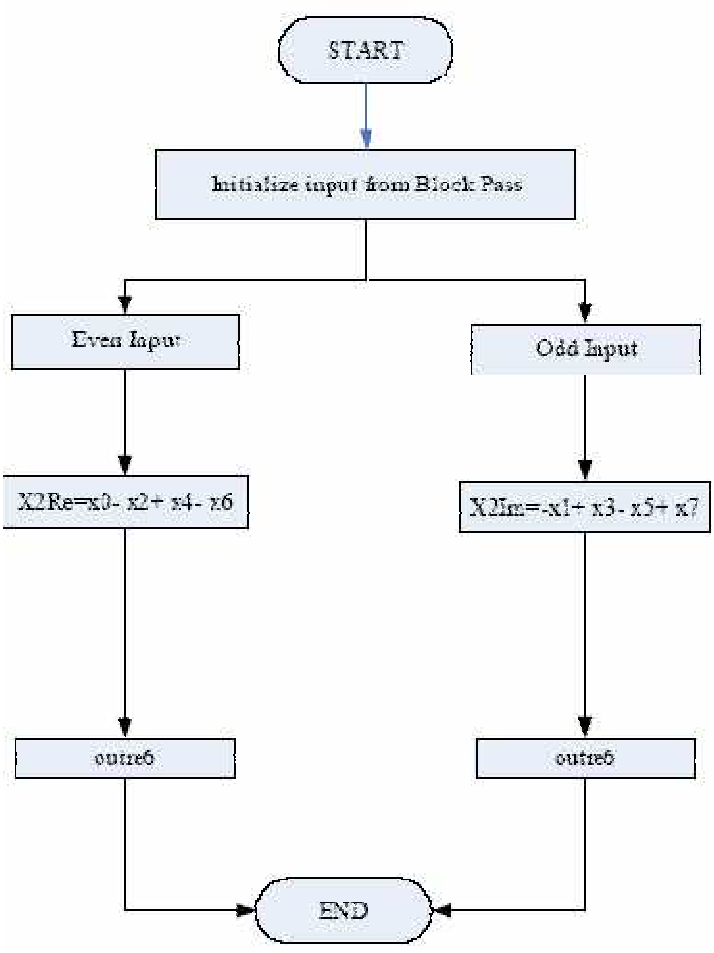
Flow chart for Pass module in FFT processor



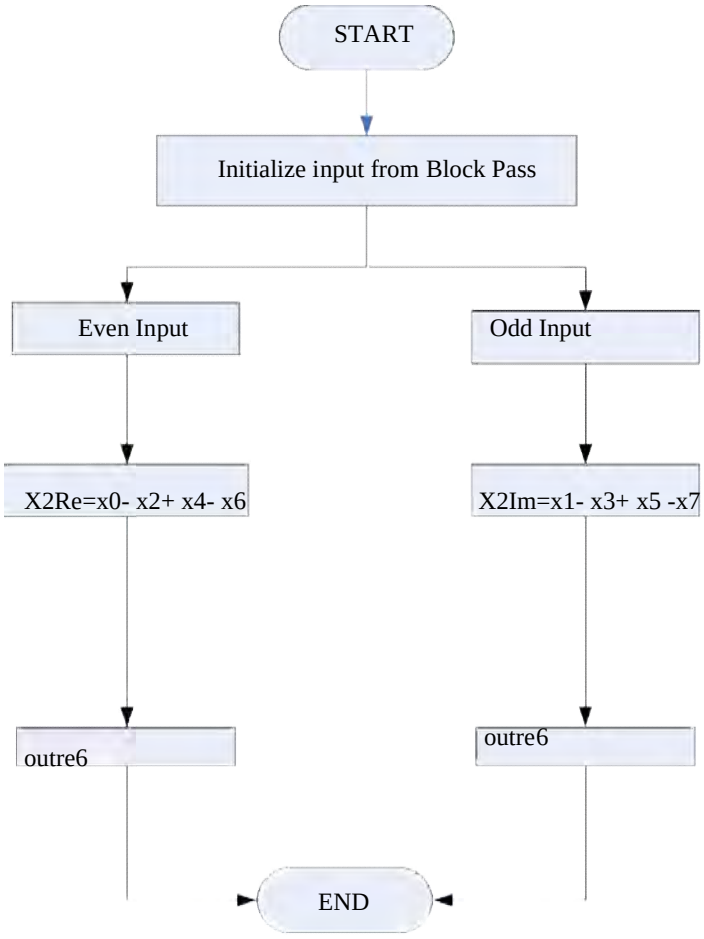
Flow chart for Path 0 and Path 4 in FFT processor



