IMPLEMENTATION OF DIT-FFT ALGORITHM USING VERILOG

A Project report submitted in partial fulfillment of the requirements for the award of the degree of

BACHELOR OF TECHNOLOGY

IN

ELECTRONICS AND COMMUNICATION ENGINEERING

Submitted by

V. Purna Satya Srinivas	**(318126512178)**	**D. Durga Sai Prasanth**	**(318126512134)**
K. Anil Kumar	**(318126512145)**	**G. Chaitanya Sai**	**(316126512135)**

Under the guidance of

Dr. B. Somasekhar

Associate Professor

PRAGNANAM BRAHMA

ANITS

DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING

ANIL NEERUKONDA INSTITUTE OF TECHNOLOGY AND SCIENCES

(UGC AUTONOMOUS)

(*Permanently Affiliated to AU, Approved by AICTE and Accredited by NBA & NAAC with 'A' Grade*)

Sangivalasa, Bheemili Mandal, Visakhapatnam dist. (A.P)

(2021-22)

ACKNOWLEDGEMENT

We would like to express our deep gratitude to our project guide **Dr. B. Somasekhar,** Associate Professor, Department of Electronics and Communication Engineering, ANITS, for his guidance with unsurpassed knowledge and immense encouragement.

We are grateful to **Dr. V. Rajya Lakshmi**, Head of the Department, Electronics and Communication Engineering, for providing us with the required facilities for the completion of the project work.

We would also express our deep gratitude to the **Dr. S. Srinivas**, Associate Professor, Department of Electronics and Communication Engineering, ANITS, for his guidance with unsurpassed knowledge and immense encouragement.

We are very much thankful to the **Principal and Management, ANITS, Sangivalasa,** for their encouragement and cooperation to carry out this work.

We express our thanks to all **teaching faculty** of Department of ECE, whose suggestions during reviews helped us in accomplishment of our project. We would like to thank **all non-teaching staff** of the Department of ECE, ANITS for providing great assistance in accomplishment of our project.

We would like to thank our parents, friends, and classmates for their encouragement throughout our project period. Finally, we thank everyone for supporting us directly or indirectly in completing this project successfully.

PROJECT STUDENTS

V. Purna Satya Srinivas	**(318126512178)**
D. Durga Sai Prasanth	**(318126512134)**
K. Anil Kumar	**(318126512145)**
G. Chaitanya Sai	**(316126512135)**

DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING

ANIL NEERUKONDA INSTITUTE OF TECHNOLOGY AND SCIENCES

(UGC AUTONOMOUS)

(Permanently Affiliated to AU, Approved by AICTE and Accredited by NBA & NAAC with 'A' Grade)

Sangivalasa, Bheemili Mandal, Visakhapatnam dist. (A.P)

PRAGNANAM BRAHMA

ANITS

CERTIFICATE

This is to certify that the project report entitled "**IMPLEMENTATION OF DIT-FFT ALGORITHM USING VERILOG**" submitted by **V. Purna Satya Srinivas (318126512178), D. Durga Sai Prasanth (318126512134), K. Anil Kumar (318126512145), G. Chaitanya Sai (316126512135)** in partial fulfillment of the requirements for the award of the Degree of **Bachelor of Technology** in **Electronics & Communication Engineering** of Andhra University, Visakhapatnam is a record of bonafide work carried out under my guidance and supervision.

	[illegible]
Project Guide	**Head of the Department**
[illegible]	
Dr. B. Somasekhar	**Dr. V. Rajya Lakshmi**
Associate Professor	**Professor & H.O.D**
Department of E.C.E	**Department of E.C.E**
ANITS	**ANITS**

Associate Professor
Department of E.C.E.
Anil Neerukonda
Institute of Technology & Sciences
Sangivalasa, Visakhapatnam-531 162

Head of the Department
Department of E C E
Anil Neerukonda Institute of Technology & Science
Sangivalasa-531 162

CONTENTS

ABSTRACT

DFT and FFT are two of the most extensively used DSP algorithms, whose relevance has grown dramatically in recent years. There are several uses of DFT, in convolution and linear filtering. The Fast Fourier Transform is another method for quickly computing DFT. There are vast variety of applications related to DFT and FFT. They are used in Communication devices to process the data received by the receivers and also to process the data to be sent by the transmitters. A DFT has a time complexity of $\boldsymbol{O(N^2)}$, while an FFT has an time complexity $\boldsymbol{O(N\log N)}$. DFT is an integral part of FFT. So in this paper mainly it is mainly focused to implement a Decimation-in-Time FFT algorithm in Verilog HDL on Vivado. Our Verilog model outputs further validated with Matlab FFT function outputs for the same input sequence, where the overall error percentage in magnitude of outputs is less than 1%.

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

FFT	Fast Fourier Transform
DIT	Decimation in Time
DFT	Discrete Fourier Transform
OFDM	Orthogonal Frequency Division Multiplexing
MIMO	Multi Input Multi Output
SSI	Small Scale Integration
MSI	Medium Scale Integration
LSI	Large Scale Integration
VLSI	Very Large Scale Integration
HDL	Hardware Description Language
VHDL	Very High Speed Integrated Circuit (VHSIC) Hardware Description Language
CAD	Computer Aided Design
IC	Integrated Circuit
RTL	Register Transfer Level
FPGA	Field Programmable Gate Arrays

CHAPTER 1

INTRODUCTION

1.1 Fourier series:

In engineering and science, a significant number of the phenomena researched are cyclic in character. In an alternating current circuit, for example, current and voltage are important. The constituent parts of these periodic functions might be analyzed (fundamentals and harmonics). Fourier analysis can be used to do this.

A Fourier series is an infinite sum of sines and cosines for a periodic function f(x). Sine and cosine functions are orthogonal to each other, and this is the basis for Fourier series. It is possible to break down any periodic function into a set of simple terms that can be plugged in, solved one at a time, and then recombined to obtain the solution to the original problem or an approximation of it to whatever accuracy is desired or practical using Fourier series computation and study, a technique known as harmonic analysis.

It is possible to plot an f(x) graph that has a period equal to L and show that f(x + L) is equal to f(x) for all values of x. We can plot a graph of the function over a larger range of x if we know how it appears during a single period (that may contain many periods).

This characteristic of repetition gives a first estimation to the periodic pattern f(x): f(x)' c1 sin(kx +a1) equals a1 cos(kx) + b1 sin(kx), where symbols with subscript “ 1 ” are constants that specify the amplitude and phase of this first approximation.

By adding an appropriate mix of harmonics to the fundamental (sine-wave) pattern, a far better approximation of the periodic pattern that is function f(x) may be built up. For example, c2 sin(2kx + c2) = a2 cos(2kx) + b2 sin(2kx) (which is the 2nd harmonic) and c3 sin(3kx + a3) = a3 cos(3kx) + b3 sin(3kx) (which is the 3rd harmonic) (that is the 3rd harmonic). Symbols with subscripts are the constants that govern the amplitude and phase of each harmonic contribution in general.

Fourier Theorem:

The Fourier Theorem states that arbitrary periodic functions have Fourier series representations that are extremely difficult to comprehend. Using the Laurent expansions, we will demonstrate that periodic analytic functions have such a form in this section.

Fourier analysis for Periodic Functions:

Laurent expansions are used to construct the Fourier series representation of analytic functions. Additional important conclusions in harmonic analysis are derived from elementary complex analysis, such as the representation of C periodic functions by Fourier series, the representation of rapidly decreasing functions by Fourier integrals, and Shannon's sampling theorem as shown in Fig 1.1.

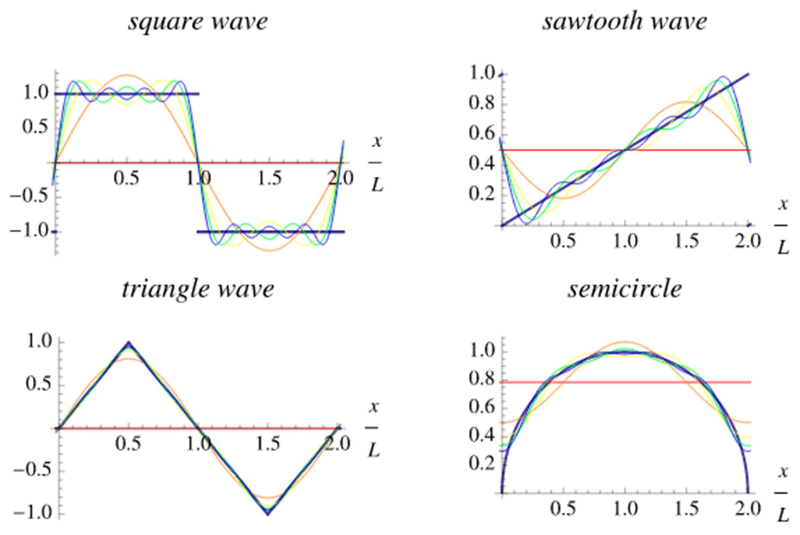


Fig 1.1: Square, Saw tooth, Triangle, Semicircle waveforms

A function has a period L if f(x+L) Equals f(x) for every x in its domain. The basic period is the smallest positive value of L.

The trigonometric functions sin x and cos x are periodic functions with fundamental period 2 and tan x is periodic with fundamental period. A constant function is a periodic function of length L.

It is simple to demonstrate that if the functions f_1,f_2,f_3..., f_n are periodic with period L, then every linear combination is valid.

Even and Odd Function:

A function f(x) is said to be even if f(−x) = f(x).

The function f(x) is said to be odd if f(−x) = −f(x).

Graphically, even functions have symmetry about the y-axis, whereas odd functions have symmetry around the origin.

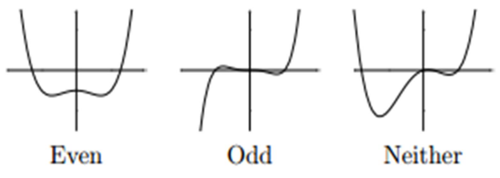


Fig 1.2: Even, Odd, Neither Even nor Odd functions

Examples:

Sums of odd powers of x are odd: $5x^3 - 3x$

Sums of even powers of x are even: $-x^6 + 4x^4 + x^2 - 3$

Since x is odd, and cos x is even.

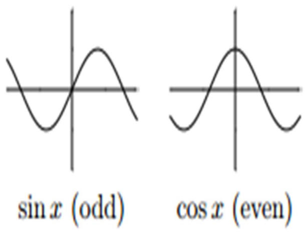


Fig 1.3: Sin and Cosine functions

The product of two odd functions is even: x sin x is even

The product of two even functions is even: x^2 cos x is even

The product of an even function and an odd function is odd: sin x cos x is odd

What does the Fourier Series Formula entail?

The Fourier series formula for a function is given as

$$f(x) = \frac{1}{2}a_0 + \sum_{n=1}^{\infty} a_n \cos nx + \sum_{n=1}^{\infty} b_n \sin nx \quad (1.1)$$

Where,

$$a_0 = \frac{1}{\pi}\int_{-\pi}^{\pi} f_x dx$$

$$a_n = \frac{1}{\pi}\int_{-\pi}^{\pi} f_x \cos nx \, dx$$

$$b_n = \frac{1}{\pi}\int_{-\pi}^{\pi} f_x \sin nx \, dx$$

$$n = 1,2,3,4, \ldots ....$$

Fourier series Applications and Uses:

A Fourier series is a sort of infinite mathematical series in which trigonometric functions are used. Fourier series are used in practical mathematics, particularly in physics and electronics, to represent periodic functions such as those seen in communications signal waveforms.

It is similar to a Taylor series, which depicts functions as potentially infinite sums of monomial terms.

- The Fourier Series is very useful in mathematical analysis. It is readily separated and integrated since it is a sum of numerous sines and cosines, which typically facilitates analysis of functions such as saw waves, which are popular signals in experiments.
- The Fourier Series enables experimenters to discover noise sources. It may also be used to remove noise sources by employing the concept of the Inverse Fast Fourier Transform (IFFT).
- The Fourier Series also provides a streamlined analytical method for dealing with discontinuous functions.

Fourier analysis is important in a wide range of fields, from scientific equipment to rigorous mathematical analysis. Digital electronics and numerical Algorithms like the FFT have made the

Fourier series one of the most extensively used and helpful mathematical tools available to scientists.

1.2 Fourier Transform:

When discussing time series analysis, the Fourier transform is a must. L tends to infinity complex Fourier series is generalized as the Fourier transform. Allowing n/L tends to k, replace discrete An with continuous F*(k)*dk forming equation 1.2, 1.3.

$$a_n = \frac{1}{L}\int_{-L}^{L} f[t]\cos[\frac{n\pi t}{L}]\,dt \quad (1.2)$$

$$b_n = \frac{1}{L}\int_{-L}^{L} f[t]\sin[\frac{n\pi t}{L}]\,dt \quad (1.3)$$

Time series (e.g. audio samples) may be broken down into their component frequencies using the Fourier Transform as in equation 1.4.

Fourier transforms may be reversed to map frequencies back to their time sequences by performing an Inverse Fourier Transform as in equation 1.5.

Fourier Transform

$$F(x) = \int_{-\infty}^{\infty} f(x)\, e^{-2\pi ikx} dk \quad (1.4)$$

Inverse Fourier Transform

$$f(x) = \int_{-\infty}^{\infty} F(k)\, e^{2\pi ikx} dk \quad (1.5)$$

1.3 Discrete Fourier Transform (DFT):

The continuous Fourier Transform is not very effective for determining the frequency spectrum of a sampled function. The Discrete Fourier Transform is required.

DFT:

$$F_n = \sum_{k=0}^{N-1} f_k e^{-\frac{2\pi i n}{N}} \quad (1.6)$$

The complex numbers $f_0 \ldots f_N$ are transformed into complex numbers $F_0 \ldots F_N$ in equation 1.6.

IDFT:

$$f_k = \frac{1}{N}\sum_{k=0}^{N-1} F_n e^{-\frac{2\pi i n k}{N}} \quad (1.7)$$

The complex numbers $F_0 \ldots F_n$ are transformed into complex numbers $f_0 \ldots f_N$ in equation 1.7.

Because real-world data contains complex integers, interpreting a DFT can be challenging. The power at that frequency is represented by the magnitude of the complex number for a DFT component. The relative quantities of the real and imaginary coefficients can be used to calculate the waveform's phase. Positive and "negative" frequencies are also present.

Fourier series and Fourier integrals are related via sampling:

Treating signals f(t) as arbitrary L2[,] functions for the sake of Fourier analysis. However, in practice, signal processing can only handle a certain quantity of data. The signal is typically sampled digitally at regular or irregular discrete time intervals. The processed sample is then utilized to recreate the signal on its own. If the sample is not tampered with, the signal should be recovered precisely. What makes this possible? How does one precisely recreate a function f(t) from discrete samples?

For arbitrary functions f, this is not feasible (t). However, the challenge is not hopeless since the signals used in signal processing, such as speech or pictures are not arbitrary. The human voice, for example, stands out clearly from static or random noise. The frequencies of sound in the human voice are typically in a small frequency band, which is one distinguishing feature. In reality, any signal that we can capture and analyze using actual hardware must be limited to a specific frequency spectrum. Shannon-Whittaker sampling, this is one method for collecting and then precisely reproducing a certain type of frequency-restricted signals. This approach is extremely practical since it is used frequently in telephone, radio and television broadcasts, radar, and other applications.

Applications of DFT:

Information is frequently encoded in the sinusoids that make up a signal. This is true for both naturally occurring signals and those generated by people. Many objects in our universe oscillate. For example, human vocal cord vibration causes speech; stars and planets change brightness as they spin on their axes and orbit around each other; ship propellers cause periodic displacement of water, and so forth. In these signals, the frequency, phase, and amplitude of the component sinusoids are more essential than the shape of the time domain waveform. This data is extracted using the DFT.

- Spectrum Analysis of a Sinusoid: Windowing, Zero-Padding, and FFT.
- Spectrograms.
- Filters and Convolution.
- Correlation Analysis.
- Power Spectral Density Estimation.

1.4 Fast Fourier Transform (FFT):

A fast Fourier transform (FFT) is a method that calculates the discrete Fourier transform (DFT) of a sequence — the discrete Fourier transform is a tool for converting specified sorts of function sequences into other types of representations.

A rapid Fourier transform can be employed in a variety of signal processing applications. It might be beneficial for interpreting sound waves or for image-processing technology. A rapid Fourier transform can be used to solve a variety of equations or to display various sorts of frequency activity in helpful ways.

The fast Fourier transform (FFT) is commonly utilized in signal processing. Reduce the amount of calculations needed to convert N points N^2 to N log N, where LG is a base two method. There are two types of FFT: time decimation and frequency decimation.

The FFT Algorithm differs in that it reorders the input components in reverse bit order before creating the output transform (time decimation). The fundamental task is to divide a transform of length N into two transforms of length N / 2.

Fast Fourier transform and the DFT are generally the domain of engineers and mathematicians attempting to improve or enhance features of various technologies since they are exceedingly mathematical aspects of both computers and electrical engineering. Fast Fourier transforms, for example, might be useful in sound engineering, seismology, or voltage measurements.

Discrete Fourier Transform for n samples, the transform would ordinarily take O(n^2) time to process:

$$F_n = \sum_{k=0}^{N-1} f_k e^{-\frac{2\pi i}{N}} \quad (1.8)$$

Fast Fourier Transform takes O(n log(n)) time as in equation 1.8.

Most common Algorithm is the Cooley-Tukey Algorithm.

