Ministry of Higher Education
& Scientific Research
University of Diyala
College of Engineering
Computer & Software
Engineering Department

Design and Analysis of Fixed 8-point FFT core in Verilog –HDL

A project
Submitted to the Department of Computer and Software Engineering University of Diyala-College of Engineering in Partial Fulfillment of the Requirement for the Degree of Bachelor in Computer and Software Engineering

By

Saja Abdul Al-hafiz

Israa Ghazei

Under The Supervision of:

Ahmed K. Jameil

May2016 **Rajab1437**

قال تعالى

(قَالَ رَبِّ اشْرَحْ لِي صَدْرِي (25) وَيَسِّرْ لِي أَمْرِي (26) وَاحْلُلْ عُقْدَةً مِنْ لِسَانِي (27) يَفْقَهُوا قَولي (28)

صدق الله العظيم

(سورة طه)

Dedications

For Our Beloved Families.....

Friends……..

Classmates……

Lecturers……

Thanks and appreciation

At the end of our modest project we would like to extend our thanks and appreciation to" MSc . Ahmed K . Jameil "who performs everything he can do to support us during the research period.

And our thanks and appreciation to all our dear college Professors…May Allah grant bright future for all…

And from Allah the Reconcile.

Students

Supervisors Certification

We certify that this project entitled " Design and Analysis fixed 8-point FFT core in Verilog –HDL ",was prepared under our supervision at the Computer and Software Department/College of Engineering by ((Saja Abd Al-hafiz , Israa Ghazei)) as a partial fulfillment of the requirements for the degree of B.SC. in Computer and Software Engineering .

Signature:

Name :Asst. Lecturer Ahmed K. Jameil

Date: / /2016.

I certify that I have carefully read corrected the final draft of this project for errors in punctuation, spelling and usage.

Proofreader's signature:

Name :

Date: / /2016.

In view of the available recommendation, I forward this project for debate by the examining committee.

Signature:

Name: Dr . Ali J . Abboud

(Head of the Department)

Date: / /2016

Certification of the Examination Committee

We certify that this project entitled "Design and Analysis fixed 8-point FFT core in Verilog –HDL " , and as examining committee examined the students ((Saja Abd Al-hafiz , Israa Ghazei)) in its contents and that in our opinion it meets the standards of a project for the degree of B.SC. In Computer and Software Engineering.

Signature:
Name:

Signature:
Name:

(Member)
Date: / /2016.

(Member)
Date: / /2016.

Signature:
Name:
(Chairman)

Date: / /2016.

Approved for Computer and Software Engineering Department.

Signature:
Name: Dr . Ali J.Abboud
(Head of the Department)
Date: / /2016.

ACKNOWLEDGEMENTS

In the name of Allah, the Most Beneficent, the Most Merciful

We would like to express our heartiest thanks to our supervisor " MSc . Ahmed K . Jameil " for his guidance, patience, advice and Devotion of time, throughout our research. He taught us how to be researchers with his wonderful personality. We have the honor to work under his supervision and we will ask Allah to keep him safe, and support him with good health and the power to help the students with the knowledge and his scientific ability .We are forever indebted to him for excellent guidance.

Finally, to families for their unconditional care until we reach to this point, without their continuous support and precious prayers we could not have been able to finish our research. We know their blessings will always be with us in all our endeavors and we dedicate this success to them .thanks our beloved families.

Studends

Abstract

This project proposes a new architecture of an FFT core that computes radix-2 8-point FFT using fixed-point operation in only eight. This project Fast Fourier Transform has been designed by the Quartus program which produced by Ultra company and this program has provided us with all possibilities for the design of this project to provides interfaces design and provide codes for the design of any circuit and this program make the design of any project easy and inexpensive.

Contents

Chapter One: Introduction

Chapter One: Literature Review

Chapter Three: Methodology

Chapter Four : calculation

Chapter Five : conclusions and future work

List of figures

List of tables

List of abbreviations

Chapter One

Introduction

1.1 History

Verilog was started initially as a proprietary hardware modeling language by Gateway Design Automation Inc. around 1984. It is rumored that the original language was designed by taking features from the most popular HDL language of the time, called HiLo, as well as from traditional computer languages such as C. At that time, Verilog was not standardized and the language modified itself in almost all the revisions that came out within 1984 to 1990. Verilog simulator was first used beginning in 1985 and was extended substantially through 1987. The implementation was the Verilog simulator sold by Gateway.

The first major extension was Verilog-XL, which added a few features and implemented the infamous "XL algorithm" which was a very efficient method for doing gate-level simulation. The time was late 1990. Cadence Design System, whose primary product at that time included thin film process simulator, decided to acquire Gateway Automation System [1][2].

Along with other Gateway products, Cadence now became the owner of the Verilog language, and continued to market Verilog as both a language and a simulator. At the same time, Synopsys was marketing the top-down design methodology, using Verilog. This was a powerful combination. In 1990, Cadence recognized that if Verilog remained a closed language, the pressures of standardization would eventually cause the industry to shift to VHDL.

Consequently, Cadence organized the Open Verilog International (OVI), and in 1991 gave it the documentation for the Verilog Hardware Description Language. This was the event which "opened" the language. OVI did a considerable amount of work to improve the Language Reference Manual (LRM), clarifying things and making the language specification as vendor-independent as possible. Soon it was realized that if there were too many companies in the market for Verilog, potentially everybody would like to do what Gateway had done so far - changing the language for their own benefit. This would defeat the main purpose of releasing the language to public domain.

LRM, several companies began working on Verilog simulators. In 1992, the first of these were announced, and by 1993 there were several Verilog simulators available from companies other than Cadence. The most successful of these was VCS, the Verilog Compiled Simulator, from Chronologic Simulation. This was a true compiler as opposed to an interpreter, which is what Verilog-XL was. As a result, compile time was substantial, but simulation execution speed was much faster.

In the meantime, the popularity of Verilog and PLI was rising exponentially. Verilog as a HDL found more admirers than well-formed and federally funded VHDL. It was only a matter of time before people in OVI realized the need of a more universally accepted standard. Accordingly, the board of directors of OVI requested IEEE to form a working committee for establishing Verilog as an IEEE standard. The working committee 1364 was formed in mid-1993 and on October 14, 1993, it had its first meeting.

As a result in 1994, the IEEE 1364 working group was formed to turn the OVI LRM into an IEEE standard. This effort was concluded with a successful ballot in 1995, and Verilog became an IEEE standard in December 1995. When Cadence gave OVI the standard, which combined both the Verilog language syntax and the PLI in a single volume, was passed in May 1995 and now known as IEEE Std. 1364-1995.

After many years, new features have been added to Verilog, and the new version is called Verilog 2001. This version seems to have fixed a lot of problems that Verilog 1995 had. This version is called 1364-2001.

1.2 Verilog HDL

Verilog is a hardware description language. Although it looks much like C, it is not a software programming language. It is very important for the Verilog programmer to understand hardware concepts. Each line of Verilog code in the design means one or more components in hardware. Verilog is rich in constructs and functionality. Some of the constructs are specific to supporting verification and modeling and do not synthesize to infer hardware. The synthesis is performed using a synthesis tool, which is a compiler that translates Verilog into a gate level design. The synthesis tool understands only a subset of Verilog, the part of Verilog called 'RTL Verilog'. All the other constructs are 'non RTL Verilog'. These constructs are very helpful in testing, verification and simulation [3].

1.3 Problem Statement

Reason of use Radix-2 for radix-2 you have powers [2, 4, 8, 16, 32, 64 …], clearly can expect the distance from N to the next larger power of 2 to be smaller because the powers of 2 are just closer together. The radix that has the smallest distance between successive powers and hence minimizes the possible distance to any N, is 2.

1.4 Objectives

The objectives of this project will be:

1. Configured to executed algorithm in Verilog code.

2. Study on digital system.

3. Processing and Study on digital system design and computer architecture.

4. To extend our knowledge about Quartus Programming.

1.5 Scope of Project

1. The general goal is to create faster design and reduce the computational complexity.

2. The behavior of the system to be designed is specified via algorithm.

3. Reduced storage requirements and reduced computational error.

4. The Comparison of study via implementation and simulation.

1.6 Project layout

This project consists of five chapters that are organizing as follows:

●**Chapter one (Introduction)**: It contains introduction about Verilog, the Objectives, and Scope of project and problem statement.

●**Chapter two** (**Literature Review of the project)** : It contains
Introduction of Quartus II, Review of FFT algorithms and radix-2 and Applications of FFT.

●**Chapter three (Execution of the project and results)** : Its contains the Methodology.

●**Chapter four (calculation)** : Its contains the results of the project.

●**Chapter five (conclusions and future work)**: Close this project by presently conclusions and future work.

Chapter Two

Literature Review

2.1 Introduction of Quartus II

Quartus II development software provides a complete design environment for System m on a Programmable Chip (SOPC) design. Quartus II offers easy design entry (using schematics, block diagrams, AHDL, VHDL and Verilog), powerful logic synthesis, functional and timing simulation, device programming and verification. Quartus II enables analysis and synthesis of HDL designs, which enables the developer to compile their designs, perform timing analysis, examine RTL diagrams, simulate a design's reaction to different stimuli, and configure the target device with the programmer. Quartus includes an implementation of VHDL and Verilog for hardware description, visual editing of logic circuits, and vector waveform simulation. The default Quartus II software graphical user interface can be divided into six main components [5][7]:.

1- Drop-down Menus provides access to all features of Quartus II software.
2- Adjustable Toolbar provides shortcuts to the most frequently used tools.
3-Project Navigator window provides control over project options and source files.
4-Status window provides status information on any process in progress.
5- Editor window provides source file review and editing capabilities.
6-Console window provides design flow related messages.