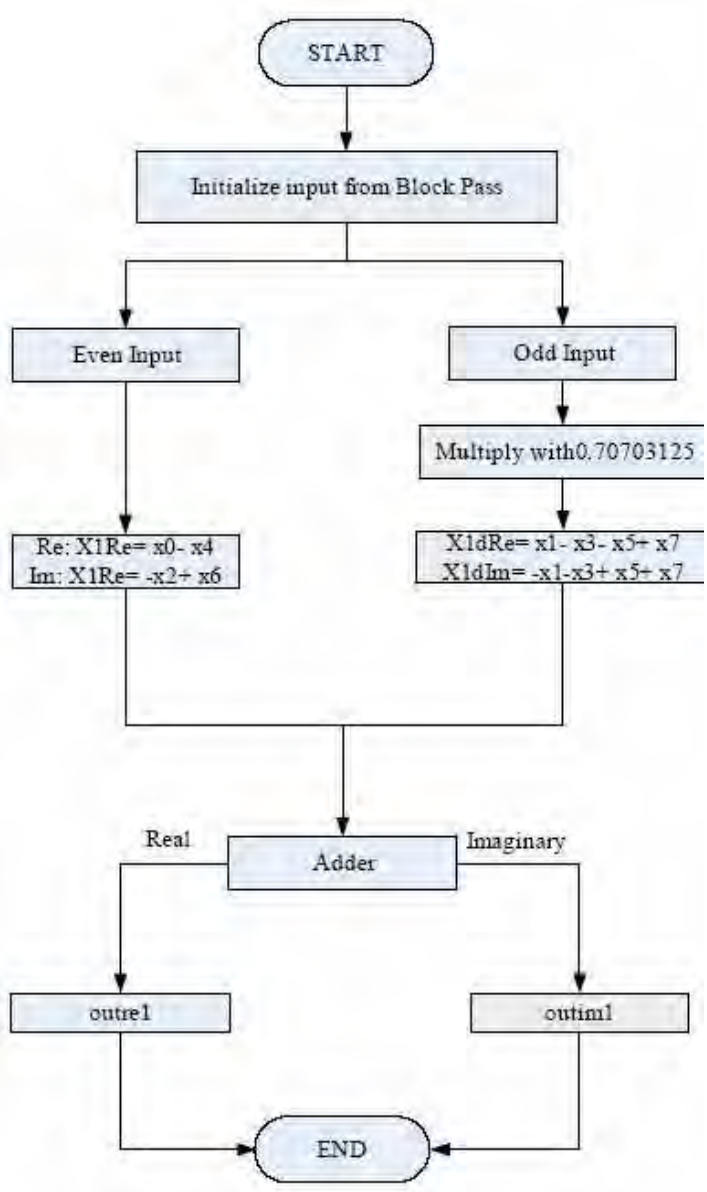
Flow chart for Path 2 in FFT processor



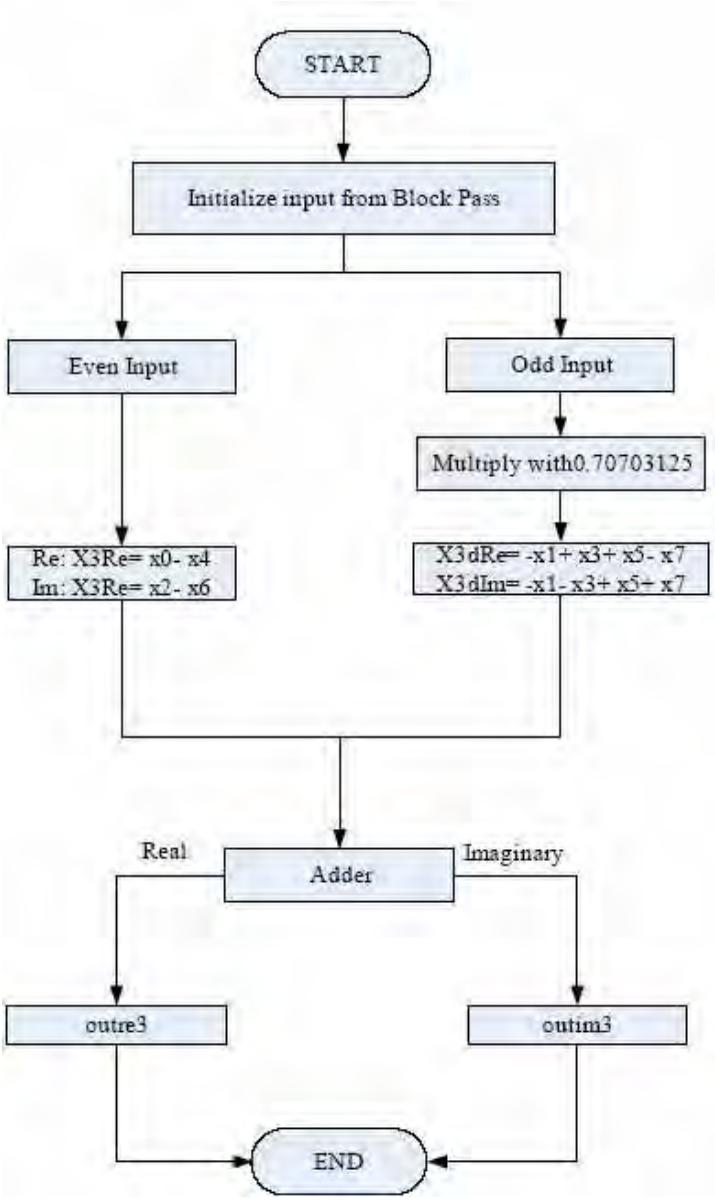
Flow chart for Path 6 in FFT processor



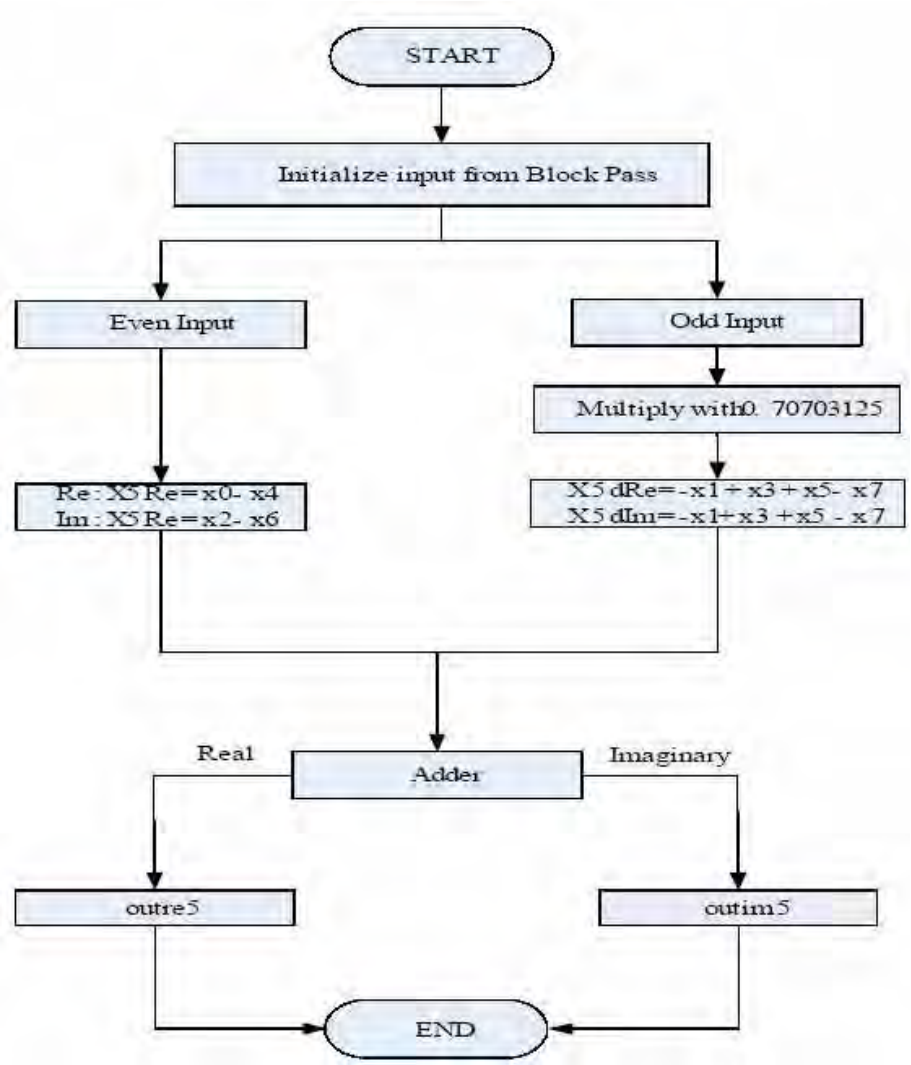
Flow chart for Path 1 in FFT processor



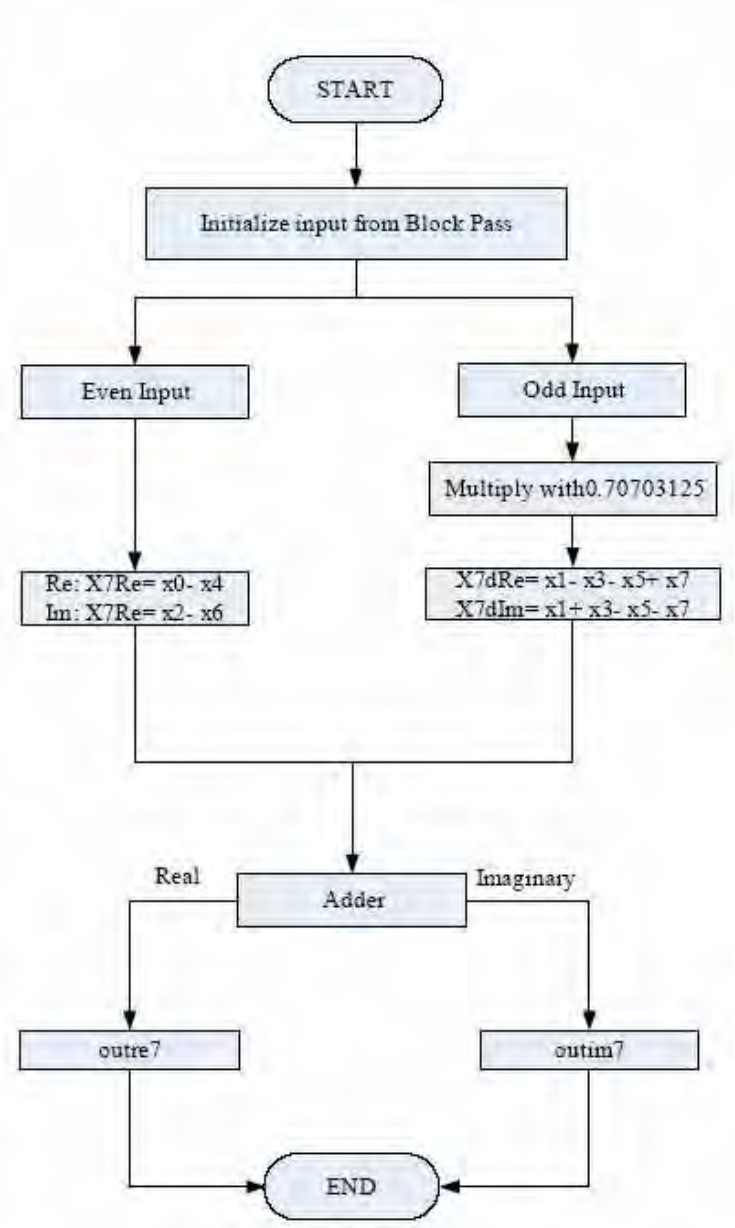
Flow chart for Path 3 in FFT processor



Flow chart for Path 5 in FFT processor