Even vs Odd Function:

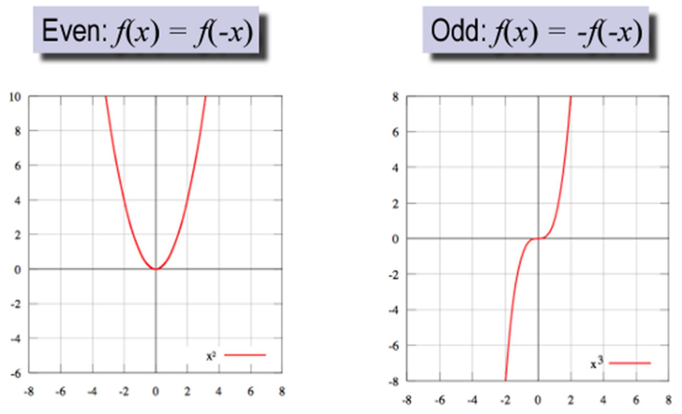


Fig 1.4: Even vs Odd Function

Applications of FFT:

The FFT has several uses, including audio processing, radar, sonar, and software defined radio, to mention a few. In all of these applications, the FFT converts a time domain signal into a frequency domain representation of the signal. It can also do an inverse FFT and re-create a signal by varying the amplitude of certain of the frequency components. By removing the high frequencies, you have created a low pass filter. It's not exactly as straightforward as that, but you get the idea.

- Digital Recording
- Sampling
- Additive Synthesis
- Pitch Correction Software.

1.5 DFT vs FFT:

Table 1.1: DFT vs FFT

DFT	FFT
The DFT Algorithms can be either programmed on general purpose digital computers or implemented directly by special hardware	The FFT Algorithm is used to compute the DFT of a sequence or its inverse.
A DFT can be performed as $O(N^2)$ in time complexity.	FFT reduces the time complexity in the order of O (NlogN).
Establish the relationship between the time domain and the frequency domain	Faster calculation
DFT can be used in many digital processing systems across a variety of applications such as calculating a signal's frequency spectrum, solving partial differential applications.	Applications of FFT include spectral analysis in analog video measurements, large integer and polynomial multiplication, filtering Algorithms, computing isotopic distributions, calculating Fourier series coefficients
Discreet version	Fast version

In a nutshell, the Discrete Fourier Transform is important in physics because it may be used as a mathematical tool to define the link between discrete signals' time domain and frequency domain representations. It is a straightforward yet time-consuming Algorithm. However, a more sophisticated but less time-consuming approach, such as the Fast Fourier Transform, can be utilized to minimize the processing time and complexity of big transformations. FFT is a DFT implementation that is used for quick DFT computation. In summary, FFT can perform everything a DFT can do, but considerably more effectively and quickly. It's a quick approach to compute the DFT.

1.6 FFT Algorithms:

Because the computational phase is too lengthy, it can be done using FFT or fast Fourier transform. FFT is just the computation of the discrete Fourier transform in an Algorithmic structure, where the computational portion is decreased.

The fundamental advantage of having FFT is that it allows you to create FIR filters.

The formula for FFT is as equation 1.9.

$$x[K] = \sum_{n=0}^{N-1} x[n] W_N^{nk} \quad (1.9)$$

Radix-2 FFT Algorithms:

Consider the computation of the N = 2v point DFT by the divide-and conquer approach.The N-point data sequence into two N/2-point data sequences $f_1(n)$ and $f_2(n)$, corresponding to the even-numbered and odd-numbered samples of x(n), respectively, that is

$$f_1(n) = x(2n) \quad (1.10)$$

$$f_2(n) = x(2n+1), \qquad n = 0,1,2,\ldots\ldots,\frac{N}{2}-1 \quad (1.11)$$

Thus $f_1(n)$ and $f_2(n)$ are obtained by decimating x(n) by a factor of 2 as in equation 1.10, 1.11 and hence the resulting FFT Algorithm is called a decimation-in-time Algorithm.

Now the N-point DFT can be expressed in terms of the DFT's of the decimated sequences as follows:

$$X(k) = \sum_{n=0}^{N-1} x[n] W_N^{nk}, \qquad k = 0,1, \dots\dots, N-1$$

$$= \sum_{n=0,even}^{N-1} x[n] W_N^{nk} + \sum_{n=0,odd}^{N-1} x[n] W_N^{nk}$$

$$= \sum_{m=0}^{\frac{N}{2}-1} x[2m] W_N^{2m} + \sum_{m=0}^{\frac{N}{2}-1} x[2m+1] W_N^{(2m+1)k} \qquad (1.12)$$

But WN2 = WN/2. With this substitution, the equation 1.12 can be expressed as

$$X(k) = \sum_{m=0}^{N/2-1} f_1[m] W_{N/2}^{km} + \sum_{m=0}^{N/2-1} f_2[m] W_{N/2}^{km}$$

$$= F_1(k) + W_N^k F_2(k), \qquad k = 0,1, .. N-1 \quad (1.13)$$

Where F1*(k)* and F2*(k)* are the N/2-point DFTs of the sequences f1(m) and f2(m), respectively.

Since F1*(k)* and F2*(k)* are periodic, with period N/2, we have $F1(k + N/2) = F1(k)$ and $F2(k + N/2) = F2(k)$. In addition, the factor $W_N^{k+N/2} = -W_N^k$. Hence the equation 1.13 may be expressed as

$$X(k) = F_1(k) + W_N^k F_2(k), \qquad k = 0,1, .. \frac{N}{2} - 1 \quad (1.14)$$

$$X\left(k + \frac{N}{2}\right) = F_1(k) - W_N^k F_2(k), \qquad k = 0,1, .. \frac{N}{2} - 1 \quad (1.15)$$

The direct computation of F_1*(k)* requires $(N/2)^2$ complex multiplications. The same applies to the computation of F_2*(k)*. Furthermore, there are N/2 additional complex multiplications required to compute $W_N^k$$F_2$*(k)*. Hence the computation of X*(k)* requires $2(N/2)^2 + N/2 = N^2/2 + N/2$ complex multiplications. This first step results in a reduction of the number of multiplications from N^2 to $N^2/2 + N/2$, which is about a factor of 2 for N large.

By computing N/4-point DFTs obtain the N/2-point DFTs F1*(k)* and F2*(k)* from the equations 1.14, 1.15.

$$F_1(k) = F[f_1(2n)] + W_{N/2}^{k} F[f_1(2n+1)], k = 0,1, \ldots., \frac{N}{4} - 1; \quad n = 0,1, \ldots. \frac{N}{4} - 1 \quad (1.16)$$

$$F_1(k + N/4) = F[f_1(2n)] - W_{\frac{N}{2}}^{k} F[f_1(2n+1)], k = 0,1, .., \frac{N}{4} - 1; \quad n = 0,1, .. \frac{N}{4} - 1 \quad (1.17)$$

$$F_2(k) = F[f_2(2n)] + W_{N/2}^{k} F[f_1(2n+1)], \quad k = 0,1, \ldots., \frac{N}{4} - 1; \quad n = 0,1, \ldots. \frac{N}{4} - 1 \quad (1.18)$$

$$F_1(k + N/4) = F[f_2(2n)] - W_{\frac{N}{2}}^{k} F[f_2(2n+1)], k = 0,1, .., \frac{N}{4} - 1; \quad n = 0,1, .. \frac{N}{4} - 1 \quad (1.19)$$

F(*)→ Fourier Transform

The decimation of the data sequence can be repeated again and again until the resulting sequences are reduced to one-point sequences as in equations 1.16, 1.17, 1.18, 1.19. For N = 2v, this decimation can be performed v = $\log_2 N$ times. Thus the total number of complex multiplications is reduced to (N/2)$\log_2 N$. The number of complex additions is $N \log_2 N$.

Radix-4 FFT Algorithm:

The radix-4 decimation-in-time and decimation-in-frequency methods are quick. FFTs increase speed by reusing the results of smaller, intermediate calculations to generate multiple DFT frequency outputs. The radix-4 decimation-in-time technique divides the discrete Fourier transform (DFT) equation into four parts: sums over all groups of every fourth discrete-time index n=[0,4,8,...,N-4], n=[1,5,9,...,N-3], n=[2,6,10,...,N-2], and n=[3,7,11,...,N-1]. (This only works if the FFT length is a multiple of four.) Further mathematical manipulation reveals that the length-N DFT, like the radix-2 decimation-in-time FFT, can be computed as the sum of the outputs of four length-N/4 DFTs, of the even-indexed and odd-indexed discrete-time samples, respectively, where three of them are multiplied by so-called twiddle factors. The radix-4 decimation-in-time and decimation-in-frequency methods are quick. FFTs increase speed by reusing the results of smaller, intermediate calculations to generate multiple DFT frequency outputs. The radix-4 decimation-in-time technique divides the discrete Fourier transform (DFT) equation into four parts: sums over all groups of every fourth discrete-time index n=[0,4,8,...,N-

4], n=[1,5,9,...,N-3], n=[2,6,10,...,N-2], and n=[3,7,11,...,N-1]. (This only works if the FFT length is a multiple of four.) Further mathematical manipulation reveals that the length-N DFT, like the radix-2 decimation-in-time FFT, can be computed as the sum of the outputs of four length-N/4 DFTs, of the even-indexed and odd-indexed discrete-time samples, respectively, where three of them are multiplied by so-called twiddle factors. The radix-4 decimation-in-time and decimation-in-frequency methods are quick. FFTs increase speed by reusing the results of smaller, intermediate calculations to generate multiple DFT frequency outputs. The radix-4 decimation-in-time technique divides the discrete Fourier transform (DFT) equation into four parts: sums over all groups of every fourth discrete-time index n=[0,4,8,...,N-4], n=[1,5,9,...,N-3], n=[2,6,10,...,N-2], and n=[3,7,11,...,N-1]. (This only works if the FFT length is a multiple of four.) Further mathematical manipulation reveals that the length-N DFT, like the radix-2 decimation-in-time FFT, can be computed as the sum of the outputs of four length-N/4 DFTs, of the even-indexed and odd-indexed discrete-time samples, respectively, where three of them are multiplied by so-called twiddle factors.

$$W_N^k = e^{-\left(i\frac{2\pi k}{N}\right)}, W_N^{2k}, and\ W_N^{3k}.$$

$$X(K) = \sum_{n=0}^{N-1} x(n)e^{-\left(i\frac{2\pi nk}{N}\right)}$$

$$= \sum_{n=0}^{\frac{N}{4}-1} x(4n)e^{-\left(i\frac{2\pi\times(4n)k}{N}\right)} + \sum_{n=0}^{\frac{N}{4}-1} x(4n+1)e^{-\left(i\frac{2\pi\times(4n+1)k}{N}\right)}$$

$$+ \sum_{n=0}^{\frac{N}{4}-1} x(4n+2)e^{-\left(i\frac{2\pi\times(4n+2)k}{N}\right)} + \sum_{n=0}^{\frac{N}{4}-1} x(4n+3)e^{-\left(i\frac{2\pi\times(4n+3)k}{N}\right)}$$

$$= DFT_{\frac{N}{4}}[x(4n)] + W_N^k DFT_{\frac{N}{4}}[x(4n+1)] + W_N^{2k} DFT_{\frac{N}{4}}[x(4n+2)]$$

$$+ W_N^{3k} DFT_{\frac{N}{4}}[x(4n+3)] \quad (1.20)$$

This is called decimation in time because the time samples are rearranged in alternating groups, and a radix-4 Algorithm because there are four groups. Figure graphically illustrates this form of the DFT computation

Radix-4 DIT structure:

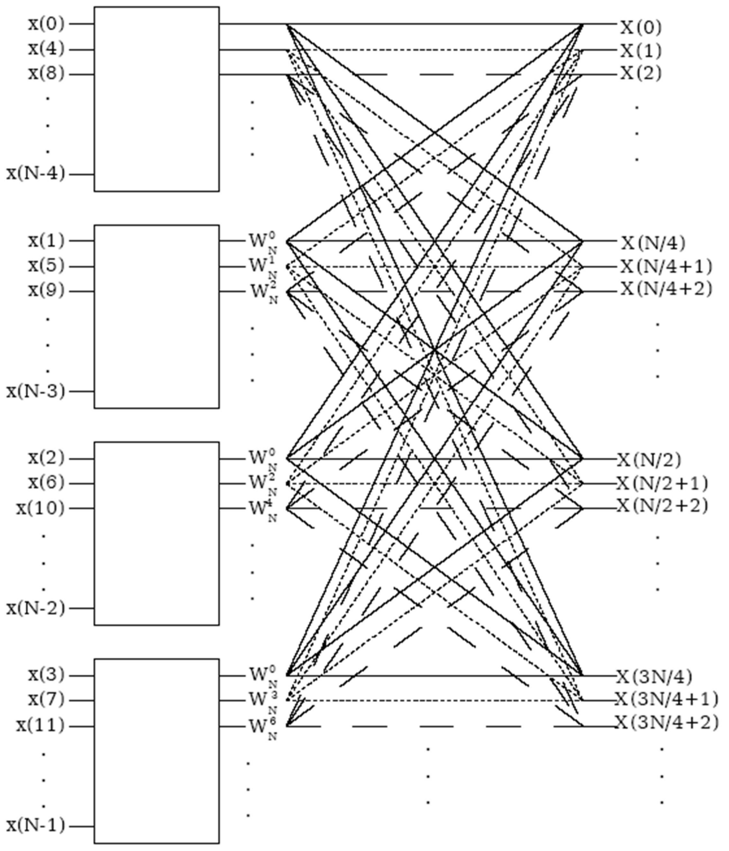


Fig 1.5: Radix-4 DIT Structure

Because of the short-length DFTs' periodicity with N/4, their outputs for frequency-sample k are utilised to compute X*(k)*, X(k+N/4), X(k+N/2), and X(k+(3/4)N). The radix-4 FFT's efficiency is due to this reuse. The radix-4 butterfly, seen in Figure, is formed by the computations associated with each set of four frequency samples. Further rearranging reveals that this computation may be reduced to three twiddle-factor multiplies and a length-4 DFT! According to the theory of multi-dimensional index maps, this must be the case, and FFTs of any factorable length can be composed of sequential stages of shorter-length FFTs with twiddle-factor multiplications in between. There are no multiplies and just eight complex additions in the length-4 DFT (this efficient computation can be derived using a radix-2 FFT).

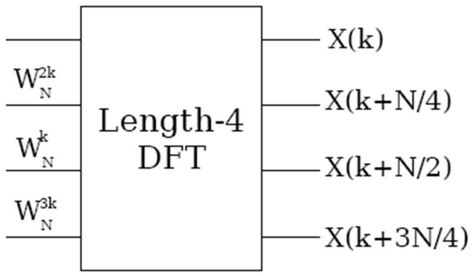


Fig 1.6: Length 4-DFT unit

If the FFT length N=4M, the shorter-length DFTs can be further decomposed recursively in the same manner to produce the full radix-4 decimation-in-time FFT. As in the radix-2 decimation-in-time FFT, each stage of decomposition creates additional savings in computation. To determine the total computational cost of the radix-4 FFT, note that there are M=log4N=log2N2 stages, each with N/4 butterflies per stage. Each radix-4 butterfly requires 3 complex multiplies and 8 complex additions. The total cost is then

Radix-4 FFT Operation Counts:

$3\frac{N}{4}\frac{\log_2 N}{2} = \frac{3}{8}N\log_2 N$ complex multiplies (75% of a radix-2 FFT)

$8\frac{N}{4}\ \frac{\log_2 N}{2} =\ \mathrm{N}\log_2 N$ complex adds (same as a radix-2 FFT)

The radix-4 FFT requires only 75% as many complex multiplies as the radix-2 FFTs, although it uses the same number of complex additions. These additional savings make it a widely-used FFT Algorithm.

The decimation-in-time operation regroups the input samples at each successive stage of decomposition, resulting in a "digit-reversed" input order. That is, if the time-sample index n is written as a base-4 number, the order is that base-4 number reversed. The digit-reversal process is illustrated for a length-N=64 example below.

Table 1.2: Bit Reversal for N=64

Original Number	Original Digit Order	Reversed Digit Order	Digit-Reversed Number
0	000	000	0
1	001	100	16
2	002	200	32
3	003	300	48
4	010	010	4
5	011	110	20
⋮	⋮	⋮	⋮

It is important to note that if the input signal data are reversed before beginning the FFT computations, the outputs of each butterfly throughout the computation can be placed in the same memory locations from which the inputs were fetched, resulting in an in-place Algorithm that requires no extra memory to perform the FFT. Most FFT implementations run in the background, overwriting the input data with intermediate values and then the result. Burrus's radix-4 FFT has a minor rearrangement that allows the inputs to be organised in bit-reversed rather than digit-reversed order.

A radix-4 decimation-in-frequency FFT can be created in the same way that a radix-2 DIF FFT can be created by computing all four groups of every fourth output frequency sample separately. The DIF radix-4 FFT is a flow-graph reversal of the DIT radix-4 FFT, with the same operation counts and twiddle factors but in the opposite order. For an in-place DIF method, the output is in digit-reversed order.

Split-radix FFT Algorithms:

The split-radix method, which Duhamel and Hollman initially described and called in 1984, requires fewer total multiply and add operations than any preceding power-of-two technique. (Yavne developed roughly the same technique in 1968, but the description was so unusual that the work was generally ignored.) Many FFT specialists felt it was ideal in terms of overall complexity for a long time, however Johnson and Frigo recently identified even more efficient versions.

The split-radix method may be obtained by carefully inspecting the radix-2 and radix-4 flowgraphs, as shown in Figure 1. While the radix-4 technique has less nontrivial twiddle factors in most places, the radix-2 structure lacks twiddle factors present in the radix-4 structure in certain places, or those twiddle factors simplify to multiplication by I which really needs just adds. An method of lower complexity than either may be devised by correctly combining radix-2 and radix-4 operations.