2.2 Quartus Programming language:-

We will be using Quartus-II software by Altera. This package allows us to write and compile Verilog HDL designs and perform RTL simulation with waveforms. Structural of this program in the design of any project [8]:-

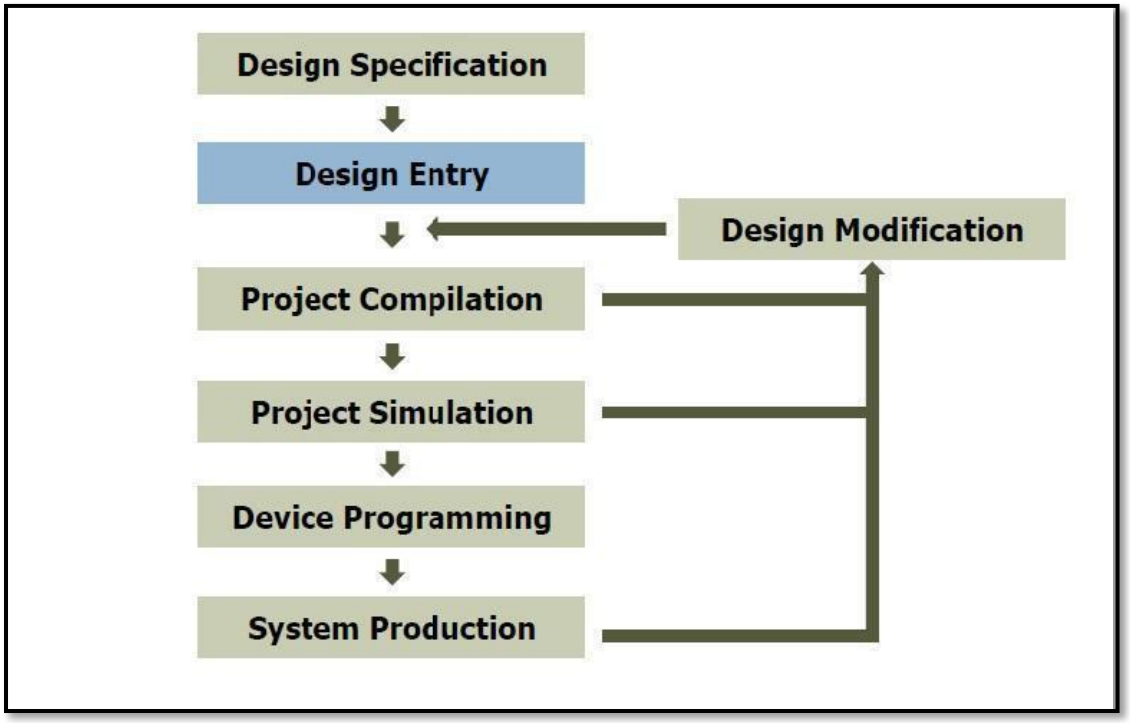


Figure 2-1: Block Diagram Structural of the Design

2.3 Hardware Implementation of Digital Signal Processing Algorithms.

As the technology advances, more complicated systems emerge, requiring sophisticated algorithms often encompassing very complex digital signal processing (DSP) techniques. Although the performance of programmable processors is continuously improving, hardware implementation of DSP algorithms is increasingly required in several areas such as wireless communications, multimedia systems, computer networks, and biomedicine [6].

There are several challenges that engineers face while designing hardware DSP algorithms. These challenges are heightened by the needs of huge computing power (required in real-time systems), reducing energy consumption (for portable, battery-operated systems) and reducing costs. Technology scaling adds additional difficulties related to variability of device parameters, leakage currents, and signal integrity. Therefore, hardware implementation of DSP algorithms has gained much attention during past years and is the focus of this special issue.

The application of hardware implementation of digital signal processing algorithms is extended from communication systems, digital filter design, and image and video processing applications to implementation of complex mathematical procedures for data analysis. The need to use hardware implementations of digital signal processing algorithms is exponentially increasing due to the explosion of stored data and the necessity of analyzing these data in less amount of time. This goal cannot simply be accomplished by computer software because they are running in operating systems hence lower processing speed [6].

2.4 Review of FFT algorithms:

The basic principle behind most Radix based FFT algorithms is to exploit the symmetry properties of a complex exponential that is the cornerstone of the Discrete Fourier Transform (DFT), These algorithms divide the problem into similar sub-problems (butterfly computations), and achieve a reduction in computational complexity. All Radix algorithms are similar in structure differing only in the core computation of the butterflies [10].

2.5 Radix-2

The 'Radix 2' algorithms are useful if N is a regular power of 2 (N=2p). If we assume that algorithmic complexity provides a direct measure of execution time and that the relevant logarithm base is 2, ratio of execution times for the (DFT) vs. (Radix 2 FFT) (denoted as 'Speed Improvement Factor') increases tremendously with increase in N.

The term 'FFT' is actually slightly ambiguous, because there are several commonly used 'FFT' algorithms. There are two different Radix 2 algorithms, the so-called 'Decimation in Time' (DIT) and 'Decimation in Frequency' (DIF) algorithms. Both of these rely on the recursive decomposition of an N point transform into 2 (N/2) point transforms. The computation involving each pair of data is called a butterfly [11]. Radix-2 algorithm can be implemented as Decimation-in-time (M=N/2 and L=2) or Decimation-in-frequency (M=2 and L=N/2) algorithms.

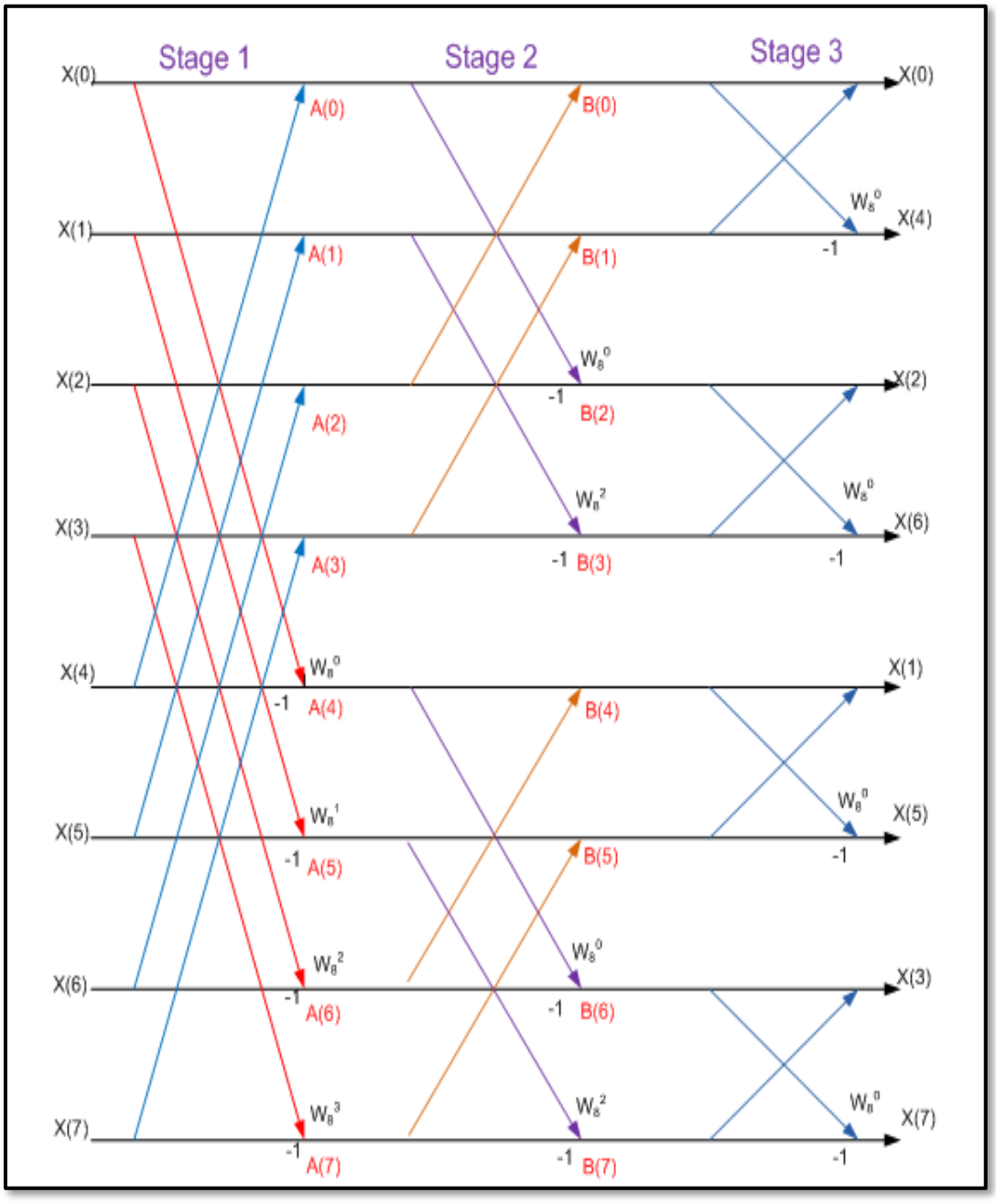


Figure 2-2: 8-point Radix-2 FFT: Decimation in frequency form

To divide N-point sequence x(n) into two N/2-point sequence :

The former N/2-point $x(n), \quad 0 \le n \le \frac{N}{2}-1$

The latter N/2-point $x(n+\frac{N}{2}), \quad 0 \le n \le \frac{N}{2}-1$

To compute the DFT of N-point sequence x (n)

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

The above equation can be simplified to

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

$$=\sum_{n=0}^{\frac{N}{2}-1}\left[x(n)+W_N^{\frac{N}{2}k}x(n+\frac{N}{2})\right]W_N^{nk} \qquad (3)$$

$$=\sum_{n=0}^{\frac{N}{2}-1}\left[x(n)+(-1)^k x(n+\frac{N}{2})\right]W_N^{nk} \quad (k=0,1,\cdots N-1) \qquad (4)$$

To separate the even and odd numbered samples of X (k)

let $k = 2r, \quad k = 2r+1, \quad (r = 0,1,\cdots,\frac{N}{2}-1)$

$$X(2r) = \sum_{n=0}^{\frac{N}{2}-1}\left[x(n) + x(n+\frac{N}{2})\right]W_{\frac{N}{2}}^{nr} \quad (r = 0,1,\cdots\frac{N}{2}-1) \tag{5}$$

$$X(2r+1) = \sum_{n=0}^{\frac{N}{2}-1}\left[x(n) - x(n+\frac{N}{2})\right]W_N^n W_{\frac{N}{2}}^{nr} \quad (r = 0,1,\cdots\frac{N}{2}-1) \tag{6}$$

let $\begin{cases} x_1(n) = x(n) + x(n+\frac{N}{2}) \\ x_2(n) = \left[x(n) - x(n+\frac{N}{2})\right]W_N^n \end{cases}$ $n = 0,1,\cdots\frac{N}{2}-1$

$$X(2r) = \sum_{n=0}^{\frac{N}{2}-1} x_1(n)W_{\frac{N}{2}}^{nr} \qquad (r = 0,1,\cdots\frac{N}{2}-1) \tag{7}$$

$$X(2r+1) = \sum_{n=0}^{\frac{N}{2}-1} x_2(n)W_{\frac{N}{2}}^{nr} \quad (r = 0,1,\cdots\frac{N}{2}-1) \tag{8}$$

2.6 Applications of FFT

FFT's importance derives from the fact that in signal processing and image processing it has made working in frequency domain equally computationally feasible as working in temporal or spatial domain. Some of the important applications of FFT includes [13].