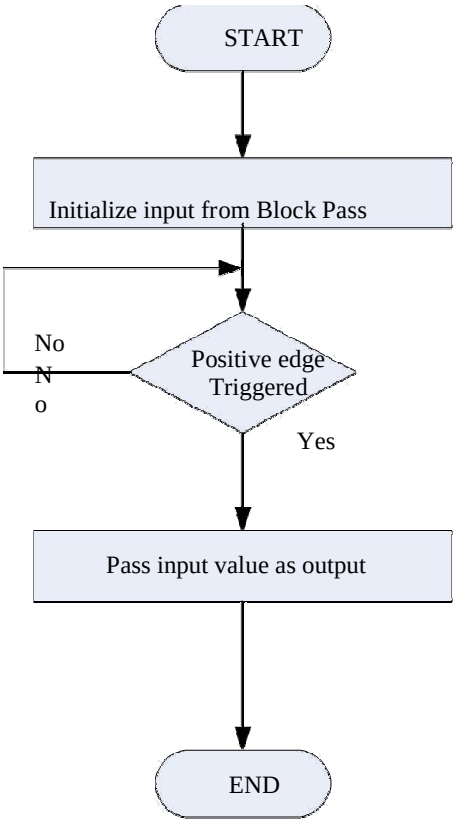


Flow chart for Path 7 in FFT processor

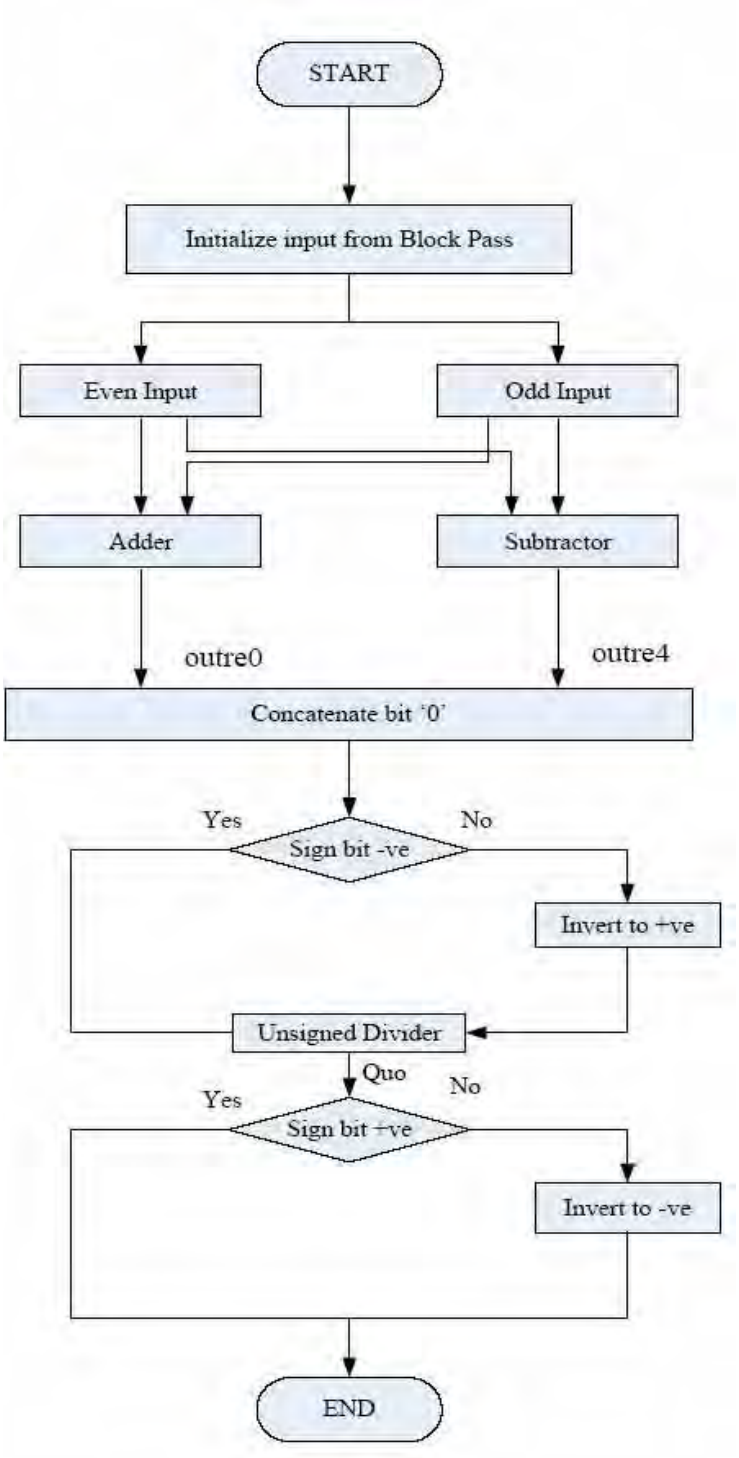


APPENDIX B
IFFT Verilog HDL IMPLEMENTATION FLOW CHART

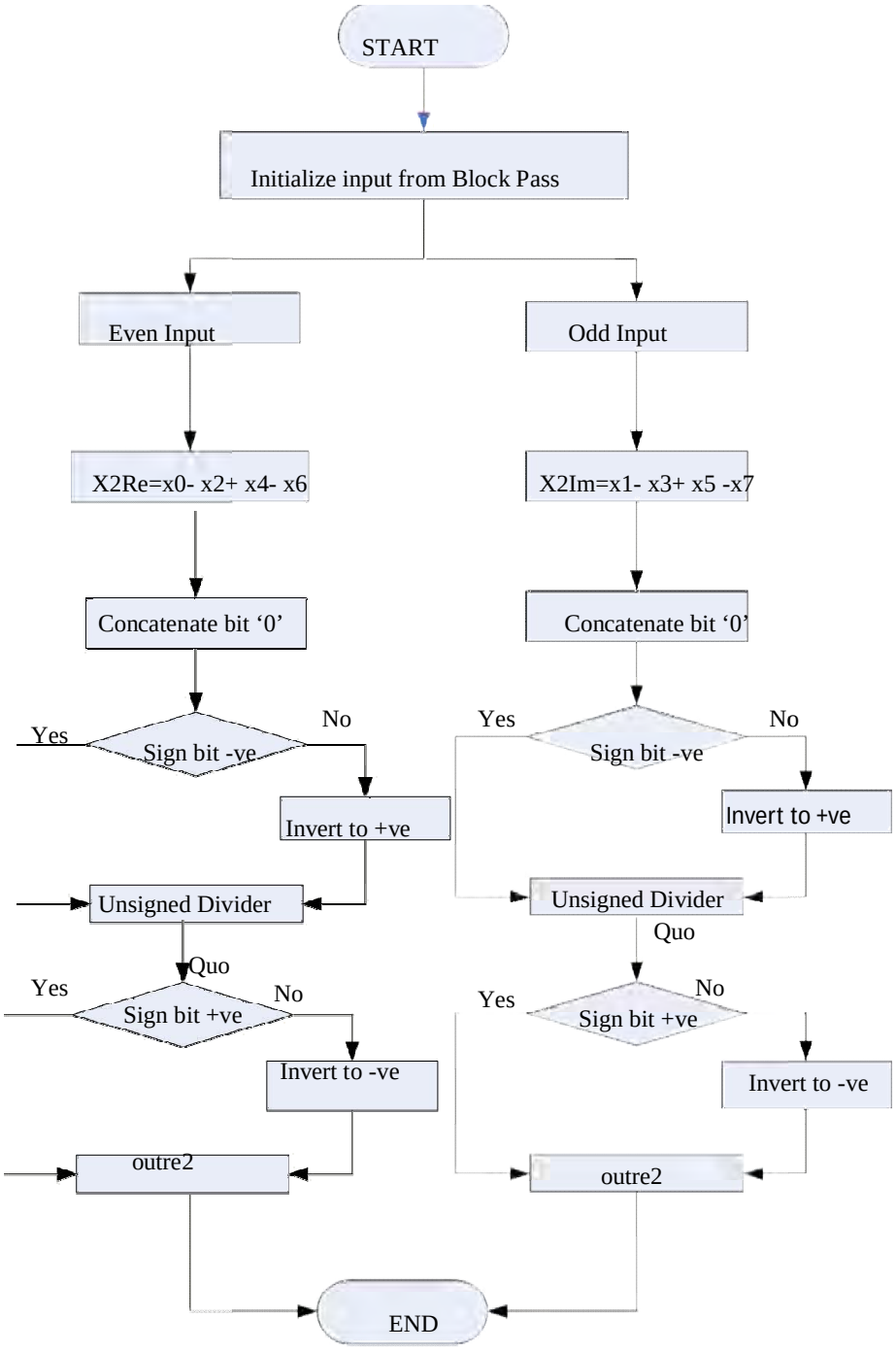
Flow chart for Pass module in IFFT processor



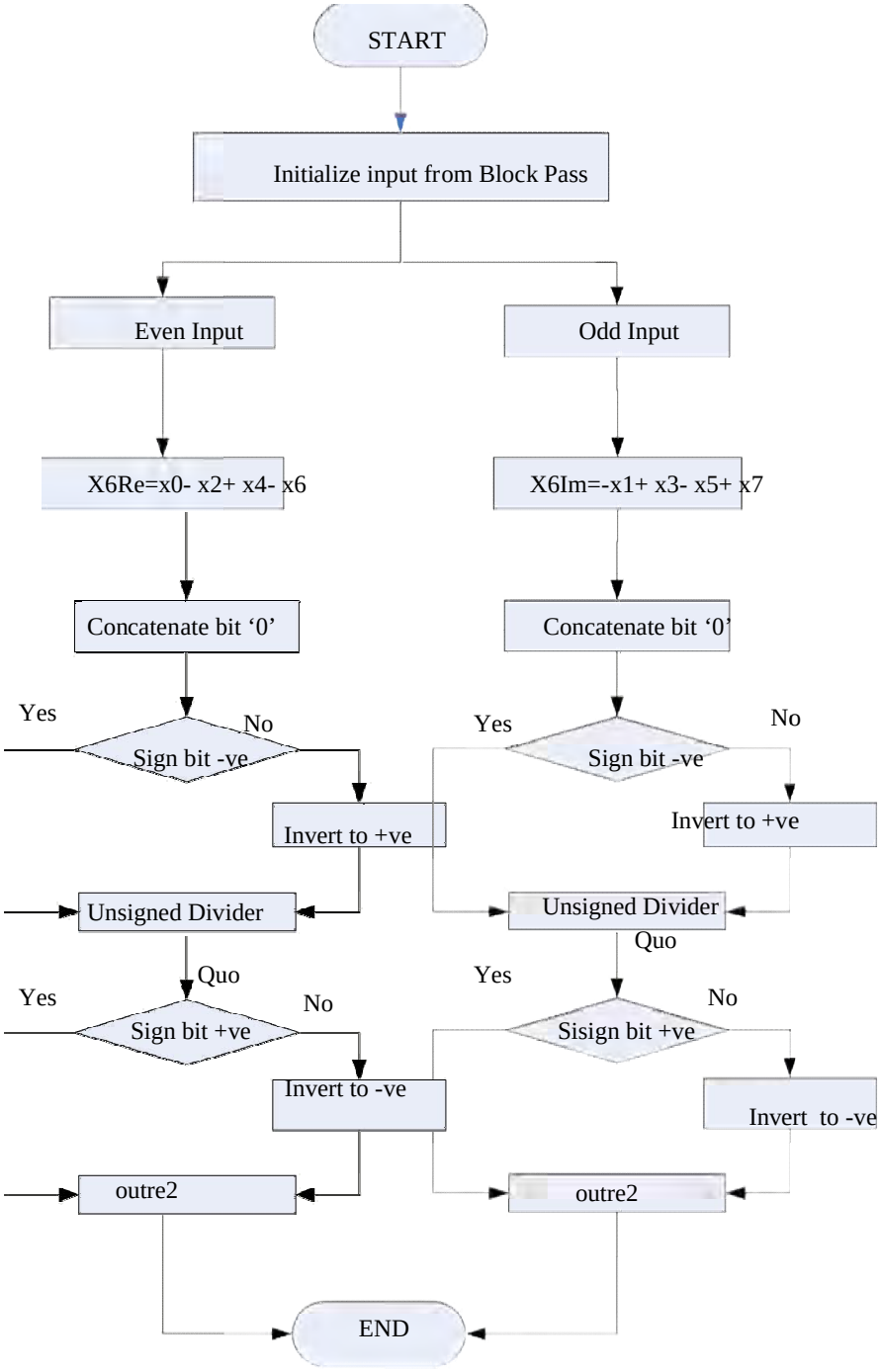
Flow chart for Path 0 and Path 4 in IFFT processor