Radix-4

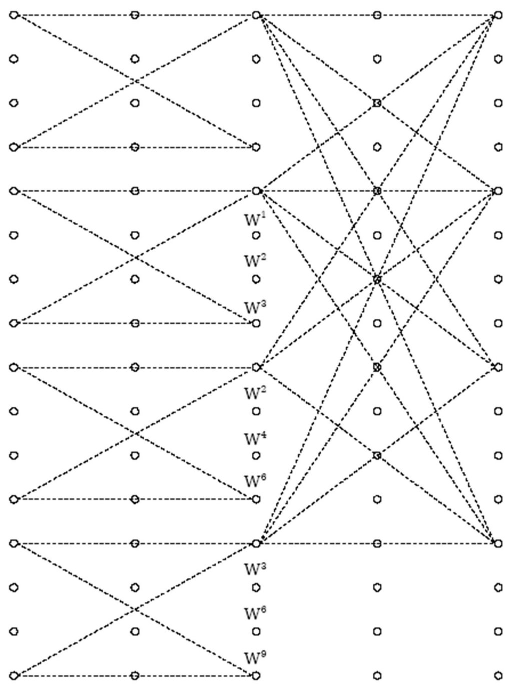


Fig 1.7: Radix-4 Butterfly diagram

By combining the radix-2 and radix-4 decompositions, the split-radix method may also be developed.

$$X(k) = \sum_{n=0}^{\frac{N}{2}-1} x(2n)e^{-\left(i\frac{2\pi\times(2n)k}{N}\right)} + \sum_{n=0}^{\frac{N}{4}-1} x(4n+1)e^{-\left(i\frac{2\pi\times(4n+1)k}{N}\right)} + \sum_{n=0}^{\frac{N}{4}-1} x(4n+3)e^{-\left(i\frac{2\pi\times(4n+3)k}{N}\right)}$$

$$= DFT_{\frac{N}{2}}[x(2n)] + W_N^k DFT_{\frac{N}{4}}[x(4n+1)] + W_N^{3k} DFT_{\frac{N}{4}}[x(4n+3)] \quad (1.21)$$

The whole split-radix method is obtained by further decomposing the half- and quarter-length DFTs. The combination of variable-length FFTs in different areas of the flowgraph produces an irregular Algorithm; Sorensen et al. describe how to alter the calculation such that the data keeps the simpler radix-2 bit-reverse order. The multiplicative complexity of the split-radix technique is around two-thirds that of the radix-2 FFT, and it outperforms the radix-4 FFT and any higher power-of-two radix. The adds within the complicated twiddle-factor multiplies are similarly decreased, but because the underlying butterfly tree remains the same in all power-of-two methods, the butterfly additions remain the same, and the total addition reduction is significantly smaller.

The split-radix Algorithm has an atypical structure. Successful programs for single-processor machines have been published (Sorensen), however it may be challenging to efficiently write the split-radix technique for vector or multi-processor devices.

Example

Considered eight points named from x0 to x7 choose the even terms in one group and the odd terms in the other. Points x0, x2, x4 and x6 have been grouped into one category and similarly, points x1, x3, x5 and x7 has been put into another category.

Diagrammatic view:

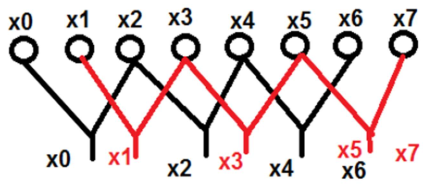


Fig 1.8: Choosing 8-points in Split radix algorithm

Here, points x0, x2, x4 and x6 have been grouped into one category and similarly, points x1, x3, x5 and x7 has been put into another category. Now, further make them in a group of two and can proceed with the computation.

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

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

$$= G[k] + H[k] \times W_N^k \quad (1.22)$$

Initially an eight-point sequence is taken but later broken that one into two parts G[k] and H[k]. G[k] stands for the even part whereas H[k] stands for the odd part.

To realize it through a diagram

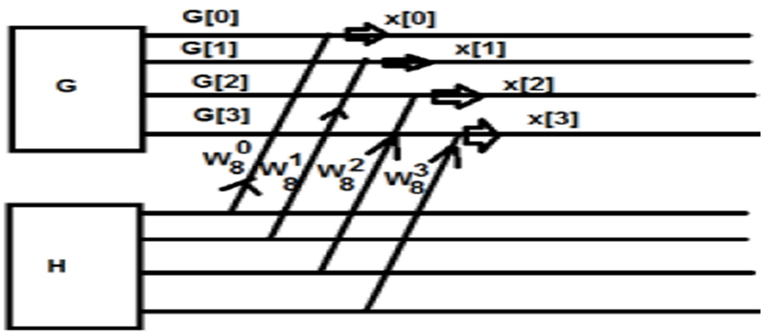


Fig 1.9: Diagram representing first 4 outputs in the output sequence

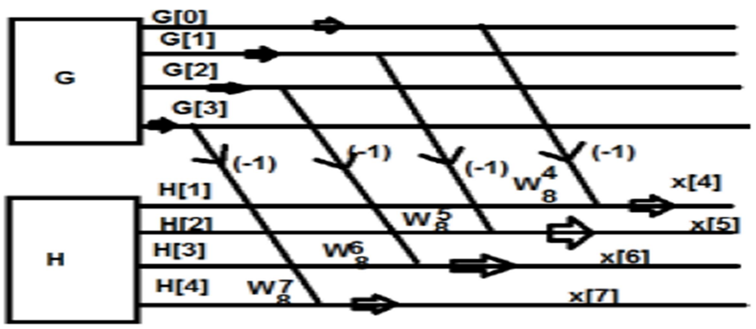


Fig 1.10: Diagram representing last 4 outputs in the output sequence

From the above figure, we can see as in equations 1.23, 1.24, 1.25, 1.26.

$$W_8^4 = -1 \quad (1.23)$$

$$W_8^5 = -W_8^1 \quad (1.24)$$

$$W_8^6 = -W_8^2 \quad (1.25)$$

$$W_8^7 = -W_8^3 \quad (1.26)$$

Similarly, the final values can be written as equations 1.27, 1.28, 1.29, 1.30.

$$G[0] - H[0] = X[5] \quad (1.27)$$

$$G[1] - W_8^1 H[1] = X[5] \quad (1.28)$$

$$G[2] - W_8^2 H[2] = X[6] \quad (1.29)$$

$$G[1] - W_8^3 H[3] = X[7] \quad (1.30)$$

The above one is a periodic series. The disadvantage of this system is that K cannot be broken beyond 4 point. Further Processing

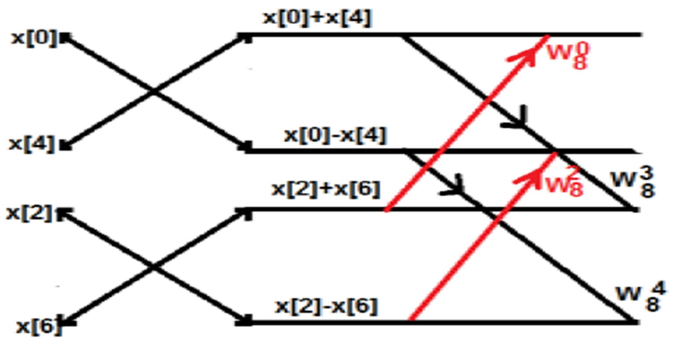


Fig 1.11: Diagram describing disadvantage of the split radix algorithm

CHAPTER 2

RELATED WORK

2.1 Literature Review:

Today's digital signal processing activities necessitate multiple multiplications, necessitating the use of an extremely fast multiplier to meet a wide range of hardware and speed requirements. [2]

The FFT/IFFT technique is one of the most widely utilized in digital signal processing. Recently, the FFT processor has seen widespread use in the field of digital signal processing, particularly in communication systems. FFT/IFFT processors are essential components of a wireless broadband communication system based on Orthogonal Frequency Division Multiplexing (OFDM). The primary restrictions for FFT processors utilized in wireless communication systems nowadays are execution time and power consumption. [3]

Digital signal processing is a method that is widely employed in video and audio applications. Many approaches for analyzing visual or audio signals are accessible in the DSP domain. The Discrete Fourier Transform (DFT) technique is frequently used in digital signal processing applications such as linear filtering, convolution, spectrum analysis, and correlation. A frequency domain representation of the original sequence is referred to as a DFT. [4]

DFT can be calculated in three different methods. The first way is to use the DFT formula or simultaneous equation. The correlation technique is the second, while the application of the Fast Fourier Transform is the third (FFT). The first way is useful for grasping the fundamental concepts of DFT, however it is unsuitable for practical and application applications. The second method entails detecting a known waveform in another signal. It is utilized in a few specific applications. When DFT contains fewer than 32 points, this approach is employed. The genius approach is the third method, in which the FFT Algorithm divides a DFT with N points into N DFTs, each with a single point. It outperforms DFT. All three ways get the same result. We concentrated on FFT in this section. The DFT employing FFT and its MATLAB implementation were discussed in this study. The FFT spectra of the outputs are examined. [5]

FFT processers are used in OFDM MIMO systems which contribute in processing of the data for transmission and reception. Orthogonal frequency division multiplexing (OFDM) (de)modulation introduces a significant amount of latency in the baseband of huge multiple-input multiple-output (MIMO) systems. A fast Fourier transform (FFT) processor and associated reordering strategy are suggested to fulfill the low-latency demand of massive MIMO systems, reducing the processing and reordering latency of OFDM-based systems, respectively. [1]

DFT is widely used in standard embedded system applications such as wireless communication protocols requiring Orthogonal Frequency Division Multiplexing. [6]

However, DFT is difficult to implement directly due to its computational complexity. In practice, Fast Fourier transform (FFT) is used for reducing the complexity of computations. For FFT processors, butterfly operation is the most computationally demanding stage. Traditionally, a butterfly unit is composed of complex adders and multipliers, and the multiplier is usually the speed bottleneck in the pipeline of the FFT processor. [7]

As there are several aspects where FFT play a key role in the processing the of the data during transmission and reception in a communication system. It enables computers to quickly calculate the various frequency components of time-varying signals, as well as to reconstruct such signals from a set of frequency components.

Thereby we decided to implement an FFT model through Verilog and design an efficient model and extract the model aspects.

CHAPTER 3

METHODOLOGY

Radix-2 decimation-in-time and decimation-in-frequency are quick. The Fourier transformations (FFTs) are the most basic FFT Algorithms. They gain speed, like other FFTs, by reusing the results of smaller, intermediate calculations to compute multiple DFT frequency outputs.

3.1 Decimation in time:

The radix-2 decimation-in-time technique divides the discrete Fourier transform (DFT) equation into two parts: a sum over the even-numbered indices n=[0,2,4,...,N-2] and a sum over the odd-numbered indices n=[1,3,5,...,N-1] as in

Equation:

$$X(K) = \sum_{n=0}^{N-1} x(n) e^{-\left(i\frac{2\pi nk}{N}\right)}$$

$$= \sum_{n=0}^{\frac{N}{2}-1} x(2n) e^{-\left(i\frac{2\pi \times (2n)k}{N}\right)} + \sum_{n=0}^{\frac{N}{2}-1} x(2n) e^{-\left(i\frac{2\pi \times (2n+1)k}{N}\right)}$$

$$= \sum_{n=0}^{\frac{N}{2}-1} x(2n) e^{-\left(i\frac{2\pi nk}{\frac{N}{2}}\right)} + e^{-\left(i\frac{2\pi k}{N}\right)} \sum_{n=0}^{\frac{N}{2}-1} x(2n+1) e^{-\left(i\frac{2\pi nk}{\frac{N}{2}}\right)}$$

$$= \boldsymbol{DFT}_{\frac{N}{2}} [\![x(0), x(2), .., x(N-2)]\!] + W_N^k \boldsymbol{DFT}_{\frac{N}{2}} \big[[x(0), x(2), .., x(N-1)]\big] \quad (3.1)$$

The mathematical simplifications in Equation 3.1, reveal that all DFT frequency outputs X*(k)* can be computed as the sum of the outputs of two length-N/2 DFTs, of the even-indexed and odd-indexed discrete-time samples, respectively, where the odd-indexed short DFT is multiplied by a so-called twiddle factor term $W_N^k = e^{-(i\frac{2\pi k}{N})}$. Because the time samples are reorganised in alternating groups, this is referred to as decimation in time and the technique is referred to as a radix-2 Algorithm because there are two groups. Figure graphically illustrates this form of the

DFT computation, where for convenience the frequency outputs of the length-N/2 DFT of the even-indexed time samples are denoted G(k) and those of the odd-indexed samples as H*(k)*. Because of the periodicity with N/2 frequency samples of these length-N/2 DFTs, G*(k)* and H*(k)* can be used to compute two of the length-N DFT frequencies, namely X(k) and X$(k+\frac{N}{2})$, but with a different twiddle factor. This reuse of these short-length DFT outputs gives the FFT its computational savings.

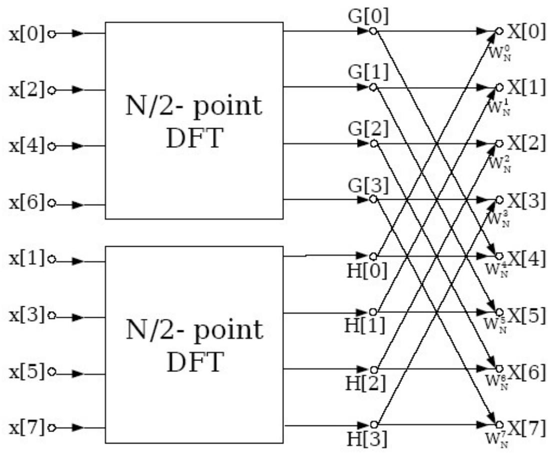


Fig 3.1: Decimation in time of a Length-N DFT into two length-N/2 DFTs followed by a combining stage.

Whereas direct computation of all N DFT frequencies according to the DFT equation would require N^2complex multiplies and N^2-N complex additions (for complex-valued data), by reusing the results of the two short-length DFTs as illustrated in Figure. Figure depicts a basic butterfly operation that requires just N/2 twiddle-factor multiplies per step. It is worth noting that after combining the twiddle factors into a single term on the lower branch, the remaining butterfly is a length-2 DFT! According to the theory of multi-dimensional index maps, this must be the case, and FFTs of any factorable length can be composed of sequential stages of shorter-length FFTs with twiddle-factor multiplications in between.

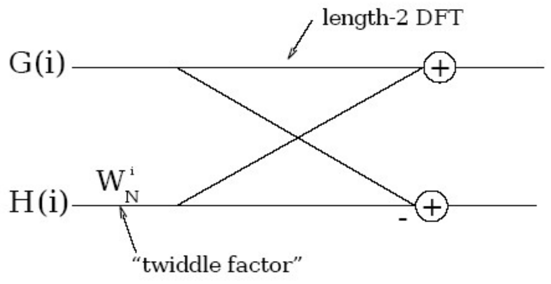


Fig 3.2: Radix-2 DIT butterfly simplification: both operations produce the same outputs

3.2 FFT with radix-2 decimation-in-time:

To save computation, the same radix-2 decimation in time can be performed recursively to the two length N/2 DFTs. The radix-2 DIT FFT method is the result of applying shorter and shorter DFTs until they reach length-2.

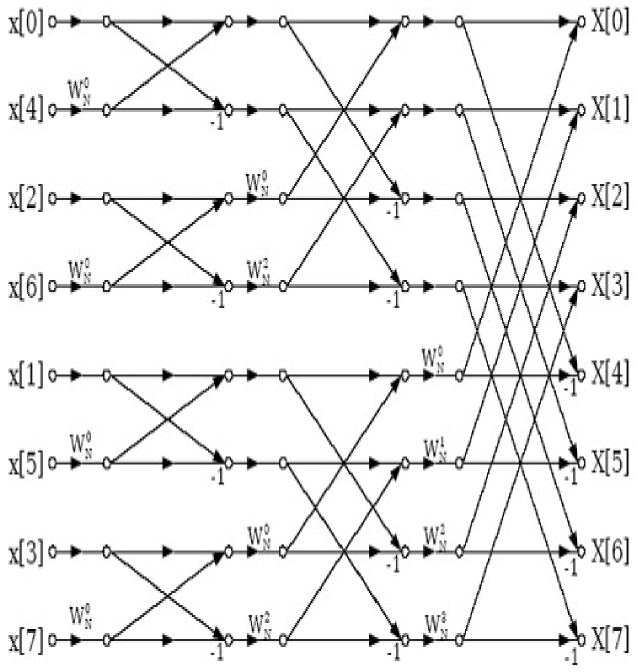


Fig 3.3: Radix-2 Decimation-in-Time FFT Algorithm for a length-8 signal

The entire radix-2 decimation-in-time decomposition shown in Figure with the simpler butterflies includes M=$\log_2 N$ steps, each with N/2 butterflies. Each butterfly needs one complicated multiplication and two additions.

CHAPTER 4

IMPLEMENTATION OF FFT ALGORITHM

The Decimation – In – Time (DIT) FFT Algorithm uses the "divide – and – conquer" approach. This is possible if the length of the sequence N is chosen as $N = r^m$. Here, r is called the radix of the FFT Algorithm. The most practically implemented choice for r = 2 leads to radix – 2 FFT Algorithms. So, with N = 2^m, the efficient computation is achieved by breaking the N – point DFT into two $N/2$ - point DFTs, then breaking each $N/2$ - point DFT into two $N/4$ - point DFTs and continuing this process until 2 – point DFTs are obtained.

4.1 Implementation of 8-Point Radix2 DIT-FFT Algorithm:

Initially the input 8-Bit sequence is sent to the butterfly unit where the corresponding 2-point DFT is calculated i.e. on multiplication and addition of twiddle factors takes place.

This process occurs in 3 stages:

Stage: 1 In this stage 2-Point DFT takes place for x(0), x(4); x(2), x(6); x(1), x(5); x(3), x(7) i.e. bit reversal order

Stage: 2 In here the output of (1st 2-Point DFT, 2nd 2-Point DFT); (3rd 2-Point DFT, 4th 2-Point DFT) will be combined.

Stage: 3 In this stage the resultant 4-Point DFTs from previous stage are combined and output will be obtained in natural order.