1. Fast large integer and polynomial multiplication.
2. Efficient matrix-vector multiplication for Toeplitz , circulant and other structured matrices.
3. Filtering algorithms.
4. Fast algorithms for discrete cosine or sine transforms (example, Fast DCT used for JPEG, MP3/MPEG encoding).
5. Fast Chebyshev approximation.
6. Fast Discrete Hartley Transform.
7. Solving Difference Equations.

Chapter Three

Methodology

3.1 Introduction of Fast Fourier transform

Direct computation of an N-point DFT requires nearly complex arithmetic operations. An arithmetic operation implies a multiplication and an addition. However, this complexity can be significantly reduced by developing efficient algorithms. These algorithms are denoted as FFT (fast Fourier transform) algorithms. Several techniques are developed for the FFT. We will initially develop the decimation-in-time (DIT) and decimation-in-frequency (DIF) FFT algorithms. The detailed development will be based on radix-2.

This will then be extended to other radices such as radix-3, radix-4, etc. The reader can then foresee that innumerable combinations of fast algorithms exist for the FFT, i.e., mixed-radix, split-radix, DIT, DIF, DIT/DIF, vector radix, vector-split-radix, etc. In general the fast algorithms reduce the computational complexity of an N-point DFT to about Nlog2N complex arithmetic operations. Additional advantages are reduced storage requirements and reduced computational error due to finite bit length arithmetic (multiplication/division, and addition/subtraction, to be practical, are implemented with finite word lengths) [19].

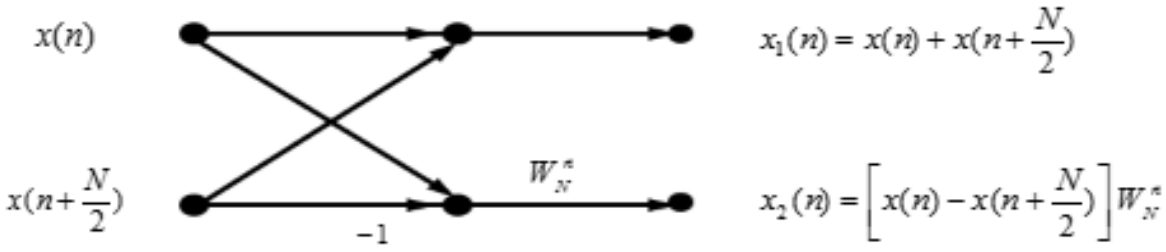
3.2 Butterfly unit

Signal-flow graph connecting the inputs x (left) to the outputs y that depend on them (right) for a "butterfly" step of a radix-2 Cooley–Tukey FFT. This diagram resembles a butterfly (as in the Morpho butterfly shown for comparison), hence the name.

In the context of fast Fourier transform algorithms, a butterfly is a portion of the computation that combines the results of smaller discrete Fourier transforms (DFTs) into a larger DFT, or vice versa (breaking a larger DFT up into sub transforms). The name "butterfly" comes from the shape of the data-flow diagram in the radix-2 case.

Most commonly, the term "butterfly" appears in the context of the Cooley–Tukey FFT algorithm, which recursively breaks down a DFT of composite size n = rm into r smaller transforms of size m where r is the "radix" of the transform. These smaller DFTs are then combined via size-r butterflies, which themselves are DFTs of size r (performed m times on corresponding outputs of the sub-transforms) pre-multiplied by roots of unity (known as twiddle factors). (This is the "decimation in time" case; one can also perform the steps in reverse, known as "decimation in frequency", where the butterflies come first and are post-multiplied by twiddle factors. See also the Cooley–Tukey FFT article.) [17].

Butterfly computation flow graph



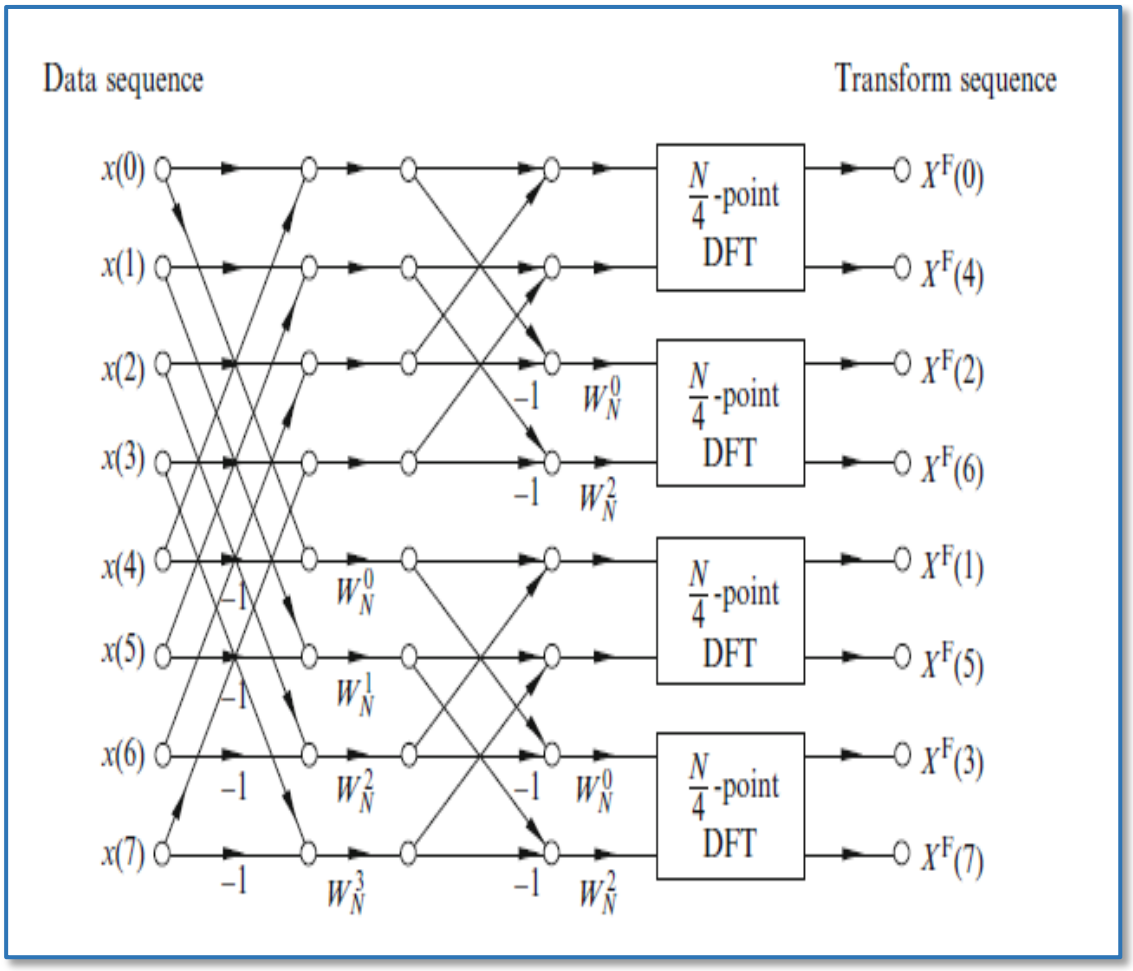


Figure 3-1: Flow graph of decimation-in frequency decomposition of an eight-point DFT into two-point DFT computations. $W_N = W_8$

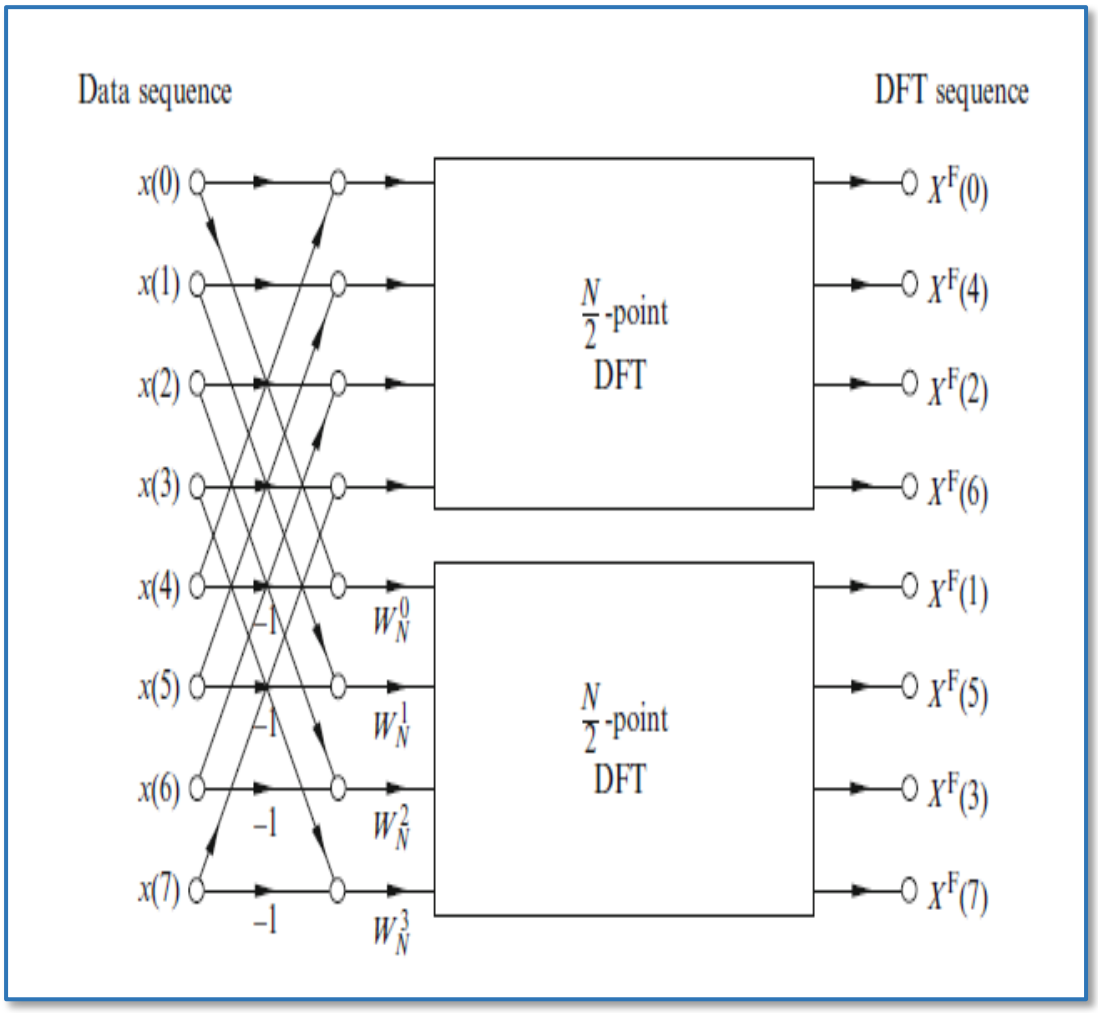


Figure 3-2: Flow graph of decimation-in frequency decomposition of an N-point DFT into two N=2-point DFT computations (N=8) ($W_N = W_8$)

Figure (3-3) shows a design for 8-point decimation in a frequency FFT algorithm that folds three stages of eight radix 2 butterflies on to three butterflies. The pipeline registers are appropriately placed to fully utilize all the butterflies in all clock cycles.