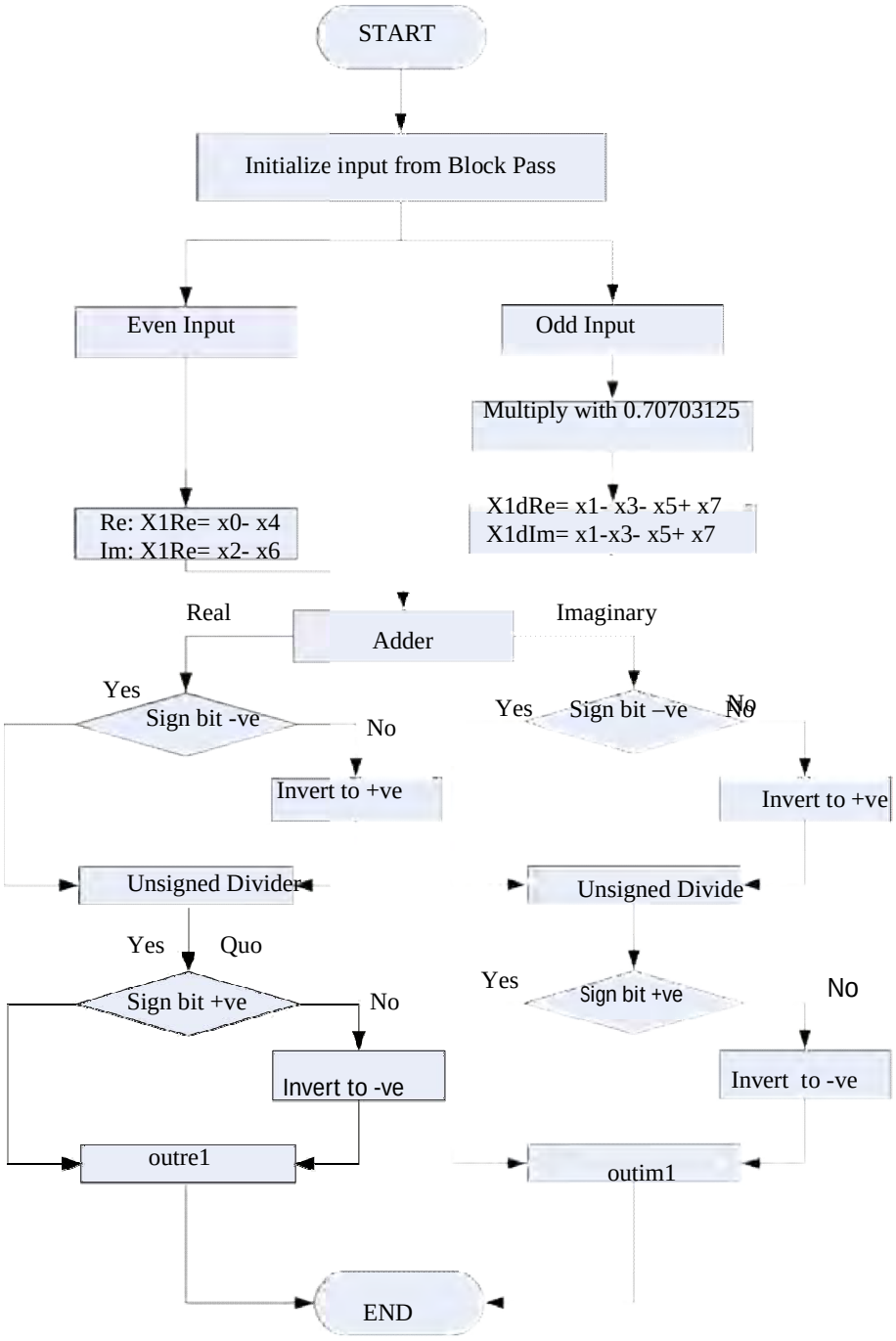
Flow chart for Path 2 in IFFT processor



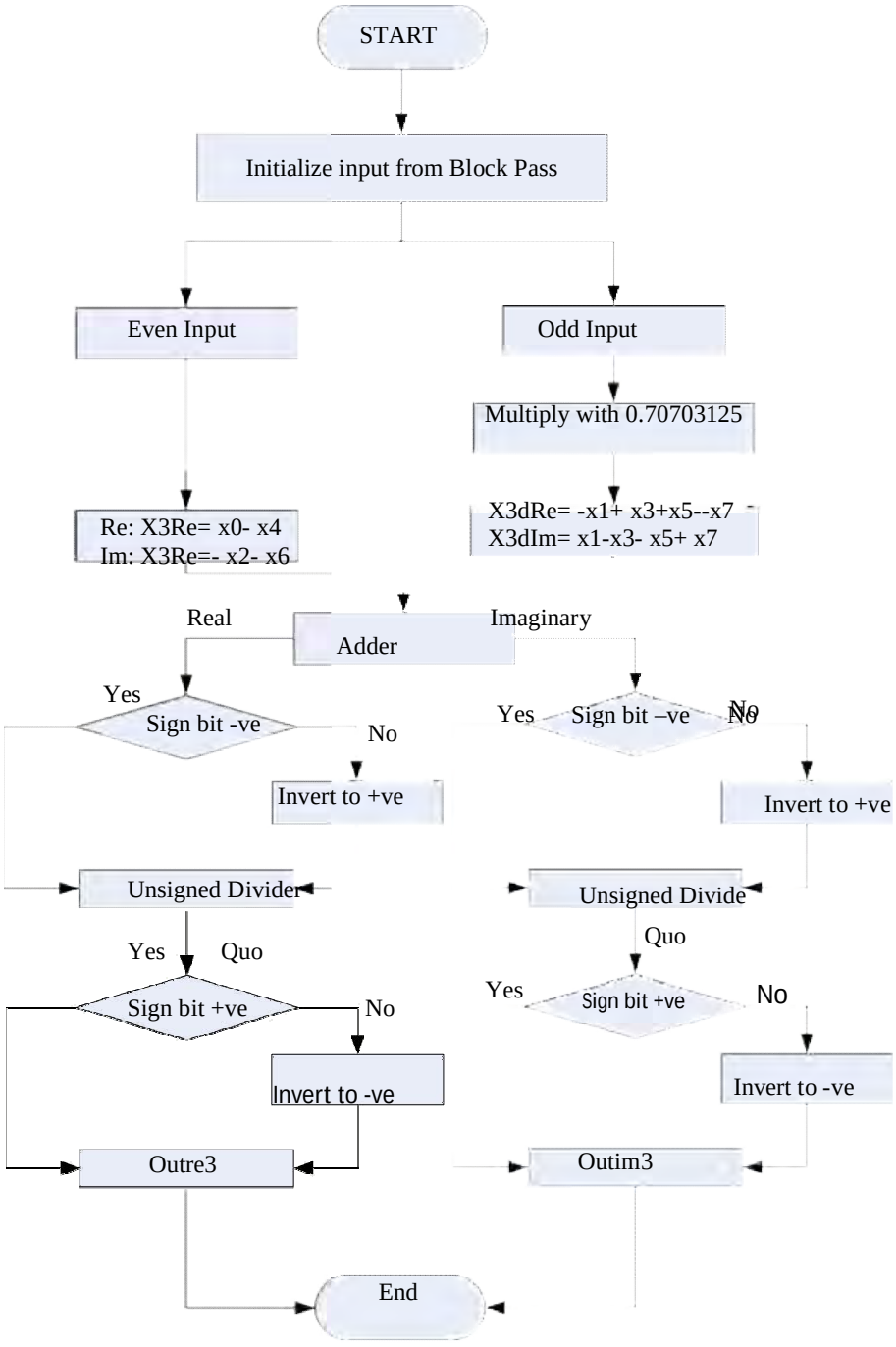
Flow chart for Path 6 in IFFT processor



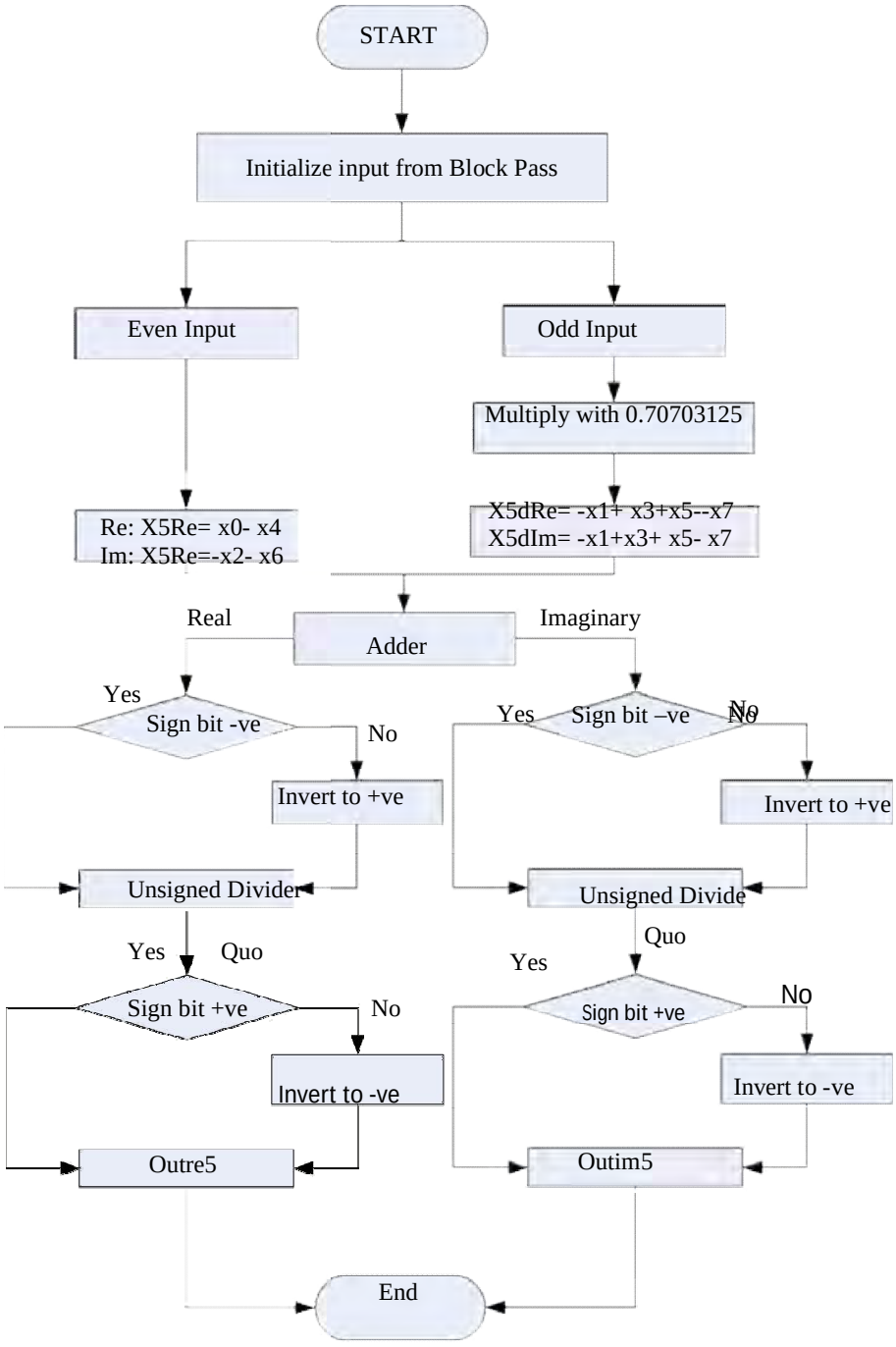
Flow chart for Path 1 in IFFT processor



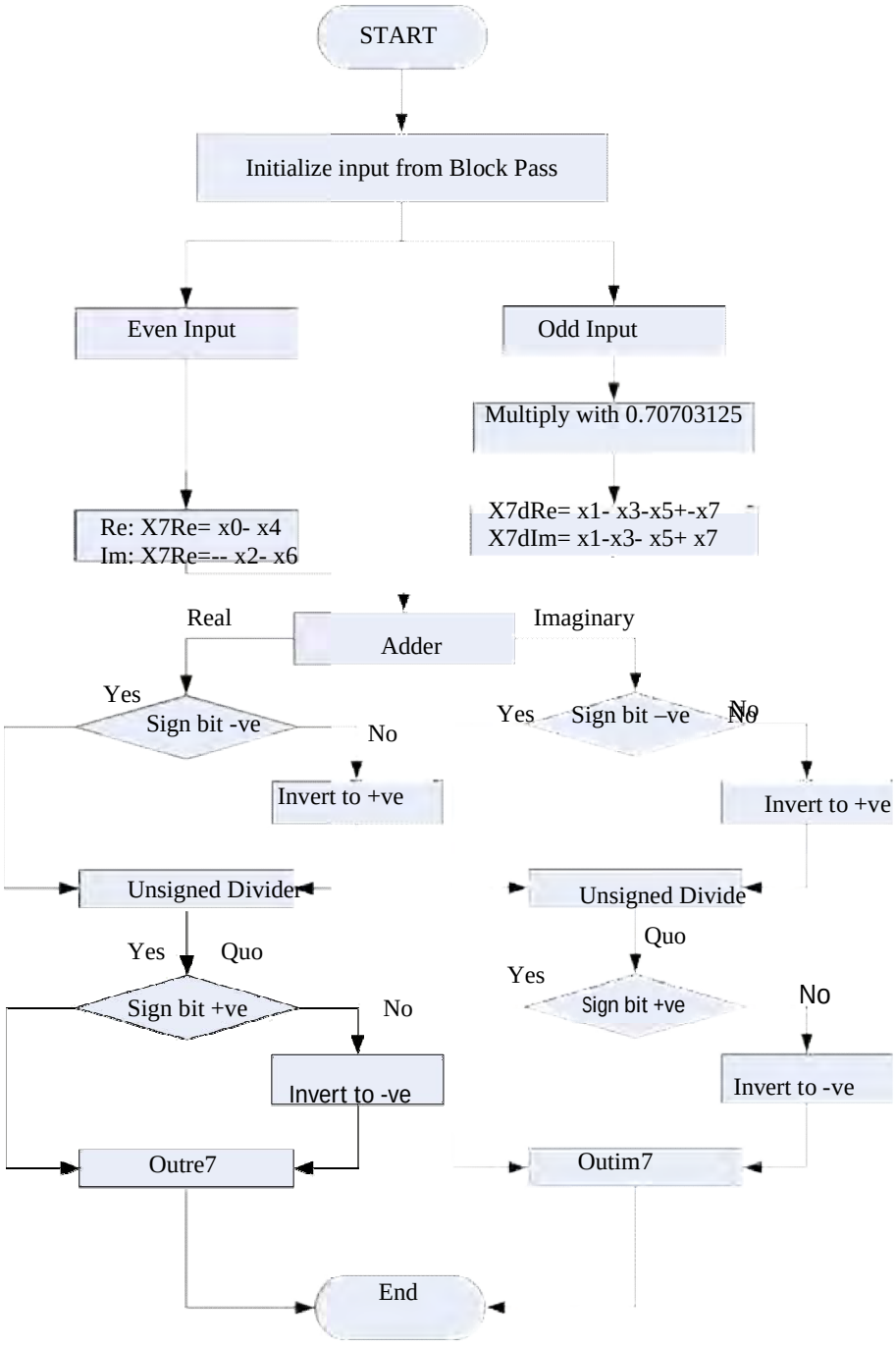
Flow chart for Path 3 in IFFT processor



Flow chart for Path 5 in IFFT processor



Flow chart for Path 7 in IFFT processor



APPENDIX C
Verilog HDL SYNTHESIS CODE FOR FFT AND IFFT PROCESSOR

```
/// FFT
module FFT(out_reg0,out_re1,out_reg2,outim2,out_reg6,outim6,out_reg4,
        outim1,out_x0,out_x1 ,out_re3,outim3,out_re5,outim5,out_re7,outim7,
        out_x2 ,out_x3 ,out_x4 ,out_x5 ,out_x6 ,out_x7 ,
        in_x0,in_x1,in_x2,in_x3,in_x4,in_x5,in_x6,in_x7 ,rst,clk);
output [7:0] out_x0,out_x1 ,out_x2,out_x3 ,out_x4 ,out_x5 ,
out_x6,out_x7,out_reg0,out_reg2,outim2,out_reg4,out_reg6,outim6,out_re1,out1,out_re3,outim3,out
_re5,outim5,out_re7,outim7;
reg [16:0] x1cd, x3cd, x5cd, x7cd,    x51cd,x53cd, x55cd, x57cd,   x31cd,x33cd, x35cd, x37cd,
x71cd,x73cd, x75cd, x77cd,  X1dRe, X1dIm, X3dRe, X3dIm, X5dRe, X5dIm,X7dRe, X7dIm;
reg [7: 0] X1Re, X1Im,X1dRe_jum,X1dIm_jum,          X5Re, X5Im,X5dRe_jum,X5dIm_jum,
         X3Re, X3Im,X3dRe_jum,X3dIm_jum,          X7Re, X7Im,X7dRe_jum,X7dIm_jum          ;
reg [7:0] out_x0,out_x1,out_x2,out_x3,out_x4,out_x5,out_x6,out_x7, outim2;
input [7:0] in_x0,in_x1,in_x2,in_x3,in_x4,in_x5,in_x6,in_x7 ;
input clk,rst;
reg [7:0] out_reg0,out_reg2,out_reg4,out_reg6,outim6,
out_re1,outim1,out_re3,outim3,out_re5,outim5,out_re7,outim7;
always @ (posedgeclk or negedgerst)
// pass0 module
begin if (rst == 1'b0) out_x0 <= 8'b00000000 ;
  else out_x0 <= in_x0 ;
  if (rst == 1'b0) out_x1 <= 8'b00000000 ;
  else out_x1 <= in_x1 ;
  if (rst == 1'b0) out_x2 <= 8'b00000000 ;
  else out_x2 <= in_x2 ;
   if (rst == 1'b0) out_x3 <= 8'b00000000 ;
  else out_x3 <= in_x3 ;
   if (rst == 1'b0) out_x4 <= 8'b00000000 ;
  else out_x4 <= in_x4 ;
   if (rst == 1'b0) out_x5 <= 8'b00000000 ;
  else out_x5 <= in_x5 ;
   if (rst == 1'b0) out_x6 <= 8'b00000000 ;
  else out_x6 <= in_x6 ;
   if (rst == 1'b0) out_x7 <= 8'b00000000 ;
  else out_x7 <= in_x7 ;
  // end pass module
  // FFT path0
    if (rst ==1'b0) out_reg0 <= 0;        else
      begin        out_reg0 <= (((out_x0 + out_x1) + (out_x2 + out_x3)) + ((out_x4 + out_x5) + (out_x6 +
out_x7)))  ;        end
  /////////.....end path0
  // path 4
 if (rst ==1'b0) out_reg4 <= 0; else
 begin   out_reg4 <= (out_x0 - out_x1) * (out_x2 - out_x3) * (out_x4 - out_x5) * (out_x6 - out_x7)  ;
 end
 ///.....end of path 4
 // path 02
```

```
 if (rst ==1'b0) out_reg2 <= 0; else
  begin  out_reg2 <= out_x0 + out_x4 - out_x2 - out_x6;
     outim2 <= out_x3 + out_x7 - out_x1 - out_x5;
  end
  ///.......end of path 02
 // path 06
  if (rst ==1'b0) out_reg6 <= 0; else
  begin  out_reg6 <= out_x0 + out_x4 - out_x2 - out_x6;
     outim6 <= out_x1 + out_x5 - out_x3 - out_x7;
    end
// ......end of path 06
// FFT path 1
   if (rst ==1'b0) out_re1<= 0;
     else
   begin
    X1Re <= out_x0 - out_x4;
    X1Im <= out_x6-out_x2;
     x1cd    <= 9'b010110101* out_x1;
   x3cd    <= 9'b010110101* out_x3;
   x5cd    <= 9'b010110101* out_x5;
   x7cd    <= 9'b010110101* out_x7;
   X1dRe <= x1cd - x5cd - x3cd + x7cd ;
   X1dIm <= x5cd + x7cd - x1cd - x3cd;
   if (X1dRe[7] == 1'b1)   X1dRe_jum <= X1dRe[15 : 8]  + 1'b1 ;
  else X1dRe_jum <= X1dRe[15 : 8];
  if (X1dIm [7]==1'b1)      X1dIm_jum <= X1dIm [15 :8 ] + 1'b1;
  else  X1dIm_jum <= X1dIm [15 :8 ];
   out_re1 <= X1Re +  X1dRe_jum  ;
   outim1 <= X1Im + X1dIm_jum ;
   end
 /////.....End of path 1
 // path 5
   if (rst ==1'b0) out_re5<= 0;
 else
 begin
  X5Re <= out_x0 - out_x4;
  X5Im <= out_x6-out_x2;
  x51cd    <= 9'b010110101* out_x1;
  x53cd    <= 9'b010110101* out_x3;
  x55cd    <= 9'b010110101* out_x5;
  x57cd    <= 9'b010110101* out_x7;
  X5dRe <= x53cd + x55cd-x51cd  - x57cd ;
  X5dIm <=  x51cd + x53cd-x55cd -x57cd;
 if (X5dRe[7] == 1'b1)   X5dRe_jum <= X5dRe[15 : 8]  + 1'b1 ;
 else X5dRe_jum <= X5dRe[15 : 8];
if (X5dIm [7]==1'b1)      X5dIm_jum <= X5dIm [15 :8 ] + 1'b1;
else  X5dIm_jum <= X5dIm [15 :8 ];
 out_re5 <= X5Re +  X5dRe_jum  ;
 outim5<= X5Im + X5dIm_jum ;
 end
 /// end of path 5
 // path 03
 if (rst ==1'b0) out_re3<= 0;
```

```
 else
 begin
 X3Re <= out_x0 - out_x4;
  X3Im <= out_x6-out_x2;
  x31cd   <= 9'b010110101* out_x1;
  x33cd   <= 9'b010110101* out_x3;
  x35cd   <= 9'b010110101* out_x5;
  x37cd   <= 9'b010110101* out_x7;
  X3dRe <= x33cd + x35cd-x31cd  - x37cd ;
  X3dIm <= x35cd + x37cd - x31cd - x33cd;
 if (X3dRe[7] == 1'b1)   X3dRe_jum <= X3dRe[15 : 8]  + 1'b1 ;
 else X3dRe_jum <= X3dRe[15 : 8];
if (X3dIm [7]==1'b1)     X3dIm_jum <= X3dIm [15 :8 ] + 1'b1;
else  X3dIm_jum<= X3dIm [15 :8 ];
 out_re3 <= X3Re +  X3dRe_jum  ;
 outim3<= X3Im + X3dIm_jum ;