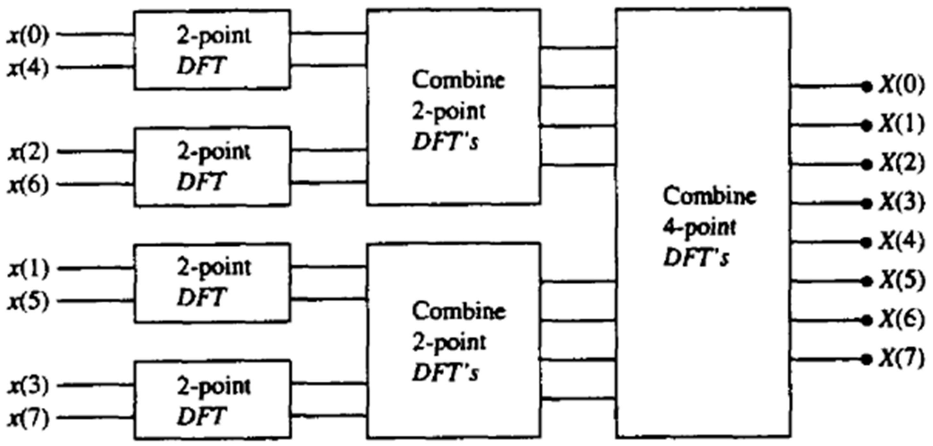


Fig 4.1: Block diagram of 8-Point Radix2 DIT-FFT Algorithm

4.2 <u>Decimation – In – Time (DIT) FFT Algorithm</u>:

In this Algorithm, the time – domain sequence x[n] is decimated into two $N/2$ – point sequences, one composed of even – indexed values of x[n], and other composed of odd – indexed values of x[n].i.e.

$$g[n] = x[2n] \quad (4.1)$$

$$and, h[n] = x[2n+1] \quad (4.2)$$

The N – point DFT of x[n] is given by

$$X(k) = \sum_{n=0}^{N-1} x[n] W_N^{nk}\,, k = 0,1,2, \dots N-1 \quad (4.3)$$

Equation 4.3 can be re-written as

$$X(k) = \sum_{n=0,even}^{N-1} x[n] W_N^{nk} + \sum_{n=0,odd}^{N-1} x[n] W_N^{nk}$$

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

$$= \sum_{n=0}^{\frac{N}{2}-1} x[2n]W_N^{2n} \ + W_N^k \sum_{n=0}^{\frac{N}{2}-1} x[2n+1]W_N^{2nk} \quad (4.4)$$

Using the third property of the twiddle factor, W_N, the above equation 4.4 can be rewritten as

$$= \sum_{n=0}^{\frac{N}{2}-1} x[2n]W_{\frac{N}{2}}^{nk} + W_N^k \sum_{n=0}^{\frac{N}{2}-1} x[2n+1]W_{\frac{N}{2}}^{nk} \quad (4.5)$$

Using equations 4.1 & 4.2 in the above equation, we obtain

$$X(k) = \sum_{n=0}^{\frac{N}{2}-1} g[2n]W_{\frac{N}{2}}^{nk} + W_N^k \sum_{n=0}^{\frac{N}{2}-1} h[n]W_{\frac{N}{2}}^{nk} \ or\ X(k) = G(k) + W_N^k H(k) \quad (4.6)$$

Where G(*k*) and H(*k*) are the N/2 – point DFTs of g[n] and h[n] respectively. So, G(*k*) and H(*k*) are periodic with period N/2 .i.e.,

$$G\left(k + \frac{N}{2}\right) = G(k) \quad (4.7)$$

$$and, H\left(k + \frac{N}{2}\right) = H(k) \quad (4.8)$$

And using the symmetry property of the twiddle factor, WN, and equations 4.7 & 4.8.

$$X\left(k + \frac{N}{2}\right) = G(k) - W_N^k H(k) \quad (4.9)$$

Equations 4.6 and 4.9 result in the following butterfly diagram

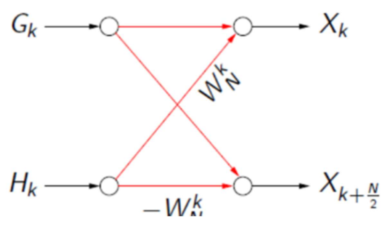


Fig 4.2: Butterfly diagram for G_k and H_k multiplied with twiddle factors

For example, for N = 8, the DFT points in terms of G and H are

$$X(0) = G(0) + W_N^0 H(0) \quad (4.10)$$

$$X(1) = G(1) + W_N^1 H(1) \quad (4.11)$$

$$X(2) = G(2) + W_N^2 H(2) \quad (4.12)$$

$$X(3) = G(3) + W_N^3 H(3) \quad (4.13)$$

For the remaining 4 points X(4) to X(7), we use equations 4.7, 4.8 and 4.9 to get

$$X(4) = G(0) - W_N^0 H(0) \quad (4.14)$$

$$X(5) = G(1) - W_N^1 H(1) \quad (4.15)$$

$$X(6) = G(2) - W_N^2 H(2) \quad (4.16)$$

$$X(7) = G(3) - W_N^3 H(3) \quad (4.17)$$

The butterfly diagram for the above set of equations 4.10, 4.11, 4.12, 4.13, 4.14, 4.15, 4.16, 4.17 is

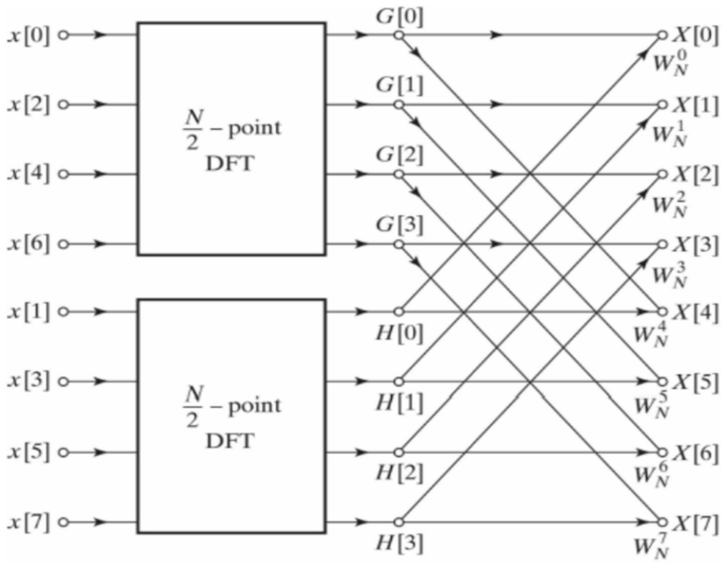


Fig 4.3: Butterfly diagram for stage-3 in 8-point radix-2 FFT algorithm

The above process is repeated for calculating the N/2 point DFTs of g[n] and h[n], and this is continued till we get two point DFTs. Once we reach a two – point sequence, say p[n]={p[0], p[1]}, its 2 – point DFT would be as equation 4.10.

$$p(k) = \sum_{n=0}^{1} p[n]W_2^{nk}, \qquad k = 0,1 \quad (4.10)$$

$$or\ p(k) = p[0] + p[1]W_2^k, \qquad k = 0,1$$

$$\Rightarrow p(0) = p[0] + p[1] \ \ and\ \ p(1) = p[0] + W_2 p[1] = p[0] - p[1]$$

The overall butterfly diagram for DIT FFT Algorithm for N =8 is

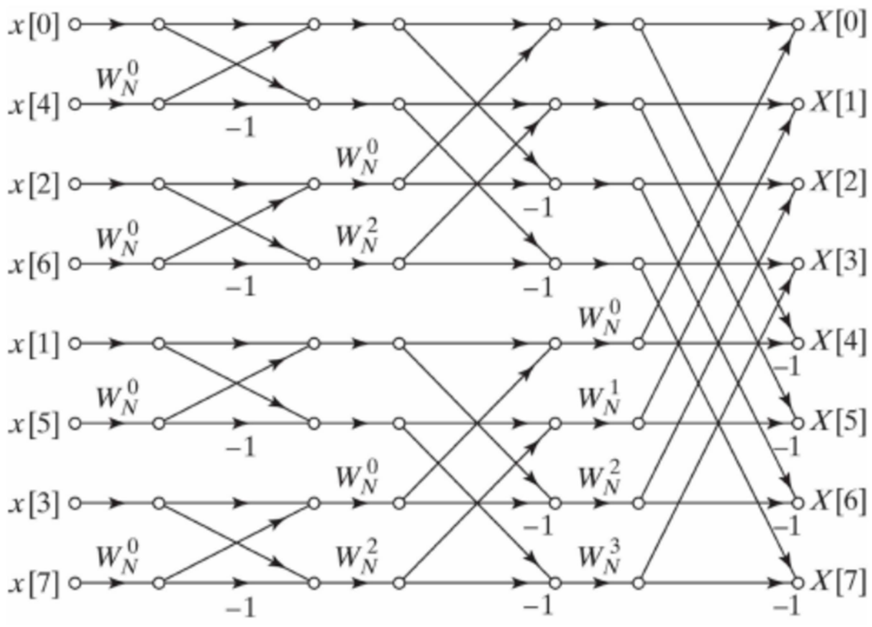


Fig 4.4: Butterfly diagram for DIT FFT Algorithm for N =8

Due to repeated decimations, the input sequence is scrambled, and the order of the final input sequence is obtained as follows

Table 4.1: Sequence table of DIT FFT Algorithm for N =8

Input sequence	Index	Binary form of index	Bit reversed form of index	Decimal representation	Final input sequence order
x[0]	0	000	000	0	x[0]
x[1]	1	001	100	4	x[4]
x[2]	2	010	010	2	x[2]
x[3]	3	011	110	6	x[6]
x[4]	4	100	001	1	x[1]
x[5]	5	101	101	5	x[5]
x[6]	6	110	011	3	x[3]
x[7]	7	111	111	7	x[7]

4.3 Flow Chart of Implementation:

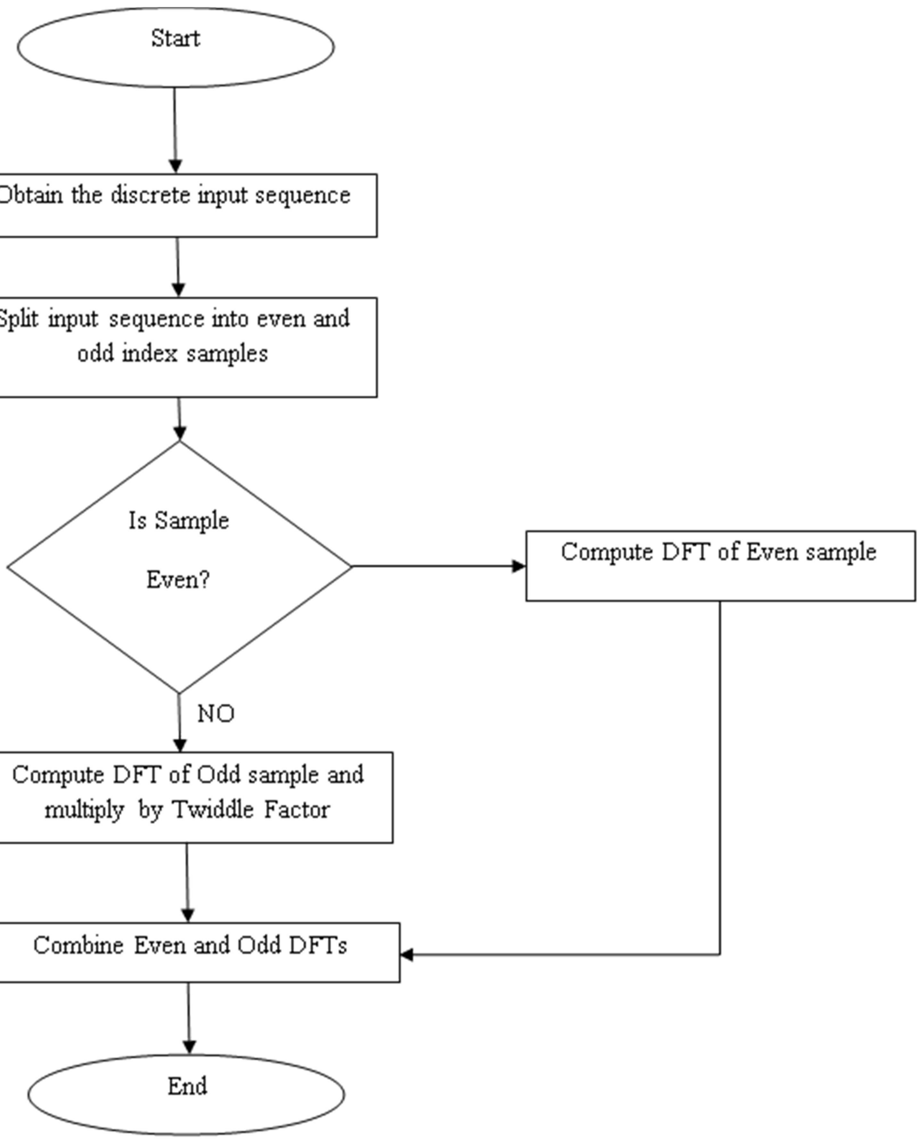

Start
Obtain the discrete input sequence
Split input sequence into even and odd index samples
Is Sample Even?
Compute DFT of Even sample
NO
Compute DFT of Odd sample and multiply by Twiddle Factor
Combine Even and Odd DFTs
End

Fig 4.5: Flow chart of implementation for DIT-FFT Algorithm

CHAPTER 5

SIMULATION RESULTS

5.1 Behavioural Simulation:

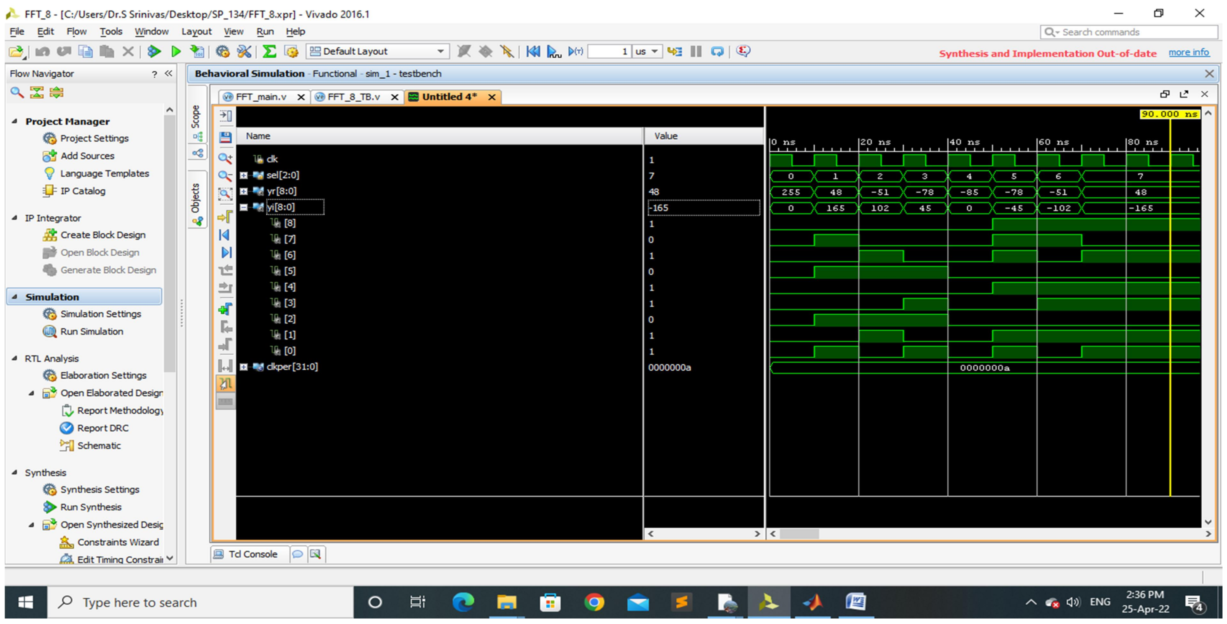

FFT_8 - [C:/Users/Dr.S Srinivas/Desktop/SP_134/FFT_8.xpr] - Vivado 2016.1
Behavioral Simulation - Functional - sim_1 - testbench
Synthesis and Implementation Out-of-date
clk
sel[2:0]
yr[8:0]
yi[8:0]
clkper[31:0]
0000000a
90.000 ns

Fig 5.1: Behavioural simulation waveform

In the above figure, we can observe the FFT simulation outputs. For the given predetermined inputs and respective clock signal and selection line input; the real and imaginary parts of the FFT output sequence is obtained.

"clk" represents clock signal, "sel" represents the output bit selection line i.e., it selects the one out of 8 output elements in the output sequence "y".

Where "yr" is real part of "y", "yi" is imaginary part of "y".