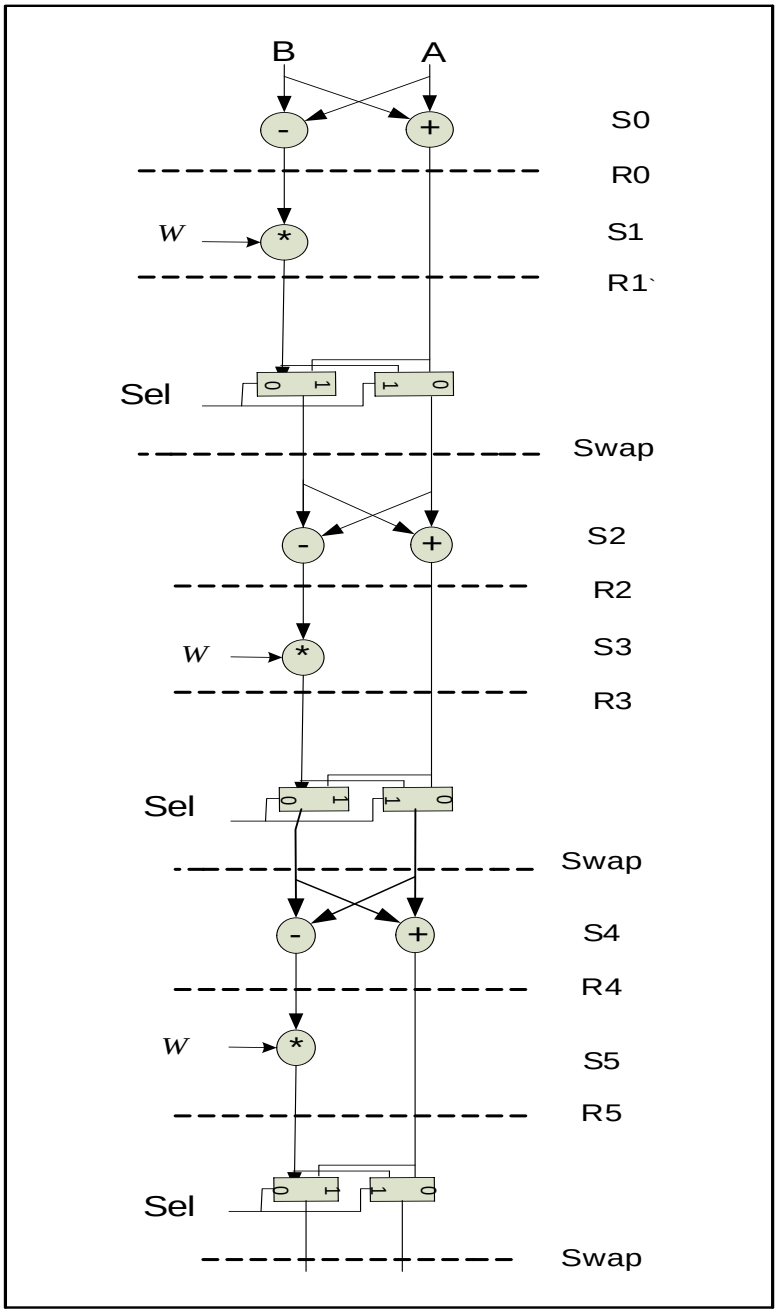


Figure 3-3: Architecture implementing 8 point Decimation in frequency FFT algorithm

For memory based implementation of an N point radix 2 FFT algorithm, one or multiple butterfly units can be placed in the data path. Each butterfly is connected to two dual ported memories and a coefficient ROM for providing input data to the butterfly [9]. The outputs of the butterflies are appropriately saved in memories to be used for the next stages of execution. The main objective of the design is to place input and output data in a way that two corresponding inputs to execute a particular butterfly are always stored in two different dual port memories. The memory management should be done in a way that avoids access conflicts.

Figure 3-3 shows the details of the design for a systolic FFT implementation. The data is serially fed to PE1 and the output is stored in registers for first two computations of butterfly operations. These outputs are then used for the next two set of computation.

The first column shows the cycles where the design repeats computation after every fourth cycle and starts computing the 8 point FFT of a new sequence of 8 data points. The second column shows the data being serially fed to the design. The first cycle takes x0 and x4, and then in each subsequent cycle two data values are fed as shown in this column. The columns labeled as PE1, PE2 and PE3 show the butterfly computations. The intermediate values stored in registers R0, R1, R2 and R3.

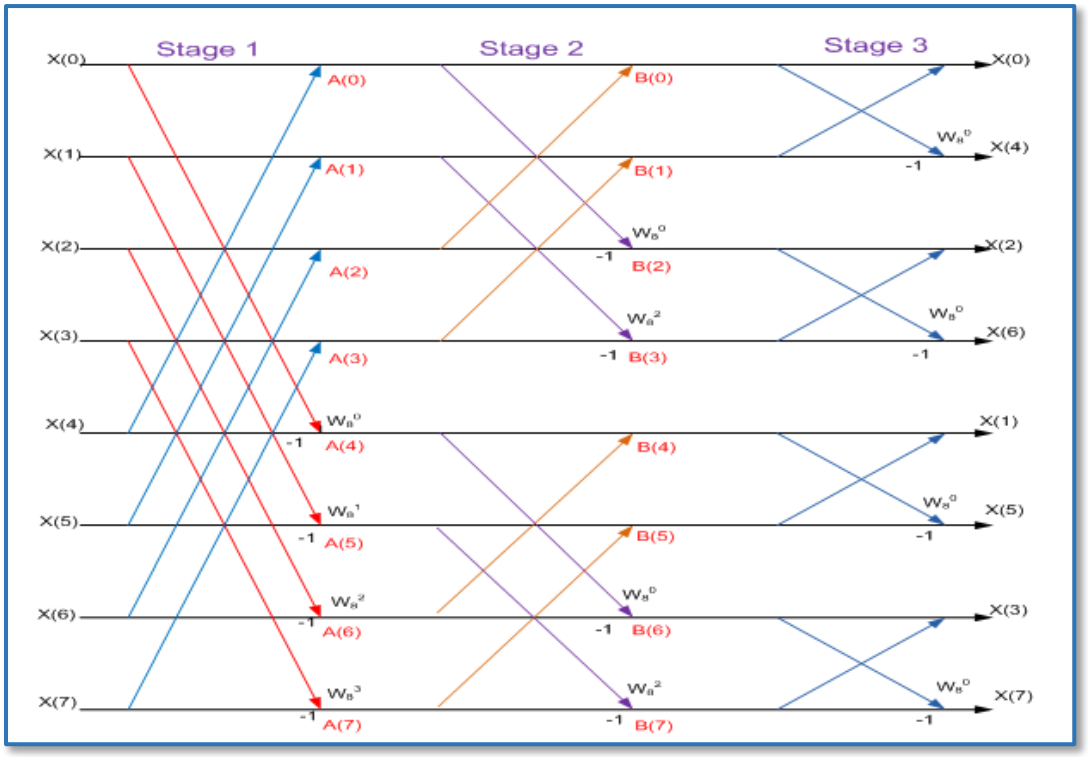


Figure 3-4: 8-point Radix-2 FFT: Decimation in frequency form

.W_N = Phase roration factor = $e^{-j2\pi n/N}$

Table 3-1: Twiddle Factor value for FFT

FFT(N=8)		
S/N	W	Value
0	W_8^0	1
1	W_8^1	0.7071-j0.7071
2	W_8^2	-j
3	W_8^3	-0.7071-j0.7071

3.3 Direct Mathematical Method of 8-point FFT.

1- X(n)={1 1 1 1 0 0 0 0}

Stage 1

$A_0 = x(0) + x(4) = 1 + 0 = 1$
$A_1 = x(l) + x(5) = 1 + 0 = 1$
$A_2 = x(2) + x(6) = 1 + 0 = 1$
$A_3 = x(3) + x(7) = 1+0 = 1$
$A_4 = [x(0) - x(4)]* W_8^0 =[1-0] * 1=1$
$A_5 = [x(1) - x(5)]*W_8^1 =[1-0] * 0.7071-j0.7071 = 0.7071-j0.7071$
$A_6 = [x(2) - x(6)]*W_8^2 = [1-0] * -j = -j$
$A_7 = [x(3) - x(7)]*W_8^3 = [1-0] * -0.7071-j0.7071 =- 0.7071-j0.7071$

Stage 2

$B_0 = A_0 + A_2 = 1 + 1 = 2$
$B_1 = A_1 + A_3 = 1 + 1 = 2$
$B_2 =[A_0 - A_2]* W_8^0= [1-1]*1=0$
$B_3 = [A_1 - A_3]* W_8^2= [1-1]* -j =0$
$B_4 = A_4 + A_{6=} 1-j$
$B_5 = A_5 + A_{7=} -j1.4142$
$B_6 = [A_4 - A_6]* W_8^0 = 1+j$
$B_7 = [A_5 - A_7]* W_8^2 = -j1.4142$

Stage 3

$X(0)= B_0 + B_{1=} 2+2=4$
$X(4)= B_0 - B_{1=} 2-2=0$
$X(2)= B_2 + B_{3=} 0+0=0$
$X(6)= B_2 - B_{3=} 0-0=0$
$X(1)= B_4 + B_{5=} [(1-j) + (-j1.4142)]= 1-j2.4142$
$X(5)= B_4 - B_{5=} [(1-j) - (-j1.4142)]= -0.4142-j$
$X(3)= B_6 + B_{7=}[(1+j) +(-j1.4142)]= 1-j0.4142$
$X(7)= B_6 - B_{7=}[(1+j) - (-j1.4142)]= 1+j2.4142$