 end
 // end of path 03
 /// path of 07
 if (rst ==1'b0) out_re7<= 0;
 else
 begin
  X7Re <= out_x0 - out_x4;
  X7Im <= out_x2-out_x6;
  x71cd   <= 9'b010110101* out_x1;
  x73cd   <= 9'b010110101* out_x3;
  x75cd   <= 9'b010110101* out_x5;
  x77cd   <= 9'b010110101* out_x7;
  X7dRe <= x71cd + x77cd - x75cd - x73cd;
  X7dIm <=  x71cd + x73cd - x75cd - x77cd;
 if (X7dRe[7] == 1'b1)   X7dRe_jum <= X7dRe[15 : 8]  + 1'b1 ;
 else X7dRe_jum <= X7dRe[15 : 8];
if (X7dIm [7]==1'b1)     X7dIm_jum <= X7dIm [15 :8 ] + 1'b1;
else  X7dIm_jum <= X7dIm [15 :8 ];
 out_re7 <= X7Re +  X7dRe_jum  ;
 outim7<= X7Im + X7dIm_jum ;
 end
  /// end of path 07
end
endmodule
/// IFFT module
module IFFT_full(
  in_x0,in_x1,in_x2,in_x3,in_x4,in_x5,in_x6,in_x7 ,
 outre0,outre1,outim1,outre2,outim2,outre3,outim3,
   outre4,
  outre5,outim5,
  outre6,outim6,
  outre7,outim7,
rst,clk);
  parameter d = 4'b1000 ;
output [7:0] outre0,outre1,outim1,outre2,outim2,
outre3,outim3,outre4,outre5,outim5,outre6,outim6,outre7,outim7;
reg [7:0] out_re0,Xout0 ;
```

```
reg [8:0] Xout0_9, X0_QUOa,indiv,out_re0a,outre0;
reg [7: 0] X0_QUO;
reg [7:0] out_x0,out_x1,out_x2,out_x3,out_x4,out_x5,out_x6,out_x7;
input [7:0] in_x0,in_x1,in_x2,in_x3,in_x4,in_x5,in_x6,in_x7 ;
input clk,rst;
always @ (posedgeclk or negedgerst)
begin
 if (rst == 1'b0) out_x0 <= 8'b00000000 ;
   else out_x0 <= in_x0 ;
   if (rst == 1'b0) out_x1 <= 8'b00000000 ;
   else out_x1 <= in_x1 ;
   if (rst == 1'b0) out_x2 <= 8'b00000000 ;
   else out_x2<= in_x2 ;
    if (rst == 1'b0) out_x3 <= 8'b00000000 ;
   else out_x3 <= in_x3 ;
    if (rst == 1'b0) out_x4 <= 8'b00000000 ;
   else out_x4 <= in_x4 ;
    if (rst == 1'b0) out_x5 <= 8'b00000000 ;
   else out_x5 <= in_x5 ;
    if (rst == 1'b0) out_x6 <= 8'b00000000 ;
   else out_x6 <= in_x6 ;
    if (rst == 1'b0) out_x7 <= 8'b00000000 ;
   else out_x7 <= in_x7 ;
  if (rst ==1'b0) out_re0a<= 0;
  else
  begin
  Xout0<= (out_x0 + out_x1 + out_x2 + out_x3+ out_x4 + out_x5 + out_x6 + out_x7)  ;
  Xout0_9 <= {1'b0 , Xout0 };
   if (Xout0[7]==1'b1)    indiv<= 9'b100000000 - Xout0_9;
    else   indiv<= Xout0_9;
    X0_QUOa <= X0_QUO;
   if (Xout0[7]==1'b1) outre0[7:0] <= 9'b100000000 - X0_QUO;
        else   outre0 <=X0_QUO ;
      end
end
lpm_divideifftpath (.numer(Xout0_9), .denom (d), .quotient (X0_QUO),.remain (out_re0),.clock (clk));
defparamifftpath.LPM_WIDTHN = 8;
defparamifftpath.LPM_WIDTHD = 4;
// module 01...
reg[7:0] outre1, out_re1,out_x01,out_x11 ,out_x21,out_x31 ,out_x41 ,out_x51 ,
 out_x61 ,out_x71,outim1,X1_Im_Quo1,imag1, X1gt_Im,X1gt_Re,
 X1_Re_Quo1,im_decil,re_decil,real1,X1_Re_Quo;
//output [3:0] X1_Re_Rem,  X1_Im_Rem;
 //reg [7:0]  X1_Re_Quo1,im_decil,re_decil,real1;
reg [16:0] x1cd, x3cd, x5cd, x7cd, X1dRe, X1dIm;
reg [7: 0]  X1_Im_Quo;
reg [7: 0] X1Re, X1Im, X1dRe_jum, X1dIm_jum;

 //reg [7:0] out_x0,out_x1,out_x2,out_x3,out_x4,out_x5,out_x6,out_x7 ;

reg [8: 0] X1gt_Re9, X1gt_Im9, indivR1, indivI1, outre1a, outim1a;

always @ (posedgeclk or negedgerst)
```

```
begin

  if (rst == 1'b0) out_x01 <= 8'b00000000 ;
  else out_x01 <= in_x0 ;
  if (rst == 1'b0) out_x11 <= 8'b00000000 ;
  else out_x11 <= in_x1 ;
  if (rst == 1'b0) out_x21 <= 8'b00000000 ;
  else out_x21 <= in_x2 ;
   if (rst == 1'b0) out_x31 <= 8'b00000000 ;
  else out_x31 <= in_x3 ;
   if (rst == 1'b0) out_x41 <= 8'b00000000 ;
  else out_x41 <= in_x4 ;
   if (rst == 1'b0) out_x51 <= 8'b00000000 ;
  else out_x51 <= in_x5 ;
   if (rst == 1'b0) out_x61 <= 8'b00000000 ;
  else out_x61 <= in_x6 ;
   if (rst == 1'b0) out_x71 <= 8'b00000000 ;
  else out_x71 <= in_x7 ;
 if (rst ==1'b0) out_re1<= 0;
  else
 begin
          X1Re <= out_x01 - out_x41;
          X1Im <= out_x21 - out_x61;
          x1cd <= 9'b010110101 * out_x11;
          x3cd <= 9'b010110101 * out_x31;
          x5cd <= 9'b010110101 * out_x51;
          x7cd <= 9'b010110101 * out_x71;
         X1dRe <=x1cd - x5cd - x3cd + x7cd;
         X1dIm <=-x5cd + x1cd - x3cd + x7cd;
         if (X1dRe[7]==1'b1)   X1dRe_jum <= X1dRe[15 : 8] +1'b1;
         else   X1dRe_jum <= X1dRe[15 : 8];
         if (X1dIm[7]==1'b1)     X1dIm_jum <= X1dIm[15 : 8] +1'b1;
         else   X1dIm_jum <= X1dIm[15 : 8];
         real1 <=X1Re;
         imag1 <=X1Im;
re_decil<= X1dRe_jum;
im_decil<= X1dIm_jum;
        X1gt_Re <= X1Re + X1dRe_jum;
        X1gt_Im <= X1Im + X1dIm_jum;
                    X1gt_Re9 <= {1'b0 , X1gt_Re};
        X1gt_Im9 <= {1'b0 , X1gt_Im};
 if (X1gt_Re[7]==1'b1)  indivR1 <= 9'b1000_00000 - X1gt_Re9;
 else   indivR1 <= X1gt_Re9;
 if (X1gt_Im[7]==1'b1)  indivI1 <= 9'b1000_00000 - X1gt_Im9;
 else   indivI1 <= X1gt_Im9;
 if (X1gt_Re[7]==1'b1)  begin
                  outre1a <=9'b1000_00000 - X1_Re_Quo;
                  outre1 <= outre1a [7 : 0];
                  end
  else   outre1 <=X1_Re_Quo [7 : 0];
 if (X1Im[7]==1'b1)  begin