5.2 Schematics:

Top level Schematic of FFT model:

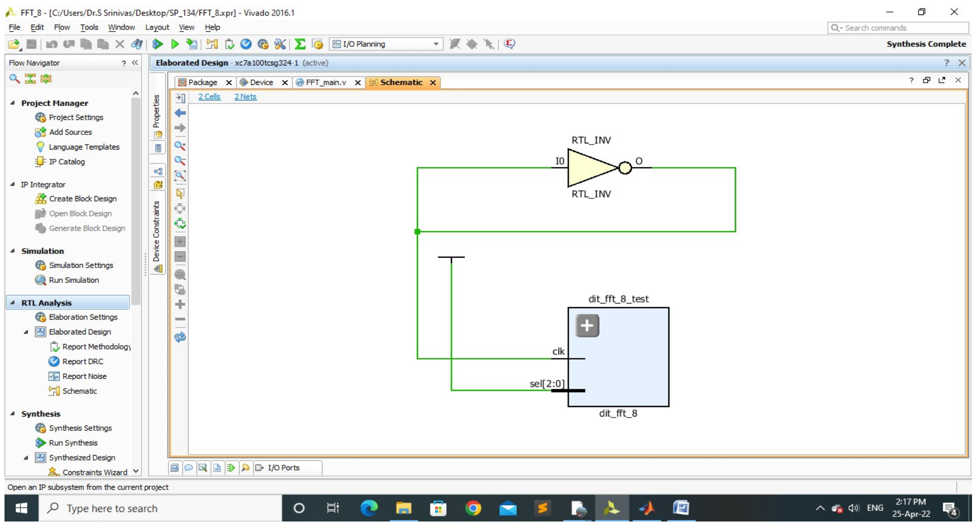

FFT_8 - [C:/Users/Dr.S Srinivas/Desktop/SP_134/FFT_8.xpr] - Vivado 2016.1
Synthesis Complete
Flow Navigator
Elaborated Design - xc7a100tcsg324-1 (active)
Package
Device
FFT_main.v
Schematic
2 Cells
2 Nets
RTL_INV
I0
O
RTL_INV
dit_fft_8_test
clk
sel[2:0]
dit_fft_8
Open an IP subsystem from the current project
Type here to search
2:17 PM
25-Apr-22

Fig 5.2: Top level Schematic of FFT model

Lower Level Schematic of FFT model:

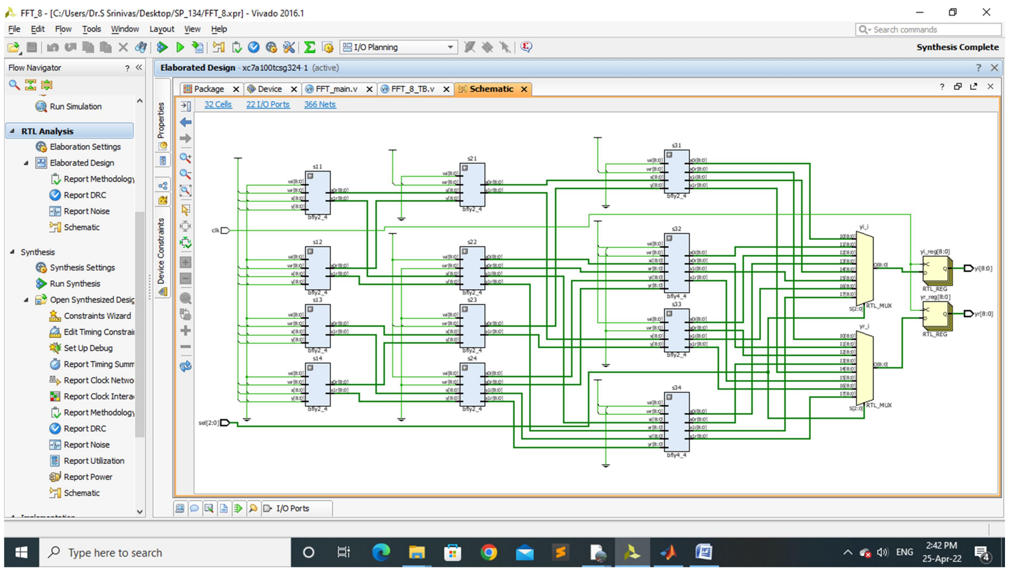


Fig 5.3: Detailed schematic of FFT model

Synthesized Schematic of FFT Processor:

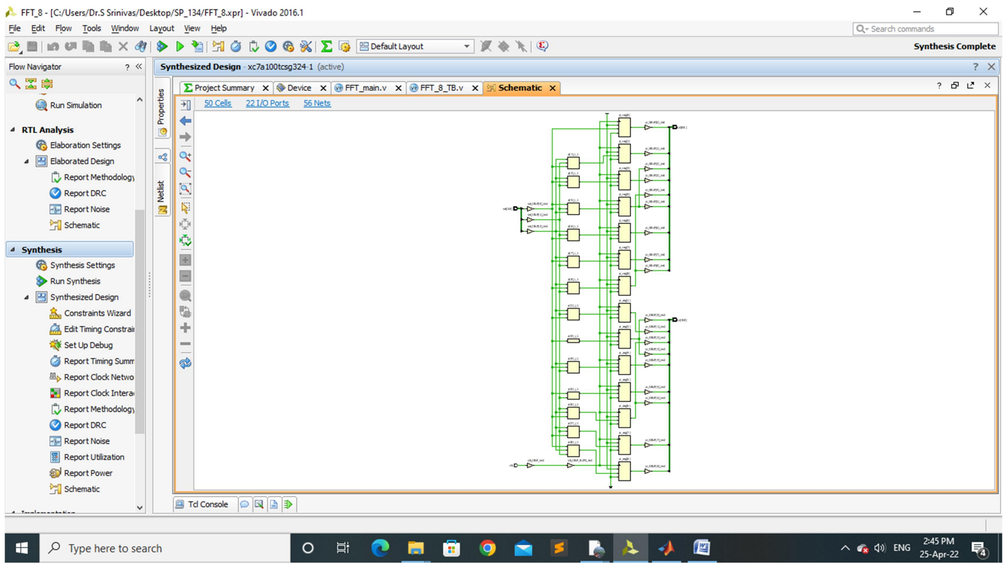


Fig 5.4: Synthesized schematic

5.3 Device outlook after Synthesis:

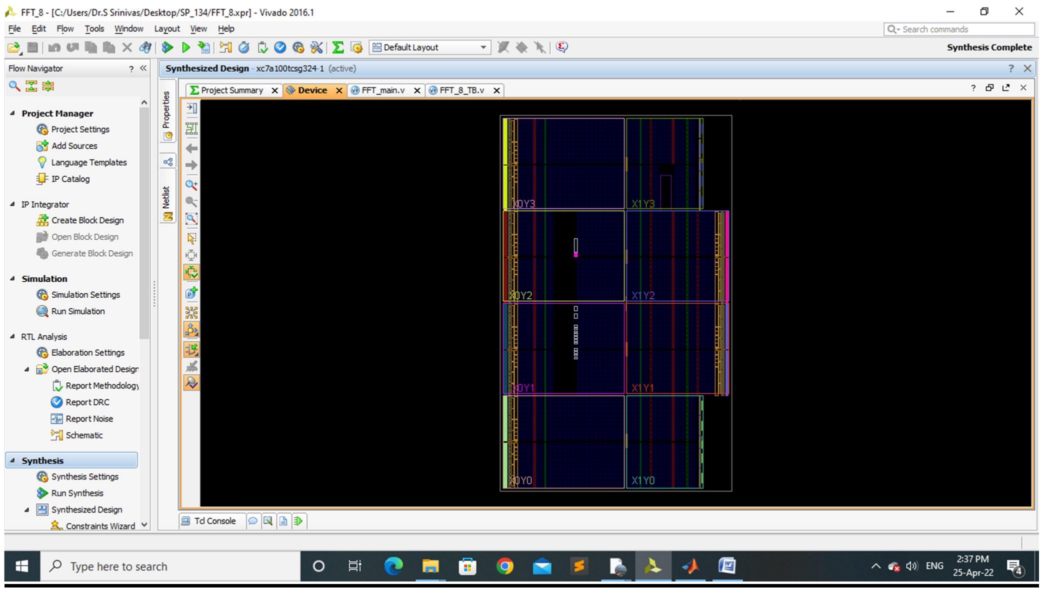


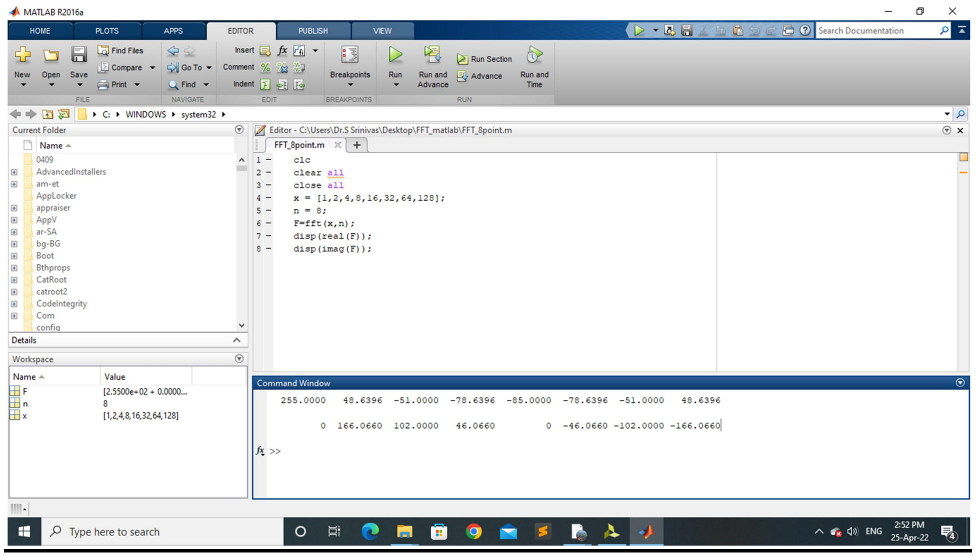
Fig 5.5: Device outlook

5.4 <u>Output Sequence displayed in Vivado and Matlab</u>:

<u>Output sequence displayed in VIVADO console</u>:

```
# run 1000ns
                   0yr=011111111,yi=000000000
                  10yr=000110000,yi=010100101
                  20yr=111001101,yi=001100110
                  30yr=110110010,yi=000101101
                  40yr=110101011,yi=000000000
                  50yr=110110010,yi=111010011
                  60yr=111001101,yi=110011010
                  70yr=000110000,yi=101011011
```

<u>Output sequence displayed in Matlab Command window</u>:



Editor - C:\Users\Dr.S Srinivas\Desktop\FFT_matlab\FFT_8point.m

```
clc
clear all
close all
x = [1,2,4,8,16,32,64,128];
n = 8;
F=fft(x,n);
disp(real(F));
disp(imag(F));
```

Command Window

```
  255.0000   48.6396  -51.0000  -78.6396  -85.0000  -78.6396  -51.0000   48.6396

         0  166.0660  102.0000   46.0660         0  -46.0660 -102.0000 -166.0660

>>
```

Fig 5.6: Matlab Command window

5.5 Utilization Report for the Design:

```
Utilization Design Information

Table of Contents
-----------------
1. Slice Logic
1.1 Summary of Registers by Type
2. Memory
3. DSP
4. IO and GT Specific
5. Clocking
6. Specific Feature
7. Primitives
8. Black Boxes
9. Instantiated Netlists

1. Slice Logic
--------------

+-------------------------+------+-------+-----------+-------+
|        Site Type        | Used | Fixed | Available | Util% |
+-------------------------+------+-------+-----------+-------+
| Slice LUTs*             |    7 |     0 |     63400 |  0.01 |
|   LUT as Logic          |    7 |     0 |     63400 |  0.01 |
|   LUT as Memory         |    0 |     0 |     19000 |  0.00 |
| Slice Registers         |   14 |     0 |    126800 |  0.01 |
|   Register as Flip Flop |   14 |     0 |    126800 |  0.01 |
|   Register as Latch     |    0 |     0 |    126800 |  0.00 |
| F7 Muxes                |    0 |     0 |     31700 |  0.00 |
| F8 Muxes                |    0 |     0 |     15850 |  0.00 |
+-------------------------+------+-------+-----------+-------+
* Warning! The Final LUT count, after physical optimizations and full
implementation, is typically lower. Run opt_design after synthesis, if not
already completed, for a more realistic count.


1.1 Summary of Registers by Type
--------------------------------
+-------+--------------+-------------+--------------+
| Total | Clock Enable | Synchronous | Asynchronous |
+-------+--------------+-------------+--------------+
| 0     |            _ |           - |            - |
| 0     |            _ |           - |          Set |
| 0     |            _ |           - |        Reset |
| 0     |            _ |         Set |            - |
| 0     |            _ |       Reset |            - |
| 0     |          Yes |           - |            - |
| 0     |          Yes |           - |          Set |
| 0     |          Yes |           - |        Reset |
| 0     |          Yes |         Set |            - |
| 14    |          Yes |       Reset |            - |
+-------+--------------+-------------+--------------+
2. Memory
---------

+----------------+------+-------+-----------+-------+
|   Site Type    | Used | Fixed | Available | Util% |
+----------------+------+-------+-----------+-------+
| Block RAM Tile |    0 |     0 |       135 |  0.00 |
|   RAMB36/FIFO* |    0 |     0 |       135 |  0.00 |
|   RAMB18       |    0 |     0 |       270 |  0.00 |
+----------------+------+-------+-----------+-------+
* Note: Each Block RAM Tile only has one FIFO logic available and
therefore can accommodate only one FIFO36E1 or one FIFO18E1. However, if a
FIFO18E1 occupies a Block RAM Tile, that tile can still accommodate a
RAMB18E1


3. DSP
------

+-----------+------+-------+-----------+-------+
| Site Type | Used | Fixed | Available | Util% |
+-----------+------+-------+-----------+-------+
| DSPs      |    0 |     0 |       240 |  0.00 |
+-----------+------+-------+-----------+-------+


4. IO and GT Specific
---------------------

+-----------------------------+------+-------+-----------+-------+
|          Site Type          | Used | Fixed | Available | Util% |
+-----------------------------+------+-------+-----------+-------+
| Bonded IOB                  |   22 |     0 |       210 | 10.48 |
| Bonded IPADs                |    0 |     0 |         2 |  0.00 |
| PHY_CONTROL                 |    0 |     0 |         6 |  0.00 |
| PHASER_REF                  |    0 |     0 |         6 |  0.00 |
| OUT_FIFO                    |    0 |     0 |        24 |  0.00 |
| IN_FIFO                     |    0 |     0 |        24 |  0.00 |
| IDELAYCTRL                  |    0 |     0 |         6 |  0.00 |
| IBUFDS                      |    0 |     0 |       202 |  0.00 |
| PHASER_OUT/PHASER_OUT_PHY   |    0 |     0 |        24 |  0.00 |
| PHASER_IN/PHASER_IN_PHY     |    0 |     0 |        24 |  0.00 |
| IDELAYE2/IDELAYE2_FINEDELAY |    0 |     0 |       300 |  0.00 |
| ILOGIC                      |    0 |     0 |       210 |  0.00 |
| OLOGIC                      |    0 |     0 |       210 |  0.00 |
+-----------------------------+------+-------+-----------+-------+
5. Clocking
-----------
+------------+------+-------+-----------+-------+
|  Site Type | Used | Fixed | Available | Util% |
+------------+------+-------+-----------+-------+
| BUFGCTRL   |    1 |     0 |        32 |  3.13 |
| BUFIO      |    0 |     0 |        24 |  0.00 |
| MMCME2_ADV |    0 |     0 |         6 |  0.00 |
| PLLE2_ADV  |    0 |     0 |         6 |  0.00 |
| BUFMRCE    |    0 |     0 |        12 |  0.00 |
| BUFHCE     |    0 |     0 |        96 |  0.00 |
| BUFR       |    0 |     0 |        24 |  0.00 |
+------------+------+-------+-----------+-------+

6. Specific Feature
-------------------
+-------------+------+-------+-----------+-------+
|  Site Type  | Used | Fixed | Available | Util% |
+-------------+------+-------+-----------+-------+
| BSCANE2     |    0 |     0 |         4 |  0.00 |
| CAPTUREE2   |    0 |     0 |         1 |  0.00 |
| DNA_PORT    |    0 |     0 |         1 |  0.00 |
| EFUSE_USR   |    0 |     0 |         1 |  0.00 |
| FRAME_ECCE2 |    0 |     0 |         1 |  0.00 |
| ICAPE2      |    0 |     0 |         2 |  0.00 |
| PCIE_2_1    |    0 |     0 |         1 |  0.00 |
| STARTUPE2   |    0 |     0 |         1 |  0.00 |
| XADC        |    0 |     0 |         1 |  0.00 |
+-------------+------+-------+-----------+-------+

7. Primitives
-------------
+----------+------+---------------------+
| Ref Name | Used | Functional Category |
+----------+------+---------------------+
| OBUF     |   18 |                  IO |
| FDRE     |   14 |        Flop & Latch |
| LUT3     |   11 |                 LUT |
| IBUF     |    4 |                  IO |
| LUT2     |    1 |                 LUT |
| LUT1     |    1 |                 LUT |
| BUFG     |    1 |               Clock |
+----------+------+---------------------+

8. Black Boxes
--------------
+----------+------+
| Ref Name | Used |
+----------+------+
9. Instantiated Netlists
------------------------
+----------+------+
| Ref Name | Used |
+----------+------+
```

5.6 Vivado Synthesis Report:

```
Start RTL Component Statistics
---------------------------------------------------------------------------
-------
Detailed RTL Component Info :
+---Adders :
	   4 Input      9 Bit       Adders := 6
	   3 Input      9 Bit       Adders := 2
+---Registers :
	                9 Bit    Registers := 2
---------------------------------------------------------------------------
-------
Finished RTL Component Statistics
---------------------------------------------------------------------------
-------
---------------------------------------------------------------------------
-------
Start RTL Hierarchical Component Statistics
---------------------------------------------------------------------------
-------
Hierarchical RTL Component report
Module dit_fft_8
Detailed RTL Component Info :
+---Registers :
	                9 Bit    Registers := 2
Module bfly4_4
Detailed RTL Component Info :
+---Adders :
	   4 Input      9 Bit       Adders := 3
	   3 Input      9 Bit       Adders := 1
---------------------------------------------------------------------------
-------
Finished RTL Hierarchical Component Statistics


Start Writing Synthesis Report
---------------------------------------------------------------------------
-------

Report BlackBoxes:
+-+--------------+----------+
| |BlackBox name |Instances |
+-+--------------+----------+
+-+--------------+----------+
Report Cell Usage:
+------+-----+------+
|      |Cell |Count |
+------+-----+------+
|1     |BUFG |     1|
|2     |LUT1 |     1|
|3     |LUT2 |     1|
|4     |LUT3 |    11|
|5     |FDRE |    14|
|6     |IBUF |     4|
|7     |OBUF |    18|
+------+-----+------+

Report Instance Areas:
+------+---------+-------+------+
|      |Instance |Module |Cells |
+------+---------+-------+------+
|1     |top      |       |    50|
+------+---------+-------+------+
---------------------------------------------------------------------------
-------
Finished Writing Synthesis Report : Time (s): cpu = 00:00:14 ; elapsed =
00:00:21 . Memory (MB): peak = 953.789 ; gain = 479.191
-----------------------------------------------------------------
```