Table 3-2: for sequences 1 of Computation of Stages of butterfly

	Stage 1	Stage 2	Stage 3
X(0)=1	A(0)=1	B(0)=2	X(0)=4
X(1)=1	A(1)=1	B(1)=2	X(4)=0
X(2)=1	A(2)=1	B(2)=0	X(2)=0
X(3)=1	A(3)=1	B(3)=0	X(6)=0
X(4)=0	A(4)=1	B(4)=1-j	X(1)=1-j2.4142
X(5)=0	A(5)=0.7071-J0.7071	B(5)=-j1.4142	X(5)=-0.4142-j
X(6)=0	A(6)=-J	B(6)=1+j	X(3)=1-j0.4142
X(7)=0	A(7)=-0.7071-J0.7071	B(7)=-j1.4142	X(7)=1+j2.4122

1- X(0)={00001111}

Stage 1

$A_0 = x(0) + x(4) = 0 + 1 = 1$

$A_1 = x(l) + x(5) = 0 + 1 = 1$

$A_2 = x(2) + x(6) = 0 + 1 = 1$

$A_3 = x(3) + x(7) = 0+1 = 1$

$A_4 = [x(0) - x(4)]* W_8^0 = [0-1] * 1 = -1$

$A_5 = [x(1) - x(5)]*W_8^1 = [0-1] * 0.7071-j0.7071 = -0.7071+j0.7071$

$A_6 = [x(2) - x(6)]*W_8^2 = [0-1] * -j = j$

$A_7 = [x(3) - x(7)]*W_8^3 = [0-1] * -0.7071-j0.7071 = 0.7071+j0.7071$

Stage 2

$B_0 = A_0 + A_2 = 1 + 1 = 2$

$B_1 = A_1 + A_3 = 1 + 1 = 2$

$B_2 = [A_0 - A_2]* W_8^0 = [1-1]*1=0$

$B_3 = [A_1 - A_3]* W_8^2 = [1-1]* -j = 0$

$B_4 = A_4 + A_6 = -1+j$

$B_5 = A_5 + A_7 = j1.4142$

$B_6 = [A_4 - A_6]* W_8^0 = -1-j$

$B_7 = [A_5 - A_7]* W_8^2 = j1.4142$

Stage 3

$X(0) = B_0 + B_1 = 2+2=4$

$X(4) = B_0 - B_1 = 2-2=0$

$X(2) = B_2 + B_3 = 0+0=0$

$X(6) = B_2 - B_3 = 0-0=0$

$X(1) = B_4 + B_5 = [(-1+j) + (j1.4142)] = -1+j2.4142$

$X(5) = B_4 - B_5 = [(-1+j) - (j1.4142)] = 0.4142+j$

$X(3) = B_6 + B_7 = [(-1-j) + (j1.4142)] = -1+j0.4142$

$X(7) = B_6 - B_7 = [(-1-j) - (j1.4142)] = -1-j2.4142$

Table 3-3: for sequences 2 of Computation of Stages of butterfly

X(n)	Stage 1	Stage 2	Stage 3
X(0)=0	A(0)=1	B(0)=2	X(0)=4
X(1)=0	A(1)=1	B(1)=2	X(4)=0
X(2)=0	A(2)=1	B(2)=0	X(2)=0
X(3)=0	A(3)=1	B(3)=0	X(6)=0
X(4)=1	A(4)=-1	B(4)=-1+j	X(1)=-1+j2.4142
X(5)=1	A(5)=-0.7071+J0.7071	B(5)=j1.4142	X(5)=0.4142+j
X(6)=1	A(6)=J	B(6)=-1-j	X(3)= -1+j0.4142
X(7)=1	A(7)=0.7071+J0.7071	B(7)=j1.4142	X(7)=-1-j2.4122

2- X(n)={ 1 1 1 1 1 1 1 1}

Stage 1

$A_0 = x(0) + x(4) = 1 + 1 = 2$
$A_1 = x(l) + x(5) = 1 + 1 = 2$
$A_2 = x(2) + x(6) = 1 + 1 = 2$
$A_3 = x(3) + x(7) = 1+1 = 2$
$A_4 = [x(0) - x(4)]* W_8^0 =[1-1] * 1=0$
$A_5 = [x(1) - x(5)]*W_8^1 =[1-1] * 0.7071-j0.7071 =0$
$A_6 = [x(2) - x(6)]*W_8^2 = [1-1] * -j = 0$
$A_7 = [x(3) - x(7)]*W_8^3 = [1-1] * -0.7071-j0.7071 = 0$

Stage 2

$B_0 = A_0 + A_2 = 2 + 2 = 4$
$B_1 = A_1 + A_3 = 2 + 4 = 4$
$B_2 =[A_0 - A_2]* W_8^0= [2-2]*1=0$
$B_3 = [A_1 - A_3]* W_8^2= [2-2]* -j =0$
$B_4 = A_4 + A_{6=} 0$
$B_5 = A_5 + A_{7=} 0$
$B_6 = [A_4 - A_6]* W_8^0 = 0$
$B_7 = [A_5 - A_7]* W_8^2 = 0$

Stage 3

$X(0)= B_0 + B_1 =4+4=8$
$X(4)= B_0 - B_{1=} 4-4=0$
$X(2)= B_2 + B_{3=} 0+0=0$
$X(6)= B_2 - B_{3=} 0-0=0$
$X(1)= B_4 + B_{5=} [0 + 0]= 0$
$X(5)= B_4 - B_{5=} [0 - 0]= 0$
$X(3)= B_6 + B_{7=}[0+0]= 0$
$X(7)= B_6 - B_{7=}[0 - 0]= 0$

Table 3-4: for sequences 2 of Computation of Stages of butterfly

X(n)	Stage 1	Stage 2	Stage 3
X(0)=1	A(0)=2	B(0)=4	X(0)=8
X(1)=1	A(1)=2	B(1)=4	X(4)=0
X(2)=1	A(2)=2	B(2)=0	X(2)=0
X(3)=1	A(3)=2	B(3)=0	X(6)=0
X(4)=1	A(4)=0	B(4)=0	X(1)=0
X(5)=1	A(5)=0	B(5)=0	X(5)=0
X(6)=1	A(6)=0	B(6)=0	X(3)=0

Chapter Four
Calculations

4.1 Introduction

The simulation of this whole project has been done using the Quartus of version Quartus II 8.1 Web Edition It provides a comprehensive simulation and debug environment Support is provided for multiple languages including Verilog, System Verilog and VHDL [7].

The FFT computation is accomplished in three stages. The x (0) until x (7) variables are denoted as the input values for FFT computation and X (0) until X (7) are denoted as the outputs. The pipeline architecture of the 8-point FFT is shown in Fig 4.1 consisting of butterfly schemes in it. There are two operations to complete the computation in each stage .The implementation of FFT flow graph in the VHDL requires three stages, final computation is done and the result is sent to the variable Y (0) to Y (7). Equation in each stage is used to construct scheduling diagram.

Processing element (PE) $=\text{Log}_2 N$

PE$= \text{Log}_2 (8) = 3$

Samples of input data $= X_{N/2-1}$

IF $X_{8/2-1} = X_3$

Input data $= X_0 \longrightarrow X_3$

IF N=8 then $(X_{N-1}) = X_7$

$X_{N/2} = X_4$

$X_{N/2+1} = X_5$

$X_{N/2+2} = X_6$

$X_{N/2+3} = X_7$

First stream $X_0, X_1 \ldots X_{N/2-1}$

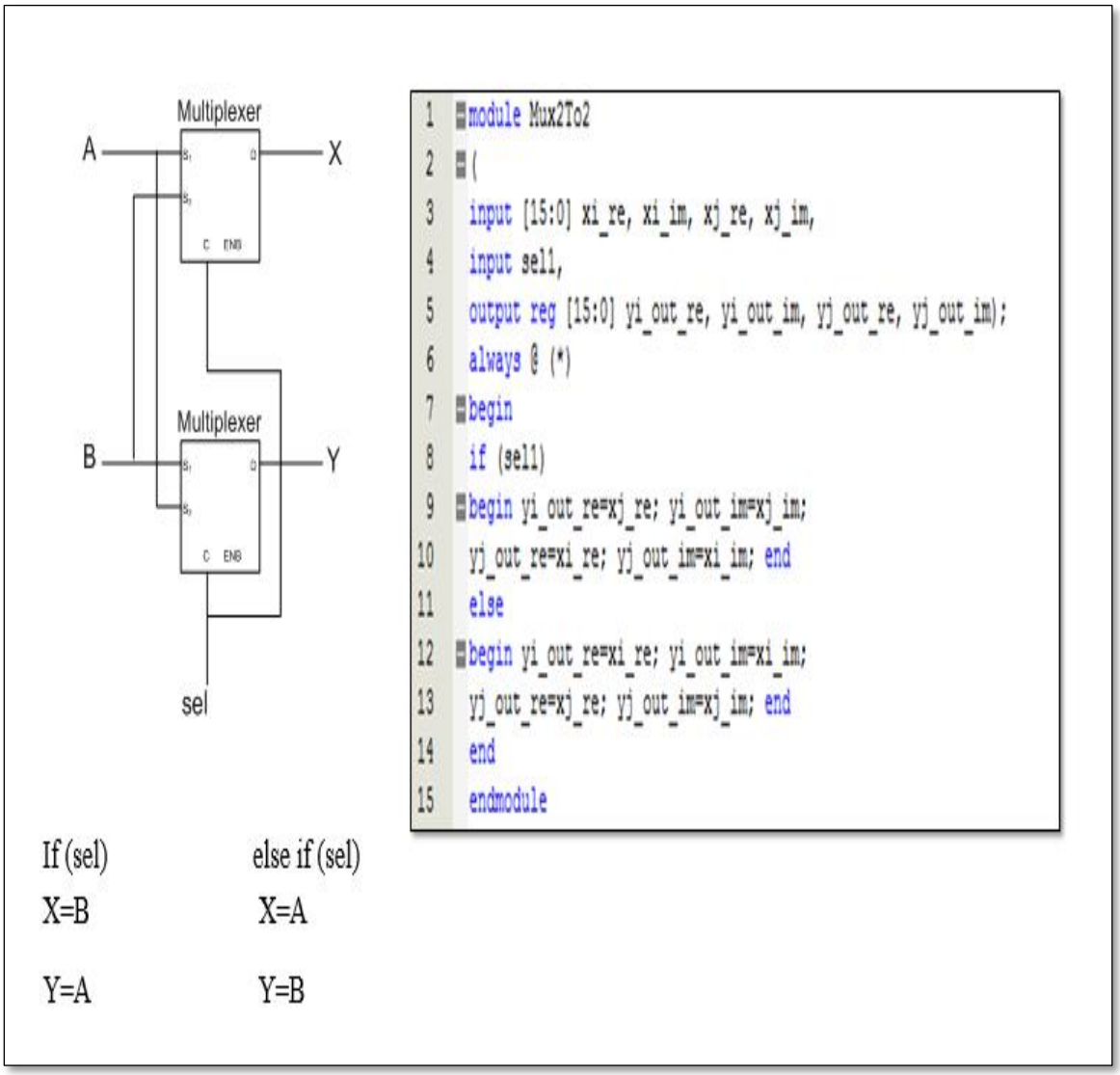
Second stream $X_{N/2}, X_{N/2+1 \ldots} X_{N-1}$

Every stage requires register at each input = $N/2^S$

Where s is the stage of butterfly computation S = 1, 2, 3… $Log_2 N$.