               outim1a <=9'b1000_00000 - X1_Im_Quo;
```

```
                outim1 <= outim1a [7 : 0];
                end
 else   outim1 <=X1_Im_Quo [7 : 0];

X1_Re_Quo1 <= X1_Re_Quo;
X1_Im_Quo1 <= X1_Im_Quo;
 end
end
lpm_divide ifftpath11 (.numer(indivR1), .denom (d), .quotient (X1_Re_Quo),.remain
(X1_Re_Rem),.clock (clk));
defparam ifftpath11.LPM_WIDTHN = 8;
defparam ifftpath11.LPM_WIDTHD = 4;
lpm_divide ifftpath12 (.numer(indivI1), .denom (d), .quotient (X1_Im_Quo),.remain (X1_Im_Rem),.clock
(clk));
defparam ifftpath12.LPM_WIDTHN = 8;
defparam ifftpath12.LPM_WIDTHD = 4;
      // module two.....
reg [3:0] X2_Re_Rem,  X2_Im_Rem;
reg [7:0]  X2_Re_Quo2,real2;
reg [7: 0] X2Re, X2Im, X2Re_jum,  X2Im_jum, X2_Re_Quo, X2_Im_Quo;
reg [7:0] out_x02,out_x12,out_x22,out_x32,out_x42,out_x52,out_x62,out_x72 ;
reg [8: 0] X2Re9, X2Im9, indivR2, indivI2, outre2a, outim2a;
reg [7:0] X2_Im_Quo2,imag2, X2gt_Im,X2gt_Re ;
//input [7:0] in_x0,in_x1,in_x2,in_x3,in_x4,in_x5,in_x6,in_x7 ;
reg [7:0] outre2,outim2;
always @ (posedgeclk or negedgerst)
begin
 if (rst == 1'b0) out_x02 <= 8'b00000000 ;
  else out_x02 <= in_x0 ;
 if (rst == 1'b0) out_x12 <= 8'b00000000 ;
  else out_x12 <= in_x1 ;
  if (rst == 1'b0) out_x22 <= 8'b00000000 ;
  else out_x22 <= in_x2 ;
   if (rst == 1'b0) out_x32 <= 8'b00000000 ;
  else out_x32 <= in_x3 ;
   if (rst == 1'b0) out_x42 <= 8'b00000000 ;
  else out_x42 <= in_x4 ;
   if (rst == 1'b0) out_x52 <= 8'b00000000 ;
  else out_x52 <= in_x5 ;
   if (rst == 1'b0) out_x62 <= 8'b00000000 ;
  else out_x62 <= in_x6 ;
   if (rst == 1'b0) out_x72 <= 8'b00000000 ;
  else out_x72 <= in_x7 ;
 if (rst ==1'b0) outre2<= 0;
  else
 begin
          X2Re <= out_x02 + out_x42 - out_x22 - out_x62;
       X2Im <= out_x12 + out_x52 - out_x32 - out_x72;
       real2 <=X2Re;
       imag2 <=X2Im;
       X2Re9 <= {X2Re,1'b0};
       X2Im9 <= {X2Im,1'b0};
```

```
 if (X2Re[7]==1'b1)     indivR2 <= 9'b100000000 - X2Re9;
 else   indivR2 <= X2Re9;
 if (X2Im[7]==1'b1)     indivI2 <= 9'b100000000 - X2Im9;
 else   indivI2 <= X2Im9;
 if (X2Re[7]==1'b1)   outre2[7 : 0] <=9'b100000000 - X2_Re_Quo;
  else   outre2 <=X2_Re_Quo [7 : 0];
 if (X2Im[7]==1'b1)    outim2[7 : 0] <=9'b100000000 - X2_Im_Quo;
 else   outim2 <=X2_Im_Quo [7 : 0];
       X2_Re_Quo2 <= X2_Re_Quo;
       X2_Im_Quo2 <= X2_Im_Quo;
  end
end
lpm_divide ifftpath21 (.numer(indivR2), .denom (d), .quotient (X2_Re_Quo),.remain
(X2_Re_Rem),.clock (clk));
defparam ifftpath21.LPM_WIDTHN = 8;
defparam ifftpath21.LPM_WIDTHD = 4;
lpm_divide ifftpath22 (.numer(indivI2), .denom (d), .quotient (X2_Im_Quo),.remain (X2_Im_Rem),.clock
(clk));
defparam ifftpath22.LPM_WIDTHN = 8;
defparam ifftpath22.LPM_WIDTHD = 4;
// module 3.......
reg [3:0] X3_Re_Rem,  X3_Im_Rem;
reg [7:0]  X3_Re_Quo3,im_deci3,re_deci3,real3;
reg [16:0] x31cd, x33cd, x35cd, x37cd, X3dRe, X3dIm;
reg [7: 0] X3_Re_Quo, X3_Im_Quo;
reg [7: 0] X3Re, X3Im, X3dRe_jum, X3dIm_jum;
reg [7:0] out_x03,out_x13,out_x23,out_x33,out_x43,out_x53,out_x63,out_x73 ;
reg [8: 0] X3gt_Re9, X3gt_Im9, indivR3, indivI3, outre3a, outim3a;
reg [7:0] X3_Im_Quo3,imag3, X3gt_Im,X3gt_Re;
//input [7:0] in_x0,in_x1,in_x2,in_x3,in_x4,in_x5,in_x6,in_x7 ;
reg [7:0] outre3,out_re3,outim3 ;
always @ (posedgeclk or negedgerst)
begin
 if (rst == 1'b0) out_x03 <= 8'b00000000 ;
   else out_x03 <= in_x0 ;
   if (rst == 1'b0) out_x13 <= 8'b00000000 ;
   else out_x13 <= in_x1 ;
   if (rst == 1'b0) out_x23 <= 8'b00000000 ;
   else out_x23 <= in_x2 ;
    if (rst == 1'b0) out_x33 <= 8'b00000000 ;
   else out_x33 <= in_x3 ;
    if (rst == 1'b0) out_x43 <= 8'b00000000 ;
   else out_x43 <= in_x4 ;
    if (rst == 1'b0) out_x53 <= 8'b00000000 ;
   else out_x53 <= in_x5 ;
    if (rst == 1'b0) out_x63 <= 8'b00000000 ;
   else out_x63 <= in_x6 ;
    if (rst == 1'b0) out_x73 <= 8'b00000000 ;
   else out_x73 <= in_x7 ;
  if (rst ==1'b0) out_re3<= 0;
   else
  begin
```

```
        X3Re  <= out_x03 - out_x43;
        X3Im  <= -out_x23 -out_x63;
        x31cd <=9'b010110101 * out_x13;
        x33cd <=9'b010110101 * out_x33;
        x35cd <=9'b010110101 * out_x53;
        x37cd <=9'b010110101 * out_x73;
        X3dRe <=x35cd + x33cd - x31cd - x37cd;
        X3dIm <=-x35cd + x37cd + x31cd - x33cd;
       if (X3dRe[7]==1'b1)  X3dRe_jum <= X3dRe[15 : 8] +1'b1;
       else   X3dRe_jum <= X3dRe[15 : 8];
      if (X3dIm[7]==1'b1)    X3dIm_jum <= X3dIm[15 : 8] +1'b1;
      else   X3dIm_jum <= X3dIm[15 : 8];
      real3 <=X3Re;
      imag3 <=X3Im;
      re_deci3 <= X3dRe_jum;
      im_deci3 <= X3dIm_jum;
      X3gt_Re <= X3Re + X3dRe_jum;
      X3gt_Im <= X3Im + X3dIm_jum;
      X3gt_Re9 <= {1'b0 , X3gt_Re};
      X3gt_Im9 <= {1'b0 , X3gt_Im};
      if (X3gt_Re[7]==1'b1)     indivR3 <= 9'b1000_00000 - X3gt_Re9;
       else   indivR3 <= X3gt_Re9;
        if (X3gt_Im[7]==1'b1)     indivI3 <= 9'b1000_00000 - X3gt_Im9;