5.7 <u>Matlab vs Vivado</u>:

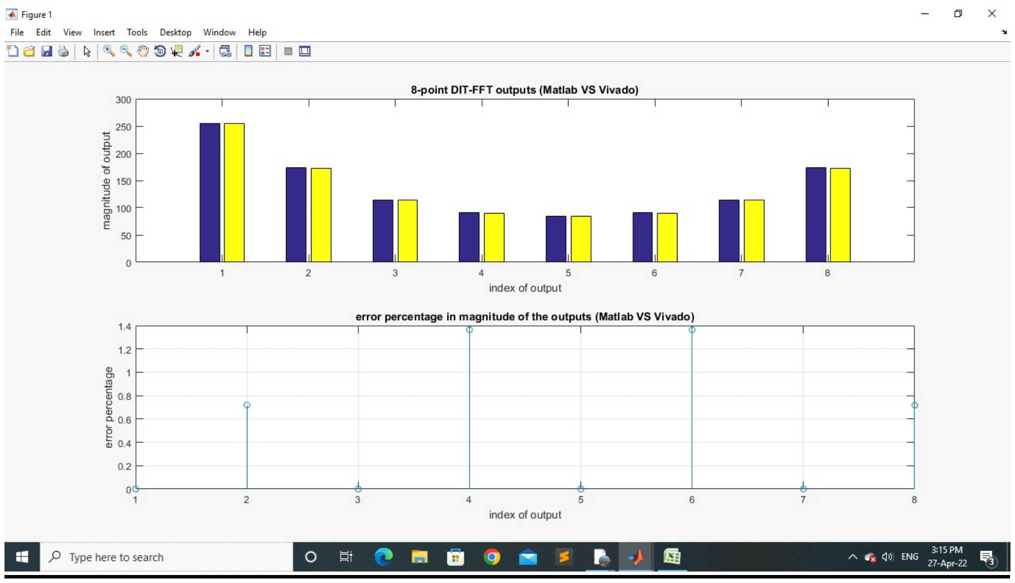
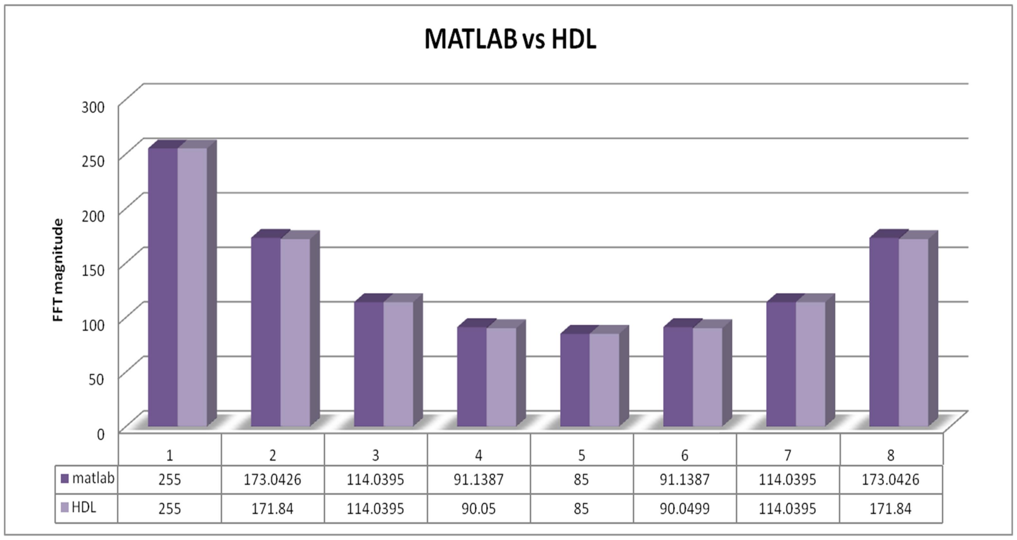


Figure 1
File Edit View Insert Tools Desktop Window Help
8-point DIT-FFT outputs (Matlab VS Vivado)
magnitude of output
index of output
error percentage in magnitude of the outputs (Matlab VS Vivado)
error percentage
index of output
Type here to search
3:15 PM
27-Apr-22

Fig 5.7: Matlab and Vivado FFT output sequence magnitudes comparison and error percentage



	1	2	3	4	5	6	7	8
▪ matlab	255	173.0426	114.0395	91.1387	85	91.1387	114.0395	173.0426
▪ HDL	255	171.84	114.0395	90.05	85	90.0499	114.0395	171.84

Fig 5.8: Detailed comparison of Matlab and Vivado FFT output sequence magnitudes as Bar graph

5.8 Conclusion and Future Scope:

In this paper, we have implemented the DIT-FFT architecture of Length-8 input sequence, which produce the complex output sequence containing signed real and imaginary parts. Here we also extracted the device on board, schematics (top level, elaborated, synthesized), Tcl console display data for obtained outputs in signed binary form, utilization report, synthesis report. And using Matlab we obtained the Matlab FFT function output for length 8 input sequence.

Here our goal is to compare our HDL model output sequence with respective matlab output for the same input sequence. For that we compared the HDL and Matlab outputs using bargraphs and also found the error percentage.

Overall average error percentage is less than 1%. By this we can say our model is efficient as it is very close to the theoretical values.

We can extend this model to longer length inputs and also we can introduce the pipelining structure for faster and accurate performance and resource sharing can also be done.

CHAPTER 6: APPENDIX

APPENDIX-A

OVERVIEW OF DIGITAL SYSTEM DESIGN

Evolution of Computer System Design:

Digital circuit design has evolved rapidly over the last 25 years.The earliest digital circuits were designed with vacuum tubes and transistors. Integrated circuits were then invented where logic gates were placed on a single chip. The first integrated circuit(IC) chips were SSI (Small Scale Integration) chips where the gate count was very small.

As technologies become sophisticated, designers were able to place circuits with hundreds of gates on a chip. These chips were called MSI (Medium Scale Integration) chips. With the advent of LSI(Large Scale Integration), designers could put thousands of gates on a single chip. At this point, design processes started getting very complicated, and designers felt need to automate these processes. Computer Aided Design (CAD) techniques began to evolve. Chips designers began to use circuit and logic simulation techniques to verify the functionality of building blocks of the order of about 100 transistors. The circuits were still tested on the breadboard, and the layout was done on paper by hand on a graphic computer terminal.

With the advent of VLSI (Very Large Scale Integration) technology, designers could design single chips with more than 100,000 transistors. Because of the complexity of these circuits, it was not possible to verify these circuits on the breadboard. Computer-aided technologies became critical for verification and design of VLSI digital circuits. Computer programs to do automatic placement and routing of circuit layouts also became popular. The designers were now building gate-level digital circuits manually on graphic terminals. They would build small building blocks and then derive higher-level blocks from them. This process would continue until they had built the top-level block. Logic simulators came into existence to verify the functionality of these circuits before they were fabricated on chip.

Design specification:

Digital design flow begins with specification of the design at various level of abstraction. It describes the functionality, interface and overall architecture of digital circuits to be designed. The design specifications generally presented as a document describing a set of functionalities that the final solution will have to provide and set constraints that it must satisfy. At this point, architects do not need to think about how they will implement the circuit.

Behavior Description:

In this process, circuit details and electronic components are not specified. Instead, the behavior of each block at the highest level of abstraction is modeling. The behavioural approach to modeling hardware components is different from circuit design in that it does not necessarily reflect hoe the design is implemented. It is basically an Algorithmic and black box approach to modeling. It accurately models what happens on the inputs and outputs of the box. For example, if you wish to simulate the operation of your custom design connected to a commercial part like a microprocessor. In this case, the microprocessor is complex and its internal operation is irrelevant (only the external behavior is important). Behavior descriptions are important as they corroborate the integrity of design idea.

Behaviour Description to RTL:

The designer starts with an abstract description of the circuit called the behavioural model. This kind of description is used primarily to verify the methodology and functioning of the circuit. The next step is to transform this description something closer to electronic circuitry. In other words, the behavioural description needs to be converted to RTL (Registered transfer language) a functional or RTL description describes a circuit in terms of its registers and the combinational logic between the registers. This behavior synthesis can either be done manually or automatically by software, the essential goal of doing this is to use logic synthesizers that makes this form of description and synthesizes it to sets of registers and combinational logic, which can be readily shipped to FPGA.

<u>Functional Verification and Testing</u>:

With the RTL design, the functional design of our digital system ends and its verification begins. From this point onward, the design process is done with the assistance of CAD tools. The underlying motivation is to remove all possible design errors before proceeding to the expensive chip manufacturing. Verification methodology still lacks any standard or even commonly accepted approach. The industrial approach to verification is functional validation. The functional model of the design is simulated with meaningful input stimuli and the output is checked for expected behavior. This model used for simulation is the RTL.

APPENDIX-B

INTRODUCTION TO VERILOG

Introduction:

Verilog, standardized as IEEE 1364, is a hardware description language (HDL) used to model electronic systems. It is most commonly used in design and verification of digital circuits at the regular -transfer level of abstraction. It is also used in verification of analog circuits and mixed signal circuits HDL's allows the design to be simulated earlier in the design circuits in order to correct errors or experiments with different architectures.

Designs described in HDL are technology independent, easy to design and debug, and are usually more readable than schematics, particularly for large circuits. Verilog can be used to describe designs at four levels if abstractions:

1) Algorithmic level (much like as code if, case and loop statements).
2) Register transfer level (RTL uses registers connected by Boolean equations)
3) Gate level (interconnected AND, NOR etc.).
4) Switch level (the switches are MOS transistors inside gates).

A Verilog design consists of a hierarchy of modules. Modules encapsulate design hierarchy, and communicate with other modules through a set of declared input, output and bidirectional ports.

Internally, a module can contain any combination of the following: net/variable declarations (wire, reg, integer, etc.), concurrent and sequential statement blocks, and instances of other modules (sub-hierarchies).

Features of Verilog HDL:

Verilog HDL offers many useful features for hardware design

- Verilog (verify logic) HDL is general purpose hardware description language that is easy to learn and easy to use. It is similar in syntax to C programming language. Designers with C programming experience will find it easy to learn Verilog HDL.
- Verilog HDL allows different levels of abstraction to be mixed in the same model. Thus a designer can define a hardware model in terms of switches, gates, RTL, or behaviour code. Also a designer needs to learn only one language for stimulus and hierarchical designs.
- Most popular logic synthesis tools support Verilog HDL. This makes it the language of choice for designers.
- All fabrication vendors provide Verilog HDL libraries for post logic synthesis simulation. Thus designing a chip in Verilog HDL allows the widest choice of vendors.
- The Programming language interface (PLI) is a powerful feature that allows the user to custom C code to interact with the internal data structures of Verilog. Designers can customize a Verilog HDL simulator to their need with the Programming language interface (PLI).

Module Declaration:

A module is the principal design entity in Verilog. The first line of a module declaration specifies the name and port list (arguments). The next few lines specifies the I/O type (**input**, **output** or **inout**) and width of each port. The default port with is 1 bit. Then the port variables must be declared **wire**, **reg**. The default is **wire**.

Typically, inputs are **wire** since their data is latched outside the module. Outputs are type **reg**if their signals were stored inside always or **initial** block

Syntax

```
module model_ name(port_list);
input[msb:lsb] input_port_list;
output[msb:lsb] output_port_list;
inout[msb:lsb] inout_port_list;
   …….statements…………
endmodule
```

Example

Module add_sub(add, in1, in2, oot);

Input add;//defaults to wire

input[7:0] in1, in2**, wire** in1, in2;

Output[7:0] oot;

Reg oot;

………statements……..

endmodule

Verilog has four levels of modelling:

1) The switch level Modeling.
2) Gate- level Modeling.
3) The Data-Flow level.
4) The behavioural or procedural level.

Switch level Modeling

A circuit is defined by explicity showing how to constrct it using transistors like pmos and nmos, predefined modules.

Example

module inverter (out,in);

output out;

input in;

Supply0 gnd;

Supply1 vdd;

Nmos x1(out, in, gnd);

Pmos x2(out, in, vdd);

endmodule

<u>Gate level Modeling</u>:

A circuit is defined by explicitly showing how to correct it using logic gates. Predefined modules, and the connections between them. In this first we think of our circuit as a box or module which is encapsulated from its outer environment, in such a way that its only communication with the outer environment, is through input and output ports. We then set out to describe structure within the module by explicitly describing its gates and sub modules, and how they connect with one another as well as to the module ports.

In other words, structural modeling is used to draw a schematic diagram for the circuit. As an example, consider the full-adder below.

Example

Module fulladder (a, b, sum, Cout);

input a, b;

Output sum, Cout;

xor x1(a, b, y);

xor x2(a, b, y);

endmodule

<u>Data-Flow Modeling</u>:

Dataflow modeling uses Boolean expressions and operators. In this we use assign statement.

Example

Module fulladder (a, b, sum, Cout);

input a, b;

output sum, Cout;

assign sum=a^b;

assign Cout=a^b;

endmodule

<u>Behavioural modeling</u>:

It is higher level of modeling where behavior of logic is modeled. Verilog behavioural

Code is inside procedure blocks, but there is an exception: some behavioural code also exist outside procedure blocks.

There are two types of procedural blocks in Verilog

Initial: initial blocks execute only once at time zero (start execution at time zero) **Always:** always blocks loop to execute over and over again; in other words, as other words as the name suggests, it executes always.

An always statement executes repeatedly, it starts and its execution at other 0 ns

Syntax:

always@ (sensitivity list)

Begin

Procedural statements

end

Example

Module fulladder(a, b, clk, sum);

Input a, b, clk;

output sum;

always@ (posedge clk)

begin

sum= a+b;

endmodule

Software Design and Development:

A description of the hardware's structure and behavior is written in a high level hardware description language (usually VHDL or Verilog and those codes is then compiled and downloaded prior to execution. Although schematic capture can be used for design entry but due to more complex designs and the improvement of the language-based tools it has become less popular.

The most distinct difference between hardware and software design is the way a developer must think about the problem. Software developers tend to think sequentially, even when they are tasked to program a multithread application. Most of the time, the source code is always executed in that order. At the design entry phase, hardware designers must think and program in parallel.

All of the input signals are processed in parallel: inside each ore is a series of macro cells and interconnections routed toward their destination output signals. Therefore, the statement of a hardware description language creates structures, all of which are process at the very same time. (Normally the link between each macro cell to another macro cell usually -synchronized to some other signal, like a common clock).

In a typical design, after each design entry is completed the next step is to perform periods of functional simulation. This is where a simulator comes in place. It is used to execute the design and confirms that the correct/required outputs are produced for a given set of test inputs.

This step is to ensure the designer that his/her logic is functionally correct before going on to the next stage development. This is a good practice as compared to simulating a full scale design entry. As the design entry gets more complex, the troubleshooting will be much difficult and time consuming.

Software tools used:

Xilinx Vivado:

Vivado enables developers to synthesize their designs, perform timing analysis examine RTL diagrams, simulate a design's reaction to different stimuli, and configure the target device with the programmer. Vivado is a design environment for FPGA products from Xilinx, and is tightly-coupled to the architecture of such chips, and cannot be used with FPGA products from other vendors.

Language support

The Vivado High-Level Synthesis compiler enables C, C++ and System C programs to be directly targeted into Xilinx devices without the need to manually create RTL. Vivado HLS is widely reviewed to increase developer productivity, and is confirmed to support C++ classes, templates, functions and operator overloading.

Xilinx vivado enables simulation, verification and synthesis for the following languages

- VHDL
- Verilog
- System Verilog

APPENDIX-C

XILINX VIVADO

Xilinx Vivado ISE Design Suite(16.1 version):

Xilinx is a powerful software tool that is used to design, synthesize, simulate, test and verify digital circuit designs. The designer can describe the digital design by either using the schematic entry tool or a hardware description language. In this software we will create VHDL design input files – the hardware description of the logic circuit, compile VHDL source files, create a test bench and simulate the design to make sure of the correct operation of the design (functional simulation). The purpose of this is to give new users an exposure to the basic and necessary steps to implement and examine your own designs using ISE environment. In this, we will design one simple module (OR gate); however, in the future, you will be designing such modules and completing the overall circuit design from these existing files. A VHDL input file in the Xilinx environment consists of: Entity Declarations: module name and interface specifications (I/O) – list of input and output ports; their mode, which is direction of data flow; and data type. Architecture: defines a component's logic operation.