Implementing decimation infrequency FFT computation, the design initially stores N/2 points of input data, X_0 , X_1,……., $X_{N/2-1}$in RAM0, and then with the arrival of every new sample, $X_{N/2}$, $X_{N/2+1}$, ……., X_{N-1}executes the following butterfly computations Stored in RAM1 and RAM0, respectively. This arrangement enables the next stage of Computation without any memory conflicts.

Intermediate registers and multiplexers are placed such that the design keeps processing data for computation of FFT and correct inputs are always available to every PE in every clock cycle for full utilization of the hardware. The multiplexers ensure the correct set of data are fed to PE1 and PE2 in every clock cycle.




```
1  module Mux2To2
2  (
3   input [15:0] xi_re, xi_im, xj_re, xj_im,
4   input sel1,
5   output reg [15:0] yi_out_re, yi_out_im, yj_out_re, yj_out_im);
6   always @ (*)
7  begin
8   if (sel1)
9  begin yi_out_re=xj_re; yi_out_im=xj_im;
10  yj_out_re=xi_re; yj_out_im=xi_im; end
11  else
12 begin yi_out_re=xi_re; yi_out_im=xi_im;
13  yj_out_re=xj_re; yj_out_im=xj_im; end
14  end
15  endmodule
```

Figure 4-1: Swap operation used multiplexer and code in Verilog.

In every stage of FFT computation, the next stage swaps the storage of results for every two sets of outputs in memories RAM_0 and RAM_1. The design swaps the storage pattern for every $N/2^S$ output samples, Where s is the stage of butterfly computation S = 1, 2, 3… Log_2 N.

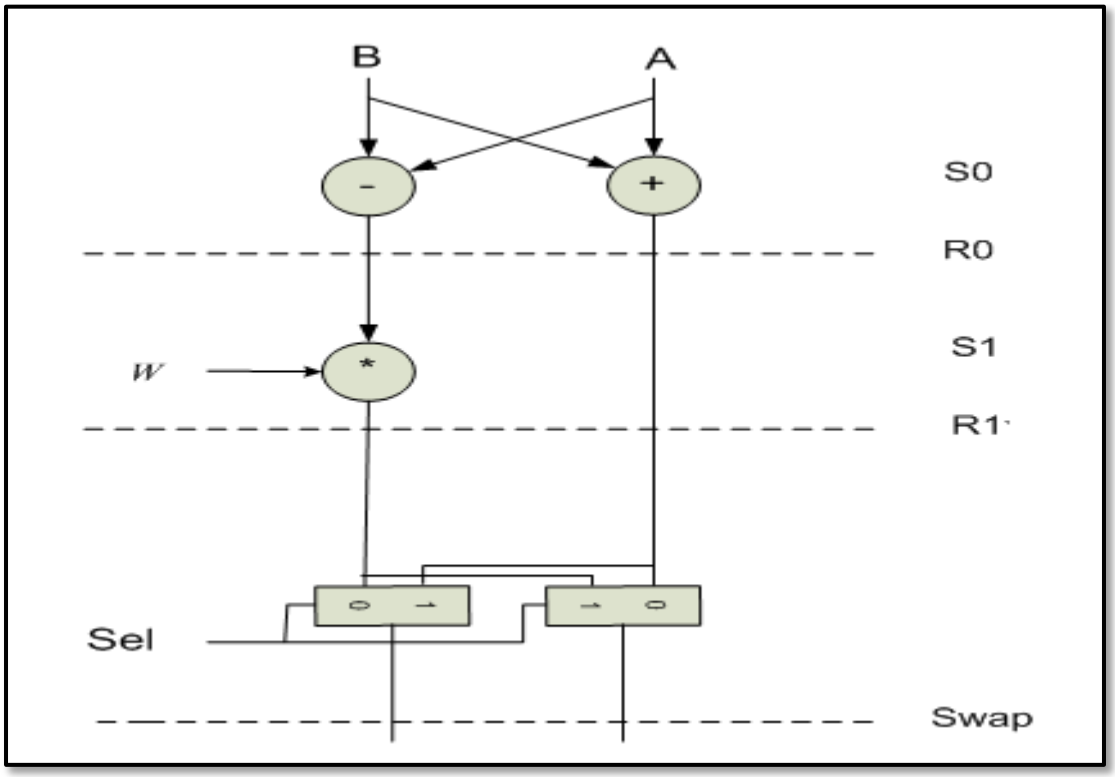


Figure 4-2: Architecture implementing of first processing element of 8-point decimation in frequency FFT algorithm

From the calculations, we got this results Figure 4-3 the technology map viewer schematic of complete FFT 8-point module.

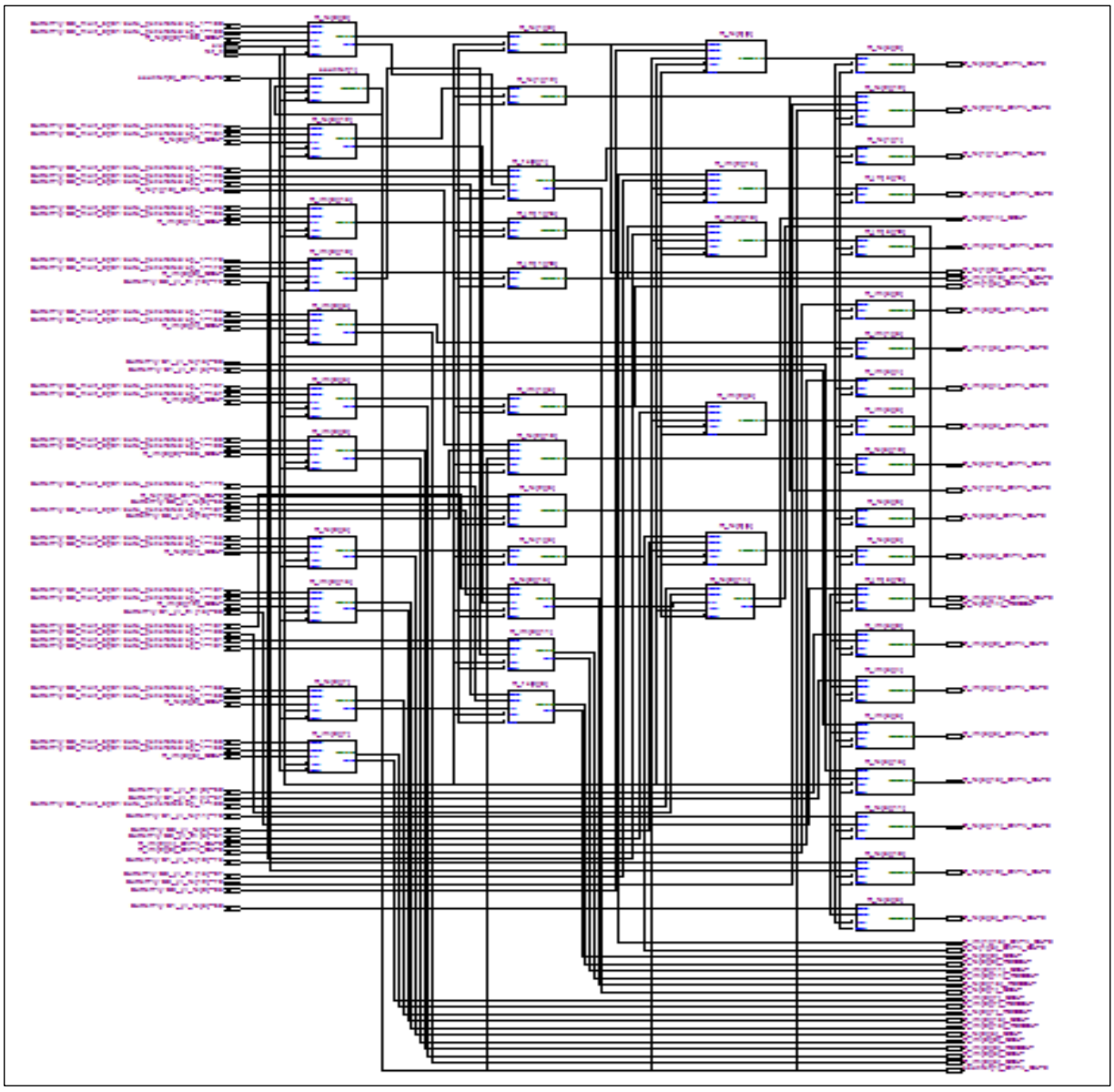


Figure 4-3: The technology map viewer schematic of complete FFT 8-point module.

Figure (4-4) shows the simulation activity for fast Fourier transform, and displays the schedule of the process that concurrently executing of 8-point FFT.