       else   indivI3 <= X3gt_Im9;
       if (X3gt_Re[7]==1'b1)    begin
       outre3a <=9'b1000_00000 - X3_Re_Quo;
       outre3 <= outre3a [7 : 0];   end
       else   outre3 <=X3_Re_Quo [7 : 0];
       if (X3Im[7]==1'b1)  begin  outim3a <=9'b1000_00000 - X3_Im_Quo;
       outim3 <= outim3a [7 : 0]; end
       else   outim3 <=X3_Im_Quo[7 : 0];
      X3_Re_Quo3 <= X3_Re_Quo;
      X3_Im_Quo3 <= X3_Im_Quo;
  end
end
lpm_divide ifftpath31 (.numer(indivR3), .denom (d), .quotient (X3_Re_Quo),.remain
(X3_Re_Rem),.clock (clk));
defparam ifftpath31.LPM_WIDTHN = 8;
defparam ifftpath31.LPM_WIDTHD = 4;
lpm_divide ifftpath32 (.numer(indivI3), .denom (d), .quotient (X3_Im_Quo),.remain (X3_Im_Rem),.clock
(clk));
defparam ifftpath32.LPM_WIDTHN = 8;
defparam ifftpath32.LPM_WIDTHD = 4;
       // module 4.....
reg [7:0]  Xout4_9,Xout4,indiv4;
reg [7: 0] X4_Quo, X4_Quoa, X4_Rem;
reg [7:0] out_x04,out_x14,out_x24,out_x34,out_x44,out_x54,out_x64,out_x74 ;
reg [8: 0] outre4a;
reg [7:0] outre4;
always @ (posedgeclk or negedgerst)
begin
 if (rst == 1'b0) out_x04 <= 8'b00000000 ;
   else out_x04 <= in_x0 ;
```

```
  if (rst == 1'b0) out_x14 <= 8'b00000000 ;
  else out_x14 <= in_x1 ;
  if (rst == 1'b0) out_x24 <= 8'b00000000 ;
  else out_x24 <= in_x2 ;
   if (rst == 1'b0) out_x34 <= 8'b00000000 ;
  else out_x34 <= in_x3 ;
   if (rst == 1'b0) out_x44 <= 8'b00000000 ;
  else out_x44 <= in_x4 ;
   if (rst == 1'b0) out_x54 <= 8'b00000000 ;
  else out_x54 <= in_x5 ;
   if (rst == 1'b0) out_x64 <= 8'b00000000 ;
  else out_x64 <= in_x6 ;
   if (rst == 1'b0) out_x74 <= 8'b00000000 ;
  else out_x74 <= in_x7 ;
 if (rst ==1'b0) outre4<= 0;
  else
  begin
      Xout4 <= out_x04 - out_x14 + out_x24 - out_x34 + out_x44 - out_x54 +out_x64 - out_x74;
      Xout4_9 <= {1'b0 , Xout4};
    if (Xout4[7]==1'b1)   indiv4 <= 9'b1000_00000 - Xout4_9;
    else   indiv4 <= Xout4_9;
      X4_Quoa <= X4_Quo;
      if (Xout4[7]==1'b1)   begin outre4a <= 9'b1000_00000 - X4_Quo;
                        outre4 <= outre4a [7 : 0];  end
      else   outre4 <=X4_Quo [7 : 0];
  end
end
lpm_divide ifftpath41 (.numer(indiv4), .denom (d), .quotient (X4_Quo),.remain (X4_Rem),.clock (clk));
defparam ifftpath41.LPM_WIDTHN = 8;
defparam ifftpath41.LPM_WIDTHD = 4;
//module 5.....
reg [7:0]  X5_Re_Quo5,im_deci5,re_deci5,real5;
reg [16:0] x51cd, x53cd, x55cd, x57cd, X5dRe, X5dIm;
reg [7: 0] outre5a, outim5a,X5_Re_Quo, X5_Im_Quo;
reg [7: 0] X5Re, X5Im, X5dRe_jum, X5dIm_jum;
reg [7:0] out_x05,out_x15,out_x25,out_x35,out_x45,out_x55,out_x65,out_x75 ;
reg [8: 0] X5gt_Re9, X5gt_Im9, indivR5, indivI5;
reg [7:0] X5_Im_Rem,X5_Im_Quo5,imag5, X5gt_Im,X5gt_Re,X5_Re_Rem ;
reg [7:0] outre5,outim5 ;
always @ (posedgeclk or negedgerst)
begin
 if (rst == 1'b0) out_x05 <= 8'b00000000 ;
  else out_x05 <= in_x0 ;
  if (rst == 1'b0) out_x15 <= 8'b00000000 ;
  else out_x15 <= in_x1 ;
  if (rst == 1'b0) out_x25 <= 8'b00000000 ;
  else out_x25 <= in_x2 ;
   if (rst == 1'b0) out_x35 <= 8'b00000000 ;
  else out_x35 <= in_x3 ;
   if (rst == 1'b0) out_x45 <= 8'b00000000 ;
  else out_x45 <= in_x4 ;
   if (rst == 1'b0) out_x55 <= 8'b00000000 ;
```

```
  else out_x55 <= in_x5 ;
   if (rst == 1'b0) out_x65 <= 8'b00000000 ;
  else out_x65 <= in_x6 ;
   if (rst == 1'b0) out_x75 <= 8'b00000000 ;
  else out_x75 <= in_x7 ;
 if (rst ==1'b0) outre5<= 0;
  else
 begin
   X5Re <= out_x05 - out_x45;
       X5Im <= out_x25 - out_x65;
x51cd <=9'b010110101 * out_x15;
x53cd <=9'b010110101 * out_x35;
 x55cd <=9'b010110101 * out_x55;
 x57cd <=9'b010110101 * out_x75;
X5dRe <=x55cd + x53cd - x51cd - x57cd;
X5dIm <=-x51cd + x53cd + x55cd - x57cd;
if (X5dRe[7]==1'b1)   X5dRe_jum <= X5dRe[15 :8] +1'b1;
else   X5dRe_jum <= X5dRe[15 :8];
if (X5dIm[7]==1'b1)     X5dIm_jum <= X5dIm[15 :8] +1'b1;
else   X5dIm_jum <= X5dIm[15 :8];
real5 <=X5Re;
imag5 <=X5Im;
re_deci5 <= X5dRe_jum;
im_deci5 <= X5dIm_jum;
X5gt_Re <= X5Re + X5dRe_jum;
X5gt_Im <= X5Im + X5dIm_jum;
X5gt_Re9 <={ X5gt_Re,1'b0 };
 X5gt_Im9 <={ X5gt_Im,1'b0 };
if (X5gt_Re[7]==1'b1)    indivR5 <= 9'b1000_00000 - X5gt_Re9;
else   indivR5 <= X5gt_Re9;
if (X5gt_Im[7]==1'b1)
  indivI5 <= 9'b1000_00000 - X5gt_Im9;
  else   indivI5 <= X5gt_Im9;
if (X5gt_Re[7]==1'b1)      outre5[7:0] <=9'b1000_00000 - X5_Re_Quo;
                        else   outre5 <=X5_Re_Quo [7 :0];
if (X5Im[7]==1'b1)   outim5[7:0] <= 9'b1000_00000- X5_Im_Quo;
else   outim5 <=X5_Im_Quo [7 :0];
X5_Re_Quo5 <= X5_Re_Quo;
 X5_Im_Quo5 <= X5_Im_Quo;

 end
end
lpm_divide ifftpath51 (.numer(indivR5), .denom (d), .quotient (X5_Re_Quo),.remain
(X5_Re_Rem),.clock (clk));
defparam ifftpath51.LPM_WIDTHN = 8;
defparam ifftpath51.LPM_WIDTHD = 4;
lpm_divide ifftpath52 (.numer(indivI5), .denom (d), .quotient (X5_Im_Quo),.remain (X5_Im_Rem),.clock
(clk));
defparam ifftpath52.LPM_WIDTHN = 8;
defparam ifftpath52.LPM_WIDTHD = 4;
///module 6.....
reg [7:0]  X6_Re_Quo6,real6;