There are different styles for the architecture body: (i) Behavioural – set of sequential assignment statements (ii) Data Flow – set of concurrent assignments o Structural – set of interconnected components A combination of these could be used, but in this tutorial we will use Dataflow. In its simplest form, the architectural body will take the following format, regardless of the style: architecture architecture_name of entity_name is begin …   -- statement end architecture_name;

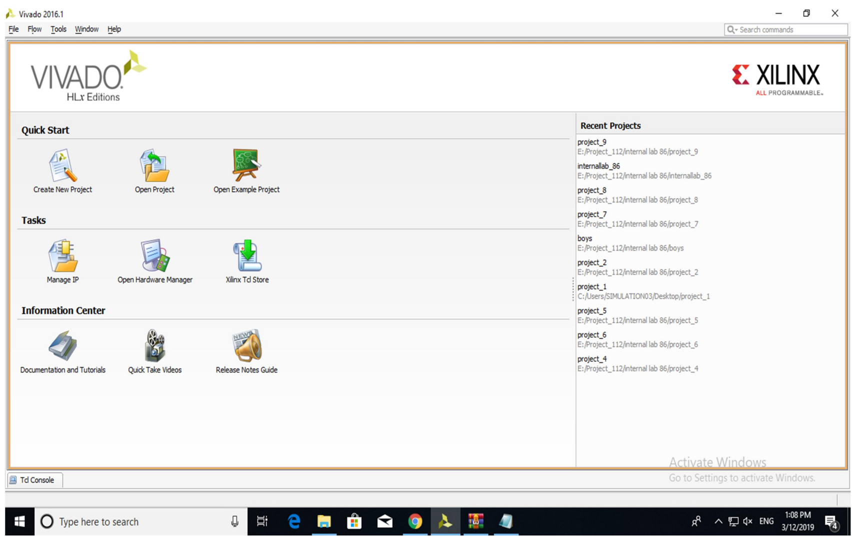
ISE (Integrated Software Environment) is a software tool produced by Xilinx for synthesis and analysis of HDL designs, enabling the developer to synthesize ("compile") their designs, perform timing analysis, examine RTL diagrams, simulate a design's reaction to different simuli, and configure the target device with the programmer.

Xilinx is an American technology company, primarily a supplier of programmable logic devices. It is known for inventing FPGA.

The Xilinx ISE is primarily used for circuit synthesis and design, while the Modelsim logic simulator is used for system-level testing.

<u>ISE Project Navigator</u>:

In this section, we introduce the reader to the main components of an "ISE Project Navigator" window, which allows us to manage our design files and move our design process from creation to synthesis and to simulation phase.



Vivado 2016.1
File Flow Tools Window Help
Search commands
VIVADO HLx Editions
XILINX ALL PROGRAMMABLE.
Quick Start
Create New Project
Open Project
Open Example Project
Tasks
Manage IP
Open Hardware Manager
Xilinx Tcl Store
Information Center
Documentation and Tutorials
Quick Take Videos
Release Notes Guide
Recent Projects
project_9
E:/Project_112/internal lab 86/project_9
internallab_86
E:/Project_112/internal lab 86/internallab_86
project_8
E:/Project_112/internal lab 86/project_8
project_7
E:/Project_112/internal lab 86/project_7
boys
E:/Project_112/internal lab 86/boys
project_2
E:/Project_112/internal lab 86/project_2
project_1
C:/Users/SIMULATION03/Desktop/project_1
project_5
E:/Project_112/internal lab 86/project_5
project_6
E:/Project_112/internal lab 86/project_6
project_4
E:/Project_112/internal lab 86/project_4
Activate Windows
Go to Settings to activate Windows.
Tcl Console
Type here to search
1:08 PM
3/12/2019

Fig 6.1: ISE Project Navigator window

By opening the Xilinx Vivado ISE suite, we will come to see the 3 main points. They are

1) Quick start
2) Tasks
3) Information Center

In the Quick start block, We have create a new project, open project and open example project

In the Tasks, We have Manage IP, open hardware manager, xilinx Td store.

In the Information center, we have documentation and tutorials,quick take videos and release notes guide.

This section describes the four basic steps to working with a project.

Step 1—– Creating a New Project

This creates .xprfile and a working library.

Step 2—– Adding Items to the project

Projects can reference or include source files, folders for organizations, simulations, and any other files you want to associate with the project. You can copy files into the project directory or simply create mappings to files in other locations.

Step 3—– Compiling the Files

This checks syntax and semantics and creates the pseudo machine code that Viavdo uses for simulation.

Step 4—– Simulating a Design

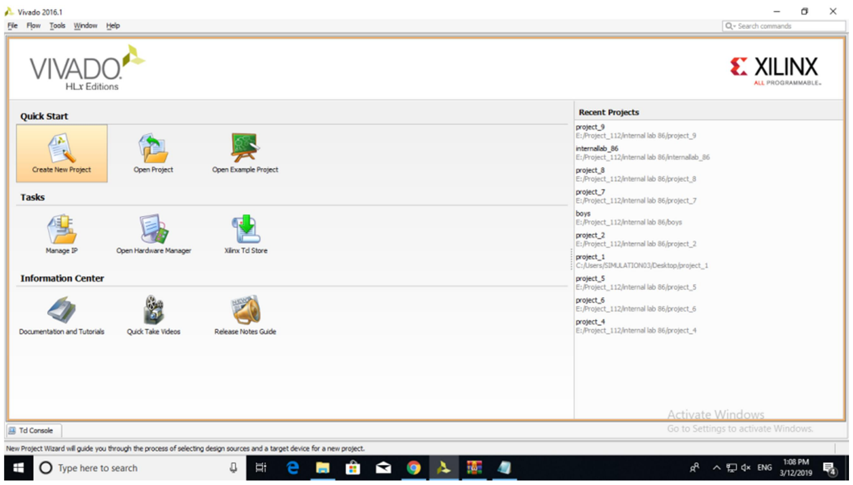
This specifies the design unit you want to simulate and opens a structure tab in the workspace pane.

you specify will be used to create a working library subdirectory within the Project

In order to start ISE double click the desktop icon: Or click**:**

Creating a New Project

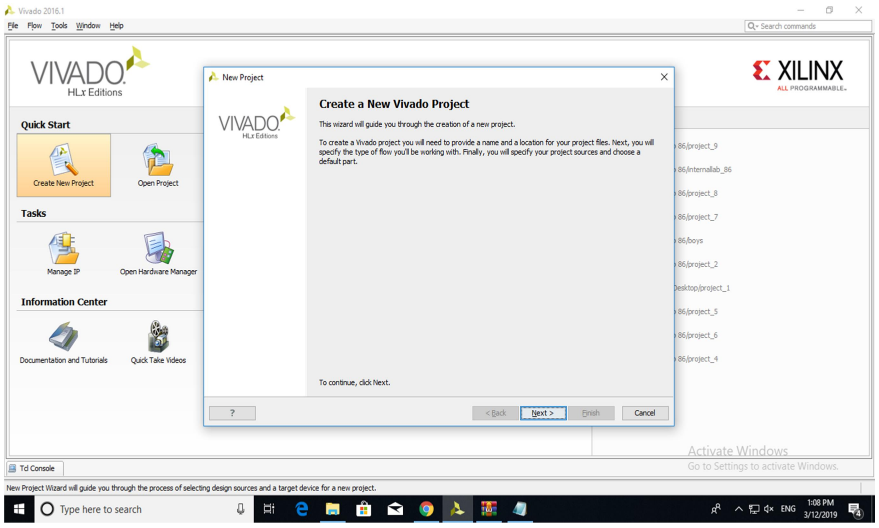
After launching Vivado, from the startup page click the “Create New Project” icon. Alternatively, you can select File -> New Project



New Project Wizard will guide you through the process of selecting design sources and a target device for a new project.

Fig 6.2: Creating new project window

The New Project wizard will launch, click the "Next >" button to proceed



Create a New Vivado Project

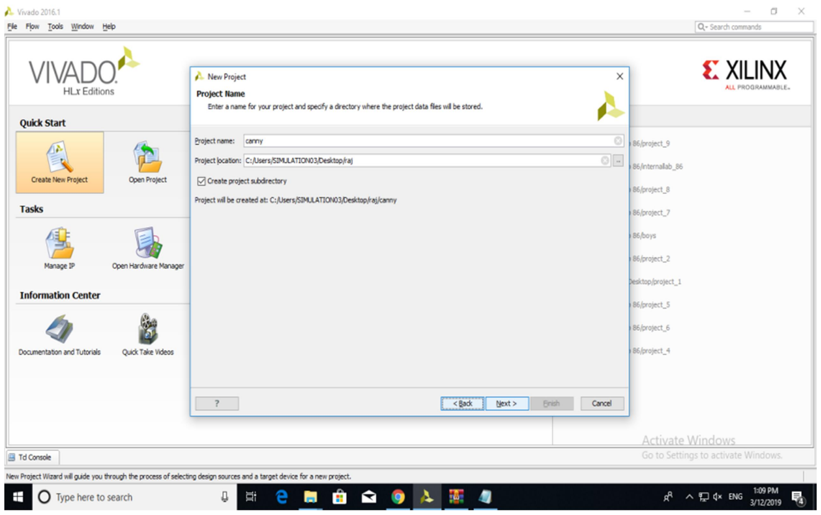
This wizard will guide you through the creation of a new project.

To create a Vivado project you will need to provide a name and a location for your project files. Next, you will specify the type of flow you'll be working with. Finally, you will specify your project sources and choose a default part.

To continue, click Next.

Fig 6.3: Guiding wizard for the project

Enter a project name and select a project location. **Make certain there are NO SPACES in either!** It's not a bad idea to only use letters, numbers, and underscores as well. If necessary simply create a new directory for your Xilinx Vivado projects in your root drive (e.g. C:\Vivado). You will likely always want to select the "Create project sub-directory" check-box as well. This keeps things neatly organized with a directory for each project and helps avoid problems. Click the “Next >” button to proceed.


New Project

Project Name

Enter a name for your project and specify a directory where the project data files will be stored.

Project name: canny

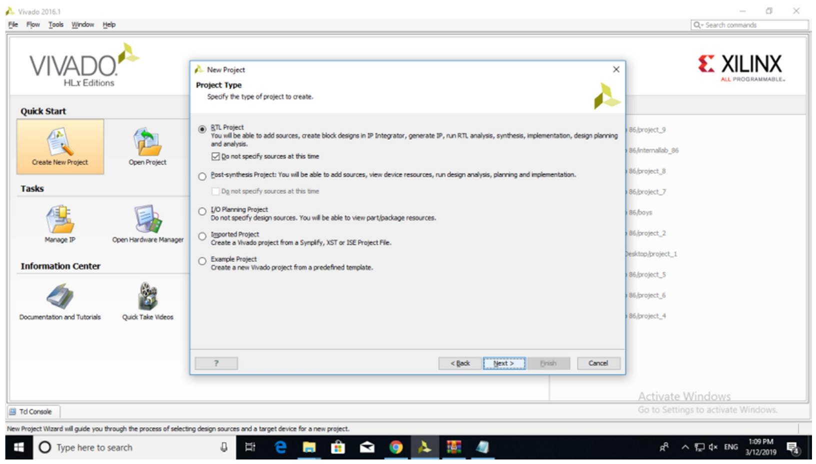
Project location: C:/Users/SIMULATION03/Desktop/raj

☑ Create project subdirectory

Project will be created at: C:/Users/SIMULATION03/Desktop/raj/canny

Fig 6.4: Creating a project name

Select the “RTL Project” radial and select the “Do not specify sources at this time” check-box. If you don’t select the check-box the wizard will take you through some additional steps to optionally add preexisting items such as VHDL or Verilog source files, Vivado IP blocks, and .XDC constraint files for device pin and timing configuration. For this first project you will add the necessary items later. Click the “Next >” button to proceed.


New Project

Project Type

Specify the type of project to create.

[✓ RTL Project]
You will be able to add sources, create block designs in IP Integrator, generate IP, run RTL analysis, synthesis, implementation, design planning and analysis.
☑ Do not specify sources at this time

Post-synthesis Project: You will be able to add sources, view device resources, run design analysis, planning and implementation.
☐ Do not specify sources at this time

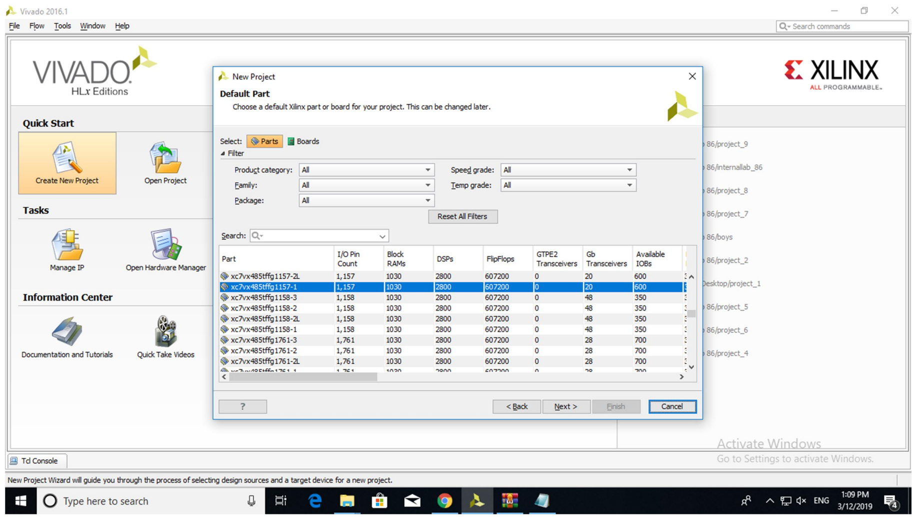
I/O Planning Project
Do not specify design sources. You will be able to view part/package resources.

Imported Project
Create a Vivado project from a Synplify, XST or ISE Project File.

Example Project
Create a new Vivado project from a predefined template.

Fig 6.5: Specifying the RTL project

You need to filter down to and select the specific part number for your project. You can physically read the markings on your chip or refer to your board's documentation to find its part number. In the case of the Basys 3 it's the Artix-7 chip that's on the board, and the filters shown will help you get to the correct device that's highlighted. Once you select the correct device click the "Next >" button to proceed.



New Project

Default Part

Choose a default Xilinx part or board for your project. This can be changed later.

Select: Parts Boards

Filter

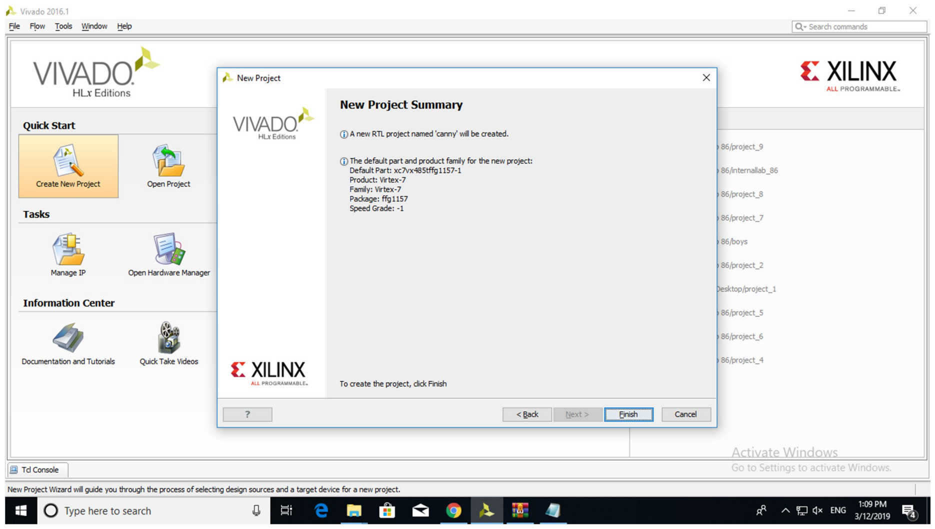
Product category: All
Family: All
Package: All
Speed grade: All
Temp grade: All

Part	I/O Pin Count	Block RAMs	DSPs	FlipFlops	GTPE2 Transceivers	Gb Transceivers	Available IOBs
xc7vx485tffg1157-2L	1,157	1030	2800	607200	0	20	600
xc7vx485tffg1157-1	1,157	1030	2800	607200	0	20	600
xc7vx485tffg1158-3	1,158	1030	2800	607200	0	48	350
xc7vx485tffg1158-2	1,158	1030	2800	607200	0	48	350
xc7vx485tffg1158-2L	1,158	1030	2800	607200	0	48	350
xc7vx485tffg1158-1	1,158	1030	2800	607200	0	48	350
xc7vx485tffg1761-3	1,761	1030	2800	607200	0	28	700
xc7vx485tffg1761-2	1,761	1030	2800	607200	0	28	700
xc7vx485tffg1761-2L	1,761	1030	2800	607200	0	28	700

New Project Wizard will guide you through the process of selecting design sources and a target device for a new project.

Fig 6.6: Choosing a board for project

Click the "Finish" button and Vivado will proceed to create your project as specified.



New Project

New Project Summary

A new RTL project named 'canny' will be created.

The default part and product family for the new project:
Default Part: xc7vx485tffg1157-1
Product: Virtex-7
Family: Virtex-7
Package: ffg1157
Speed Grade: -1

To create the project, click Finish

New Project Wizard will guide you through the process of selecting design sources and a target device for a new project.

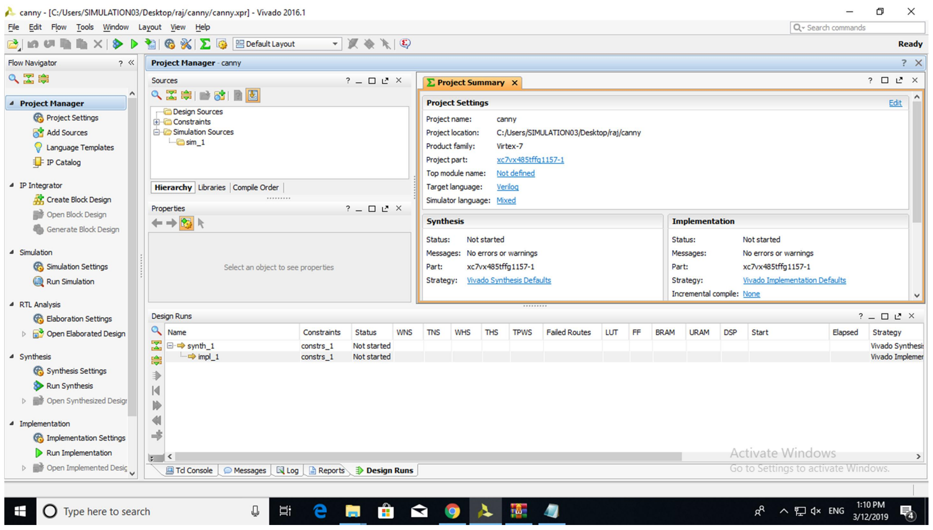
Fig 6.7: Project summary

Steps for Design entry:

Working through the Basic Project Flow:

The Vivado project window contains a lot of information, and the information displayed can change depending on what part of the design you currently have open as you work through the steps of your project. Keep this in mind as you work through this guide, because if you don't see a specific sub-window or sub-window tab it's possible you aren't in the correct part of the design.