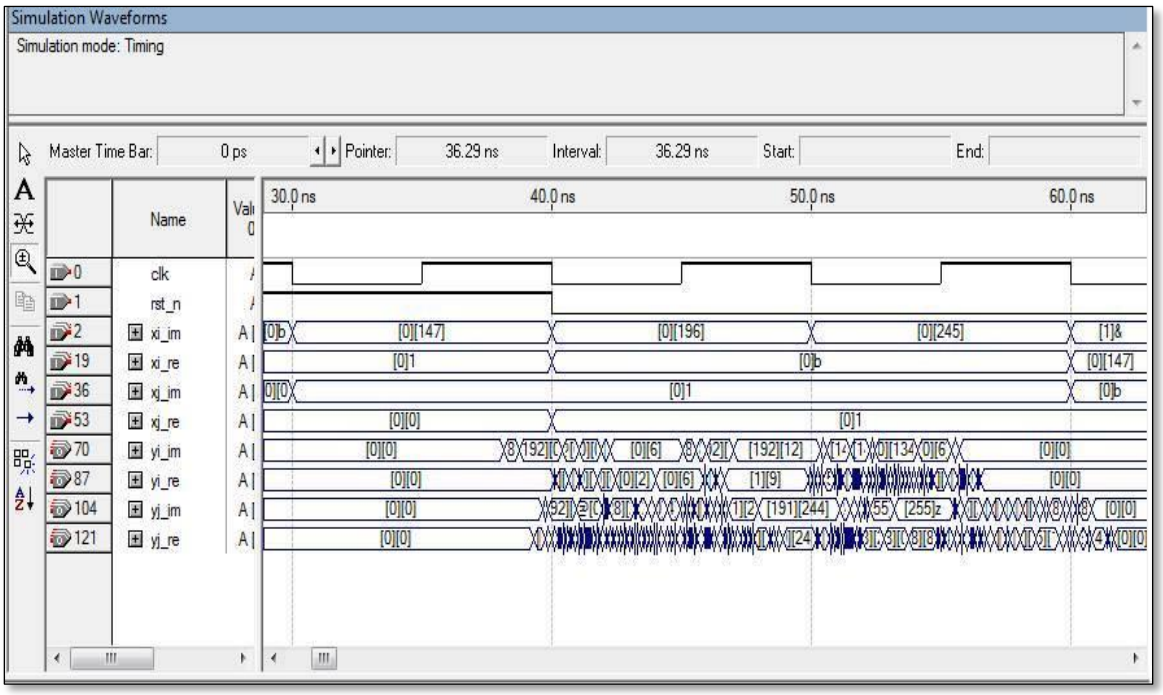


Figure 4-4: simulation waveforms of 8-point fast Fourier transform

One or multiple butterfly units can be placed in the data path. Each butterfly is connected to two dual ported memories and a coefficient ROM for providing input data to the butterfly. The outputs of the butterflies are appropriately saved in memories to be used for the next stages of execution. Figure 4-5 show the RTL of butterfly of FFT 8-point module.

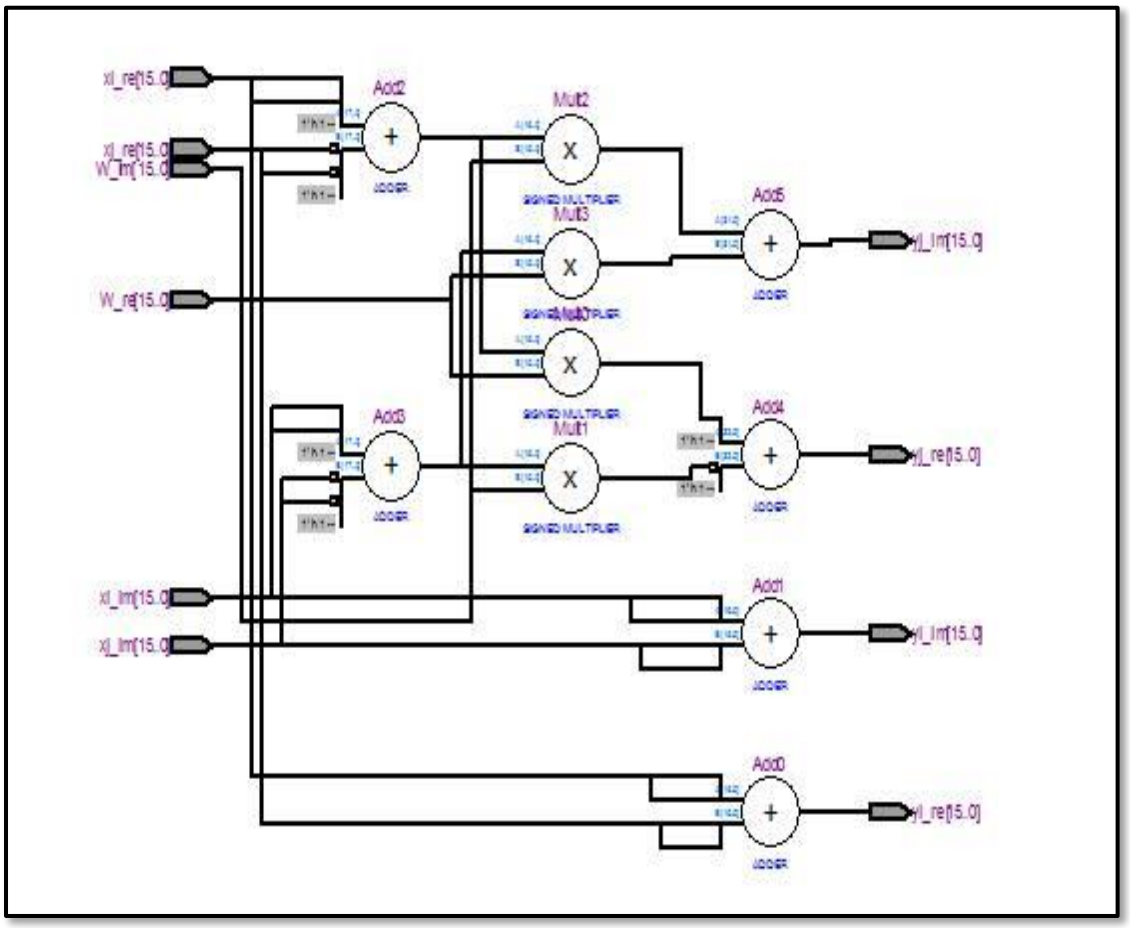


Figure 4-5: RTL of butterfly of FFT 8-point module.

And here is the operation adder utilize.

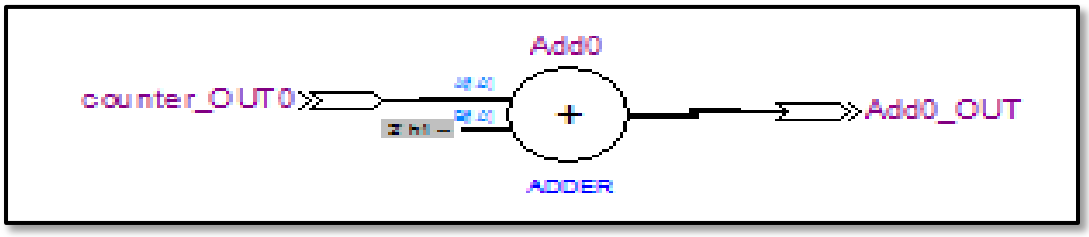


Figure 4-6: Operation Adder

The multiplexers ensure the correct set of data are fed to processing element in every clock cycle. And show circuit multiplexer in code Verilog in figure 4-7.

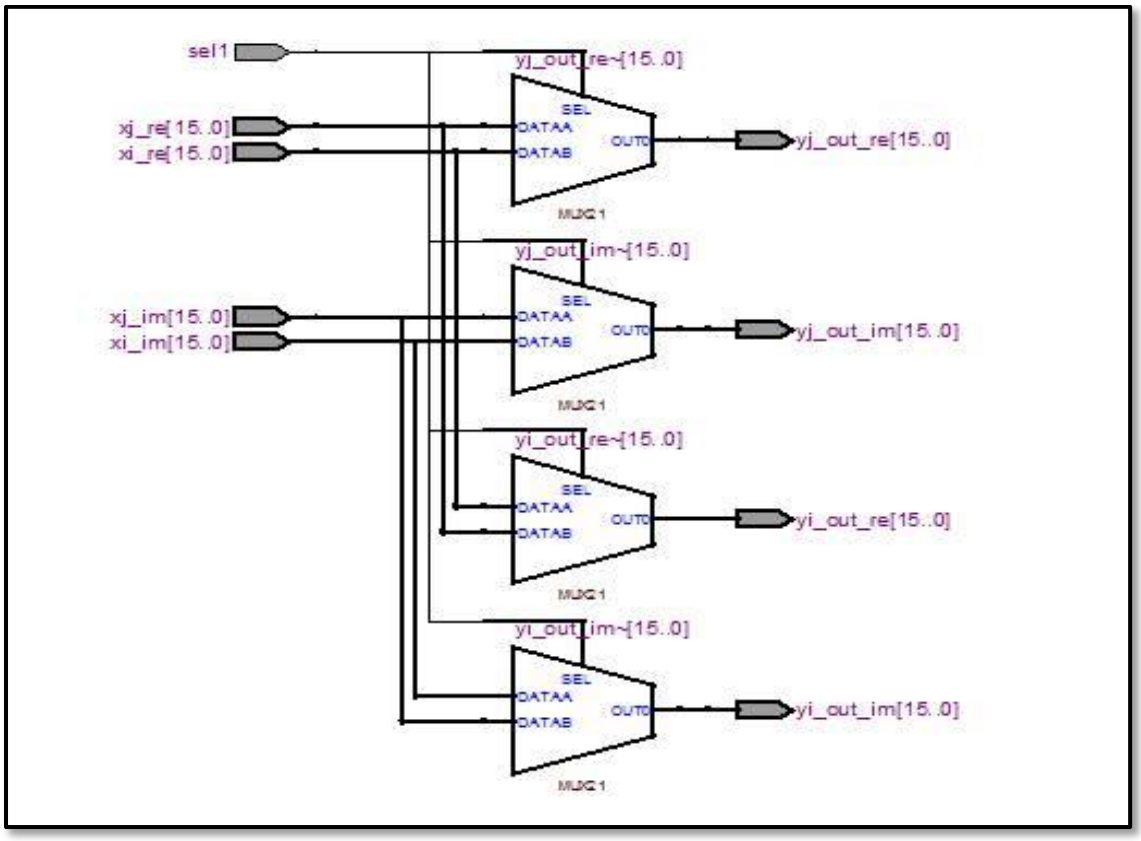


Figure 4-7: Internal Structure of circuit multiplexer

Chapter five

Conclusion & Future work

5.1 Conclusion

This project regard to the implementation of The Fast Fourier transform algorithm Based on Decimation-In- Frequency (DIF) Radix-2 algorithm and 8-points computation, which have been programmed using Quartus II with program language Verilog HDL treatment , Where we have the analysis of the architectural FFT and implementation in VHDL and account Direct Mathematical Method of 8-point FFT . In conclusion, the main objective of this project has been successfully accomplished and the result obtained from this work is verified.

5.2 Future Work

The following points can be considered as a future work:

1) This design can be modified to accept complex number, that this design can accept complex number as inputs.