reg [7: 0] X6Re, X6Im, X6Re_jum,  X6Im_jum, X6_Re_Quo, X6_Im_Quo;
```

```
reg [7:0] out_x06,out_x16,out_x26,out_x36,out_x46,out_x56,out_x66,out_x76 ;
reg [8: 0] X6Re9, X6Im9, indivR6, indivI6, outre6a, outim6a;
reg [7:0] X6_Im_Rem,X6_Im_Quo6,imag6, X6gt_Im,X6gt_Re,X6_Re_Rem ;
reg [7:0] outre6,outim6 ;
always @ (posedgeclk or negedgerst)
begin
 if (rst == 1'b0) out_x06 <= 8'b00000000 ;
  else out_x06 <= in_x0 ;
  if (rst == 1'b0) out_x16 <= 8'b00000000 ;
  else out_x16 <= in_x1 ;
  if (rst == 1'b0) out_x26 <= 8'b00000000 ;
  else out_x26 <= in_x2 ;
   if (rst == 1'b0) out_x36 <= 8'b00000000 ;
  else out_x36 <= in_x3 ;
   if (rst == 1'b0) out_x46 <= 8'b00000000 ;
  else out_x46 <= in_x4 ;
   if (rst == 1'b0) out_x56 <= 8'b00000000 ;
  else out_x56 <= in_x5 ;
   if (rst == 1'b0) out_x66 <= 8'b00000000 ;
  else out_x66 <= in_x6 ;
   if (rst == 1'b0) out_x76 <= 8'b00000000 ;
  else out_x76 <= in_x7 ;
 if (rst ==1'b0) outre6<= 0;
  else
 begin
          X6Re <= out_x06 + out_x46 - out_x26 - out_x66;  X6Im <= out_x36 + out_x76 - out_x16 -
out_x56;
                    real6 <=X6Re;
                    imag6 <=X6Im;
                     X6Re9 <={1'b0 , X6Re};
                       X6Im9 <={1'b0 , X6Im};
      if (X6Re[7]==1'b1 )     indivR6 <= 9'b1000_00000 - X6Re9;
      else   indivR6 <= X6Re9;
    if (X6Im[7]==1'b1 )   indivI6 <= 9'b1000_00000  - X6Im9;
    else   indivI6 <= X6Im9;
    if (X6Re[7]==1'b1 )  outre6[7 : 0] <=9'b1000_00000  - X6_Re_Quo;
    else   outre6 <=X6_Re_Quo [7 : 0];
    if (X6Im[7]==1'b1 )  outim6[7 : 0] <=9'b1000_00000  - X6_Im_Quo;
    else   outim6 <=X6_Im_Quo [7 : 0];
    X6_Re_Quo6 <= X6_Re_Quo;
    X6_Im_Quo6 <= X6_Im_Quo;
 end
end
lpm_divide ifftpath61 (.numer(indivR6), .denom (d), .quotient (X6_Re_Quo),.remain
(X6_Re_Rem),.clock (clk));
defparam ifftpath61.LPM_WIDTHN = 8;
defparam ifftpath61.LPM_WIDTHD = 4;
lpm_divide ifftpath62 (.numer(indivI6), .denom (d), .quotient (X6_Im_Quo),.remain (X6_Im_Rem),.clock
(clk));
defparam ifftpath62.LPM_WIDTHN = 8;
defparam ifftpath62.LPM_WIDTHD = 4;
// module 7.....
reg [7:0]  X7_Re_Quo7,im_deci7,re_deci7,real7;
```

```
reg [16:0] x71cd, x73cd, x75cd, x77cd, X7dRe, X7dIm;
reg [7: 0] outre7a, outim7a,X7_Re_Quo, X7_Im_Quo;
reg [7:0] X7Re,X7Im,X7dRe_jum,X7dIm_jum ;
reg [7:0] out_x07,out_x17,out_x27,out_x37,out_x47,out_x57,out_x67,out_x77 ;
reg [8: 0] X7gt_Re9, X7gt_Im9, indivR7, indivI7;
// output [8:0] X7gt_Re9, indivR;
reg [7:0] X7_Im_Rem,X7_Im_Quo7,imag7, X7gt_Im, X7gt_Re, X7_Re_Rem ;
 //output [7:0] X7gt_Re;
reg [7:0] outre7,outim7;
always @ (posedgeclk or negedgerst)
begin
 if (rst == 1'b0) out_x07 <= 8'b00000000 ;
   else out_x07 <= in_x0 ;
   if (rst == 1'b0) out_x17 <= 8'b00000000 ;
   else out_x17 <= in_x1 ;
   if (rst == 1'b0) out_x27 <= 8'b00000000 ;
   else out_x27 <= in_x2 ;
    if (rst == 1'b0) out_x37 <= 8'b00000000 ;
   else out_x37 <= in_x3 ;
    if (rst == 1'b0) out_x47 <= 8'b00000000 ;
   else out_x47 <= in_x4 ;
    if (rst == 1'b0) out_x57 <= 8'b00000000 ;
   else out_x57 <= in_x5 ;
    if (rst == 1'b0) out_x67 <= 8'b00000000 ;
   else out_x67 <= in_x6 ;
    if (rst == 1'b0) out_x77 <= 8'b00000000 ;
   else out_x77 <= in_x7 ;
 if (rst ==1'b0) outre7<= 0;
  else
 begin
     X7Re <= out_x07 - out_x47;
     X7Im <= -out_x27 - out_x67;
                    x71cd <=9'b010110101 * out_x17;
                    x73cd <=9'b010110101  * out_x37;
                    x75cd <=9'b010110101  * out_x57;
                    x77cd <=9'b010110101  * out_x77;
      X7dRe <=x71cd + x77cd - x75cd - x73cd;
      X7dIm <=-x1cd + x3cd + x5cd -x7cd;
  if (X7dRe[7]==1'b1)    X7dRe_jum <= X7dRe[15 :8] +1'b1;
 else
   X7dRe_jum <= X7dRe[15 :8] ;
if (X7dIm[7]==1'b1)   X7dIm_jum <= X7dIm[15 :8] +1'b1;
else
 X7dIm_jum <= X7dIm[15 :8] ;
       real7 <=X7Re;
       imag7 <=X7Im;
       re_deci7 <= X7dRe_jum;
       im_deci7 <= X7dIm_jum;
    X7gt_Re <= X7Re + X7dRe_jum; //4
     X7gt_Im <= X7Im + X7dIm_jum;
    X7gt_Re9 <={X7gt_Re, 1'b0};
    X7gt_Im9 <={X7gt_Im, 1'b0};
if (X7gt_Re[7]==1'b1)   indivR7 <=  9'b1000_00000 - X7gt_Re9;
```

```
else
indivR7 <= X7gt_Re9;
if (X7gt_Im[7]==1'b1)    indivI7 <=9'b1000_00000 - X7gt_Im9;
else   indivI7 <= X7gt_Im9;
     if (X7gt_Re[7]==1'b1)  outre7[7 : 0]<= 9'b1000_00000 - X7_Re_Quo;
     else
     outre7 <= X7_Re_Quo[7 : 0];
if (X7Im[7]==1'b1)  outim7[7 : 0]<= 9'b1000_00000 - X7_Im_Quo;
 else   outim7 <=X7_Im_Quo [7 : 0];
X7_Re_Quo7 <= X7_Re_Quo;
X7_Im_Quo7 <= X7_Im_Quo;
 end
end
lpm_divide ifftpath07 (.numer(indivR7), .denom (d), .quotient (X7_Re_Quo),.remain
(X7_Re_Rem),.clock (clk));
defparam ifftpath07.LPM_WIDTHN = 8;
defparam ifftpath07.LPM_WIDTHD = 4;
lpm_divide ifftpath70 (.numer(indivI7), .denom (d), .quotient (X7_Im_Quo),.remain (X7_Im_Rem),.clock
(clk));
defparam ifftpath70.LPM_WIDTHN = 8;
defparam ifftpath70.LPM_WIDTHD = 4;
endmodule
```