2) Use higher fixes point representation for point value representation. In real time applications and system that required high precision and high speed, 8 points computation is not suitable.

Therefore the higher N-points such 16-point, 32-point and 64-point should be design to comply this need.

3) Use higher radix algorithm to reduce the real multiplications and real additions. When the number of data point N in the DFT is increasing, radix-2 algorithm still can be used for the computation. However, to achieve shorter time delay, it is more efficient computationally to employ a higher radix such as Radix-4 FFT and Radix-8 FFT algorithm.

REFERENCES

[1] *Digital systems VHDL & Verilog design , Dr. mohamed khalil hani, UTM.*

[2] *Stephen Brown & Zvonko Vranesic, Fundamentals of Digital Logic*
with
Verilog Design, McGraw-Hill, 2004

[3] *Advance Digital Design with Verilog HDL - Michael D. Cilitte. Aug 13, 2002*

[4] *computer organization design the hardware/software interface. David A. Patterson, John L. Hennessy,2005*

[5] *Designing Digital Computer Systems with Verilog, David J. Lilja and Sachin S. Sapatnekar,2005*

[6] *Digital Signal processing algorithms and applications. I.Pitas.*

[7] *Jose S.(2004) "Quartus II Handbook, Volume 1 Design & Synthesis".*

[8] *Zainalabedin Navabi(2006), "Verilog Digital System Design".*

[9] J.W. Cooley and J.W. Tukey, (1965), An Algorithm for Machine Computation of Complex Fourier Series, Mathematical Computation, vol. 19, pp. 297-301.

[10] C.S. Burrus and T.W. Parks, (1985), DFT/FFT and Convolution Algorithms: Theory and Implementation, John Wiley and Sons, New York, NY, USA.

[11] R.D. Preuss, Very Fast Computation of the Radix-2, Discrete Fourier Transform, IEEE Transactions on Acoustics Speech and Signal Processing.

[12] John G. Proakis and D. G. Manolakis, "Digital Signal Processing: Principles, Algorithms and Applications", New York:

Macmillan Publishing Company, Second Edition, 1992, chapter - 6, page 456-464.

[13] J. G. Proakis and D.G. Manolakis, "Digital Signal Processing, Principles, Algorithms and Applications." 3rd Edition, 1998, Prentice Hall India Publications.

[14] Ma, Y. 1999. An e_ective memory addressing scheme for _t processors. Signal Processing, IEEE Transactions on 47, 3 (mar), 907 {911.

[15] 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.

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

[17] M.Petrov, M. Glesner, "Optimal FFT Architecture Selection for OFDM Receivers on FPGA," In Proc. of 2005 IEEE Intern. Conf. on Field Programmable Technology, pp. 313–314, PI. 0-7803-9407-0, Singapore. there by greatly reducing the memory storage requirement.

[18] J. Valls, T. Sansaloni et al. "Efficient pipeline FFT processors for WLAN MIMO OFDM systems," Electronics Letters, 2005, vol. 41, pp. 1043 1044.

[19] L. Jia, Y. Gao, J. Isoaho and H. Tenhunen, "A new VLSI oriented FFT algorithm and implementation," in Proceedings of 11th IEEE International ASIC Conference, 1998, pp. 337 341.

[20] K. K. Parhi, "High level algorithm and architecture transformations for DSP synthesis," IEEE Journal of VLSI Signal Processing, 1995, vol. 9, pp. 121 143.

Appendices

Computation Of 8-Point Fast Fourier Transform Using Verilog Code.

```
module FFT_8point
(
 input signed [15:0] xi_re, xi_im, xj_re, xj_im,
 input clk, rst_n,
 output signed [15:0] yi_re, yi_im, yj_re, yj_im);
 reg signed [15:0] R_re[0:5], R_im[0:5];
 reg [1:0] counter;
 wire signed [15:0] W_re[0:3], W_im[0:3];
 // Twiddle factors of three butterflies
 reg signed [15:0] W0_re, W0_im, W1_re, W1_im, W2_re, W2_im;
 wire sel1, sel2;
 wire signed [15:0] xj_re1, xj_im1, xj_re2, xj_im2;
 wire signed [15:0] yi_re1, yi_im1, yj_re1, yj_im1,
 yi_re2, yi_im2, yj_re2, yj_im2;
 wire signed [15:0] yi_out_re1, yi_out_im1, yi_out_re2,
 yi_out_im2;
 // The control signals
 assign sel1 =counter[1];
 assign sel2 =counter[0];
 // Calling butterfly tasks
ButterFly B0(xi_re, xi_im, xj_re, xj_im, W0_re, W0_im,
 yi_re1, yi_im1, yj_re1, yj_im1);
ButterFly B1(R_re[3], R_im[3], xj_re1, xj_im1, W1_re,
 W1_im, yi_re2, yi_im2, yj_re2, yj_im2);
ButterFly B2(R_re[5], R_im[5], xj_re2, xj_im2, W2_re,
 W2_im, yi_re, yi_im, yj_re, yj_im);
Mux2To2 MUX1(yi_re1, yi_im1, R_re[1], R_im[1], sel1,
```

```
28    yi_out_re1, yi_out_im1, xj_re1, xj_im1);
29    Mux2To2 MUX2(yi_re2, yi_im2, R_re[4], R_im[4], sel2,
30    yi_out_re2, yi_out_im2, xj_re2, xj_im2);
31    always @(*)
32    begin
33    // Reading values of twiddle factors for three butterflies
34    W0_re = W_re[counter];
35    W0_im = W_im[counter];
36    //W1_re=W_re[{counter[0],1'b0}];
37    //W1_im=W_im[{counter[0],1'b0}];
38    W2_re=W_re[0];
39    W2_im=W_im[0];
40    end
41    always @(posedge clk or negedge rst_n)
42    begin
43    if(!rst_n)
44    begin
45    R_re[0] <= 0; R_im[0] <= 0;
46    R_re[1] <= 0; R_im[1] <= 0;
47    R_re[2] <= 0; R_im[2] <= 0;
48    R_re[3] <= 0; R_im[3] <= 0;
49    R_re[4] <= 0; R_im[4] <= 0;
50    R_re[5] <= 0; R_im[5] <= 0;
51    end
52    else
53    begin
54    R_re[0] <= yj_re1; R_im[0] <= yj_im1;
```

Appendices

```
54    R_re[0] <= yj_re1; R_im[0] <= yj_im1;
55    R_re[1] <= R_re[0]; R_im[1] <= R_im[0];
56    R_re[2] <= yi_out_re1; R_im[2] <= yi_out_im1;
57    R_re[3] <= R_re[2]; R_im[3] <= R_im[2];
58    R_re[4] <= yj_re2; R_im[4] <= yj_im2;
59    R_re[5] <= yi_out_re2; R_im[5] <= yi_out_im2;
60    end
61    end
62    always @(posedge clk or negedge rst_n)
63    if(!rst_n)
64    counter <= 0;
65    else
66    counter <= counter+1;
67    // 1, 0
68    assign W_re[0] = 16'h4000;
69    assign  W_im[0] = 16'h0000;
70
71     assign  W_re[1] = 16'h2D41;
72    assign W_im[1] = 16'hD2BF;
73    // 0, -1
74     assign W_re[2] = 16'h0000;
75     assign W_im[2] = 16'hC000;
76    // -0.707, -0.707
77    assign  W_re[3] = 16'hD2BF;  assign W_im[3]=16'hD2BF;
78
79    endmodule
80
```

Appendices

Computation Of The FFT Butterfly unit.

```
1  module ButterFly
2  (
3  input signed [15:0] xi_re, xi_im, xj_re, xj_im, // Input data
4  input signed [15:0] W_re, W_im, // Twiddle factors
5  output reg signed [15:0] yi_re, yi_im, yj_re, yj_im);
6  // Extra bit to cater for overflow
7  reg signed [16:0] tempi_re, tempi_im;
8  reg signed [16:0] tempj_re, tempj_im;
9  reg signed [31:0] mpy_re, mpy_im;
10 always @(*)
11 begin
12 // Q2.14
13 tempi_re = xi_re + xj_re; tempi_im = xi_im + xj_im;
14 // Q2.14
15 tempj_re = xi_re - xj_re; tempj_im = xi_im - xj_im;
16 mpy_re = tempj_re*W_re - tempj_im*W_im;
17 mpy_im = tempj_re*W_im + tempj_im*W_re;
18 // Bring the output format to Q3.13 for first stage
19 // and to Q4.12 and Q5.11 for the second and third stages
20 yi_re = tempi_re>>>1; yi_im = tempi_im>>>1;
21 // The output for Q2.14 x Q 2.14 is Q4.12
22 yj_re = mpy_re[30:15]; yj_im = mpy_im[30:15];
23 end
24 endmodule
```

Appendices

```
1  module Mux2To2
2  (
3   input [15:0] xi_re, xi_im, xj_re, xj_im,
4   input sel1,
5   output reg [15:0] yi_out_re, yi_out_im, yj_out_re, yj_out_im);
6   always @ (*)
7  begin
8   if (sel1)
9  begin yi_out_re=xj_re; yi_out_im=xj_im;
10  yj_out_re=xi_re; yj_out_im=xi_im; end
11  else
12 begin yi_out_re=xi_re; yi_out_im=xi_im;
13  yj_out_re=xj_re; yj_out_im=xj_im; end
14  end
15  endmodule
16
```

الخلاصة

يقترح هذا المشروع هيكل جديد لتحويل فورير السريع الأساسية التي يحسب ٨ نقطة الأصل-٢ باستخدام عملية نقطة ثابتة في ثمانية فقط. تحويل وقد تم تصميم هذا فورير السريع المشروع من قبل برنامج كوارتز التي تنتجها شركة الترا وهذا البرنامج قد وفر لنا كل الإمكانيات لتصميم هذا المشروع ليوفر واجهات تصميم وتقديم رموز لتصميم أي دائرة وهذا البرنامج جعل تصميم أي مشروع سهلة وغير مكلفة.

وزارة التعليم العالي والبحث العلمي

جامعة ديالى

كلية الهندسة

قسم هندسة الحاسوب والبرمجيات

تصميم وتحليل ثابت ٨ نقطة تحويل فورير السريع بلغة فيريلوج

مشروع مقدم الى قسم هندسة الحاسبات والبرمجيات

في جامعة ديالى – كلية الهندسة كجزء من متطلبات نيل درجة البكالوريوس في هندسة الحاسبات والبرمجيات

اعداد :-

سجى عبد الحافظ – اسراء غازي

بإشراف

م.م. احمد خضير جميل

رجب ١٩٣٧ حزيران ٢٠١٦