The "Flow Navigator" on the left side of the screen has all the major project phases organized from top to bottom in their natural chronological order. You begin in the "Project Manager" portion of the flow and the header at the top of the screen next to the Flow Navigator reflects this. This header and the corresponding highlighted section in the Flow Navigator will tell you which phase of the design you have open.


canny - [C:/Users/SIMULATION03/Desktop/raj/canny/canny.xpr] - Vivado 2016.1

Project Summary

Project Settings

Project name: canny
Project location: C:/Users/SIMULATION03/Desktop/raj/canny
Product family: Virtex-7
Project part: xc7vx485tffg1157-1
Top module name: Not defined
Target language: Verilog
Simulator language: Mixed

Synthesis

Status: Not started
Messages: No errors or warnings
Part: xc7vx485tffg1157-1
Strategy: Vivado Synthesis Defaults

Implementation

Status: Not started
Messages: No errors or warnings
Part: xc7vx485tffg1157-1
Strategy: Vivado Implementation Defaults
Incremental compile: None

Design Runs

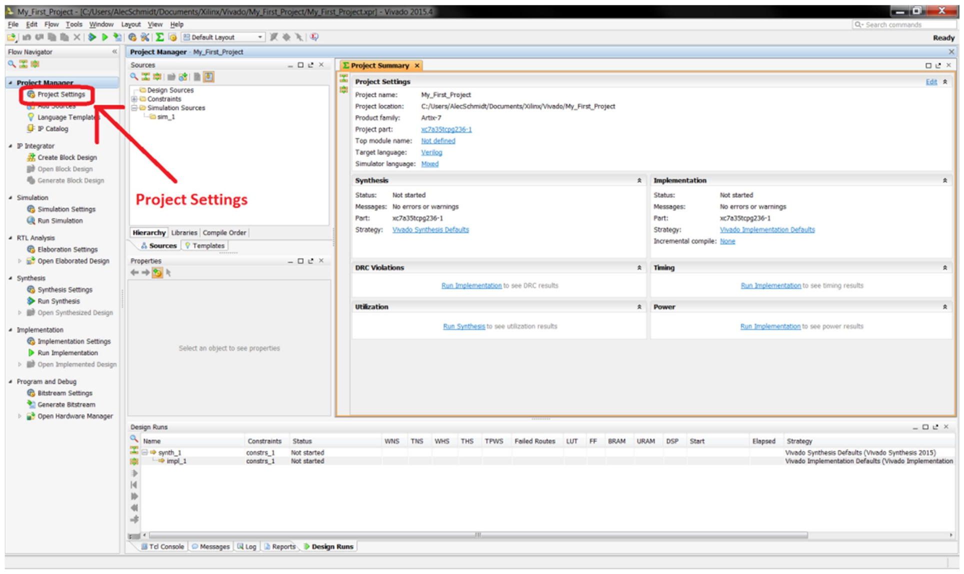
Name	Constraints	Status	WNS	TNS	WHS	THS	TPWS	Failed Routes	LUT	FF	BRAM	URAM	DSP	Start	Elapsed	Strategy
synth_1	constrs_1	Not started														Vivado Synthesi
impl_1	constrs_1	Not started														Vivado Implemer

Fig 6.8: Main window for the project

Project Manager

Project Settings

Begin by clicking on “Project Settings” under the Project Manager phase of the Flow Navigator



Project Settings

Project name:	My_First_Project
Project location:	C:/Users/AlecSchmidt/Documents/Xilinx/Vivado/My_First_Project
Product family:	Artix-7
Project part:	xc7a35tcpg236-1
Top module name:	Not defined
Target language:	Verilog
Simulator language:	Mixed

Synthesis

Status:	Not started
Messages:	No errors or warnings
Part:	xc7a35tcpg236-1
Strategy:	Vivado Synthesis Defaults

Implementation

Status:	Not started
Messages:	No errors or warnings
Part:	xc7a35tcpg236-1
Strategy:	Vivado Implementation Defaults
Incremental compile:	None

DRC Violations: Run Implementation to see DRC results

Timing: Run Implementation to see timing results

Utilization: Run Synthesis to see utilization results

Power: Run Implementation to see power results

Design Runs

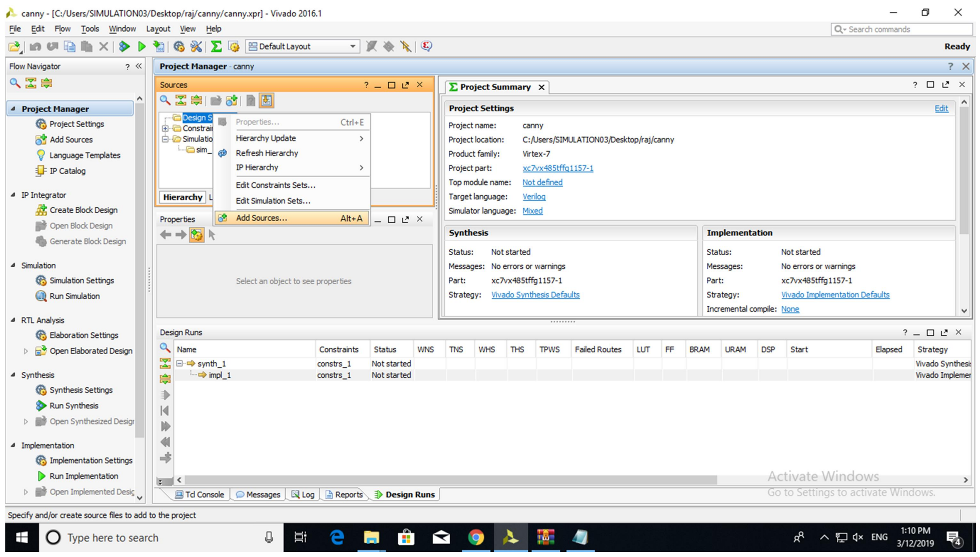
Name	Constraints	Status	WNS	TNS	WHS	THS	TPWS	Failed Routes	LUT	FF	BRAM	URAM	DSP	Start	Elapsed	Strategy
synth_1	constrs_1	Not started														Vivado Synthesis Defaults (Vivado Synthesis 2015)
impl_1	constrs_1	Not started														Vivado Implementation Defaults (Vivado Implementation

Fig 6.9: Project settings window

There are a lot of settings available here for all phases of the project flow, but for now just select “System Verilog” from the drop-down for the “Target language” in the “General” project settings and click the “OK” button.

Add Sources

Now click on “Add Sources” under the Project Manager phase of the Flow Navigator



Project Settings

Project name:	canny
Project location:	C:/Users/SIMULATION03/Desktop/raj/canny
Product family:	Virtex-7
Project part:	xc7vx485tffg1157-1
Top module name:	Not defined
Target language:	Verilog
Simulator language:	Mixed

Synthesis

Status:	Not started
Messages:	No errors or warnings
Part:	xc7vx485tffg1157-1
Strategy:	Vivado Synthesis Defaults

Implementation

Status:	Not started
Messages:	No errors or warnings
Part:	xc7vx485tffg1157-1
Strategy:	Vivado Implementation Defaults
Incremental compile:	None

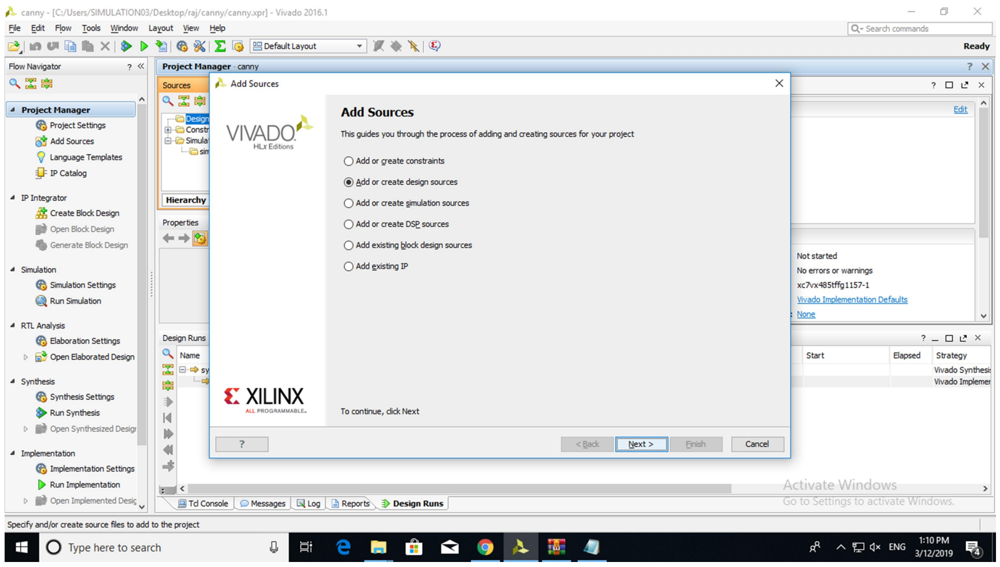
Design Runs

Name	Constraints	Status	Strategy
synth_1	constrs_1	Not started	Vivado Synthesis
impl_1	constrs_1	Not started	Vivado Implemen

Specify and/or create source files to add to the project

Fig 6.10: Adding the source files

Select the “Add or create design sources” radial and then click the “Next >” button.



Add Sources

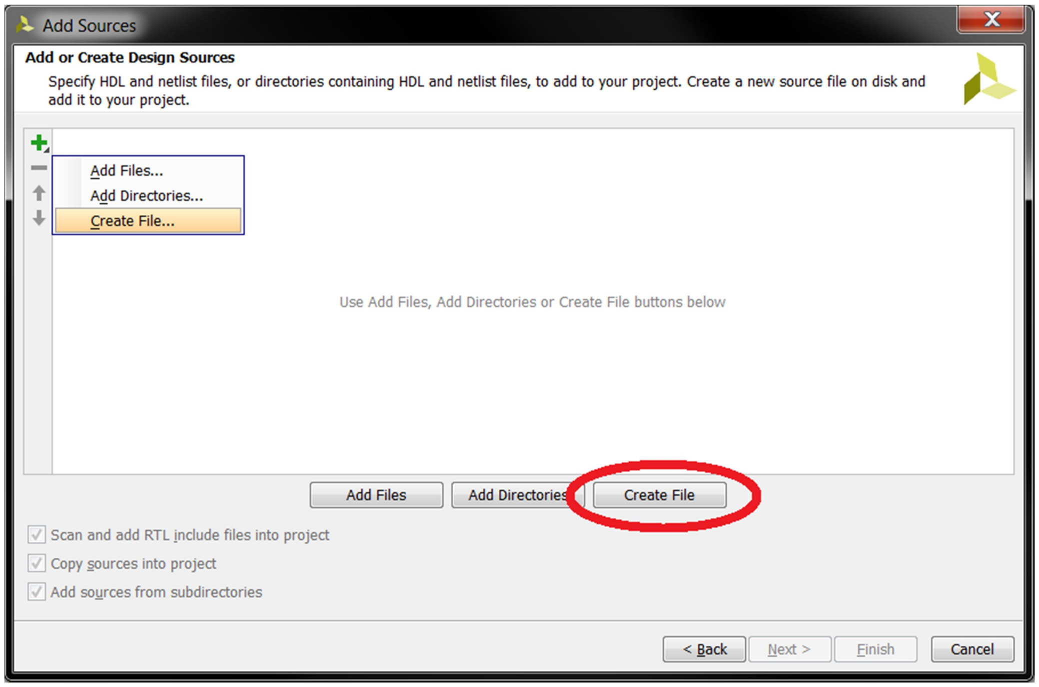
This guides you through the process of adding and creating sources for your project

( ) Add or create constraints
[✓ Add or create design sources]
( ) Add or create simulation sources
( ) Add or create DSP sources
( ) Add existing block design sources
( ) Add existing IP

To continue, click Next

Fig 6.11: Wizard that shows to the design source

Click the "Create File" button or click the green "+" symbol in the upper left corner and select the "Create File…" option.



Add or Create Design Sources

Specify HDL and netlist files, or directories containing HDL and netlist files, to add to your project. Create a new source file on disk and add it to your project.

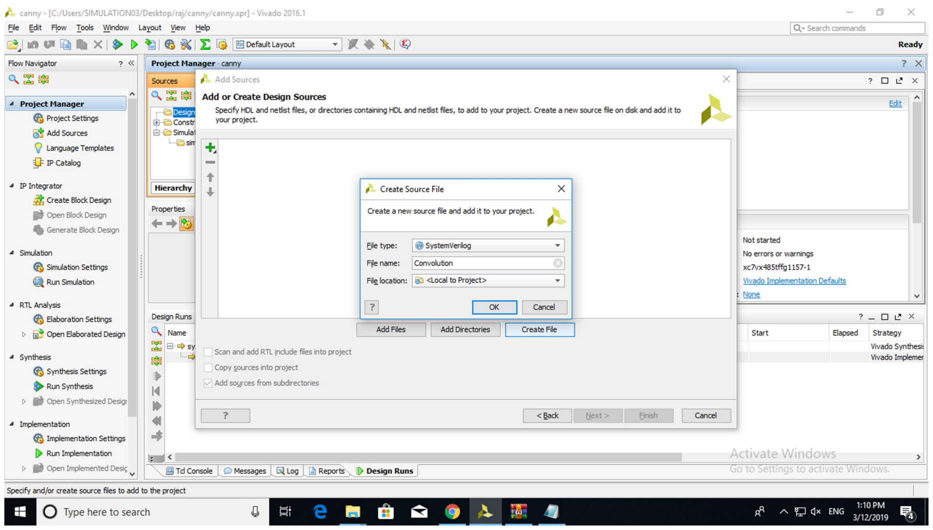
Use Add Files, Add Directories or Create File buttons below

☑ Scan and add RTL include files into project
☑ Copy sources into project
☑ Add sources from subdirectories

Fig 6.12: Creating a new file name for new design source

Make sure the options shown are selected in the "Create Source File" popup, and for the sake of following along enter "convolution (Gaussian filter)" for the "File name". Click the "OK" button when finished.

You can normally enter anything you like for the "File name" as long as it's valid, but **always make certain there are NO SPACES!**



Specify HDL and netlist files, or directories containing HDL and netlist files, to add to your project. Create a new source file on disk and add it to your project.

Create a new source file and add it to your project.

File type: SystemVerilog

File name: Convolution

File location: <Local to Project>

Fig 6.13: Selecting the type of file and location

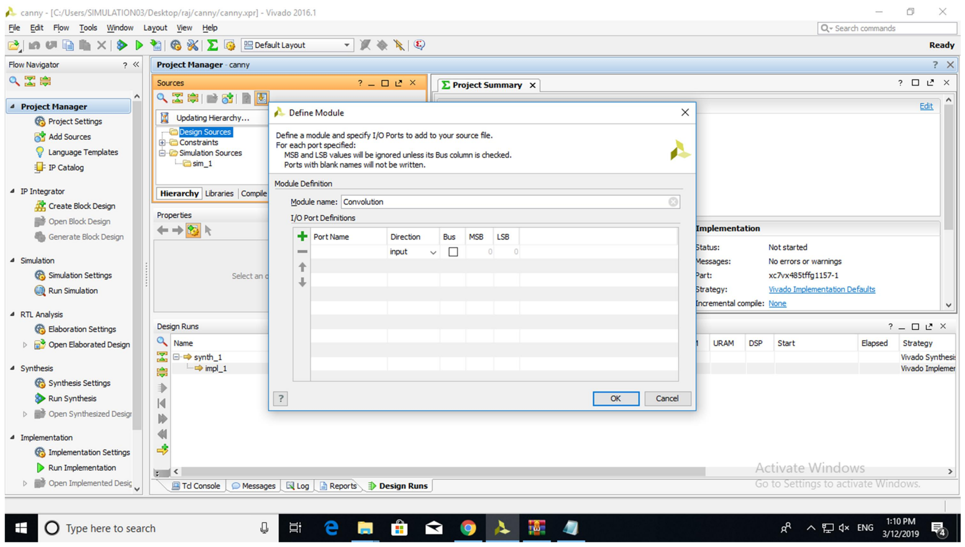
Click the "Finish" button and Vivado will then bring up the "Define Module" window.

Define Module

You can use the "Define Module" window to automatically write some of the VHDL code for you. Additional "I/O Port Definitions" can be added by either clicking the green "+" symbol in the upper left or by simply clicking on the next empty line. The "Entity name" and "Architecture name" will be the corresponding Verilog HDL identifiers used in the code, as will whatever is typed in for each "Port Name". Any valid Verilog HDL identifier can be used for any of these, but for the sake of following along enter the information as shown. Make sure the proper "Direction" is set for each. Click the "OK" button when finished.

Note that if you would rather write your own code from scratch you can simply click the "Cancel" button and Vivado will create a completely blank System Verilog, VHDL source file inside your project. If you click the "OK" button without defining any "I/O Port Definitions" Vivado will still write the basic Verilog HDL code structure but the port definition will be empty and commented out for you to uncomment and fill later.

Also note that the port names here match the silkscreen reference designators of the switches and LEDs on the Basys 3 board that will be utilized for the example. This is for the convenience of those following along with the Basys 3, but should not be inferred as a requirement by beginners; each name is simply an arbitrary identifier.


Define Module

Define a module and specify I/O Ports to add to your source file.
For each port specified:
MSB and LSB values will be ignored unless its Bus column is checked.
Ports with blank names will not be written.

Module Definition

Module name: Convolution

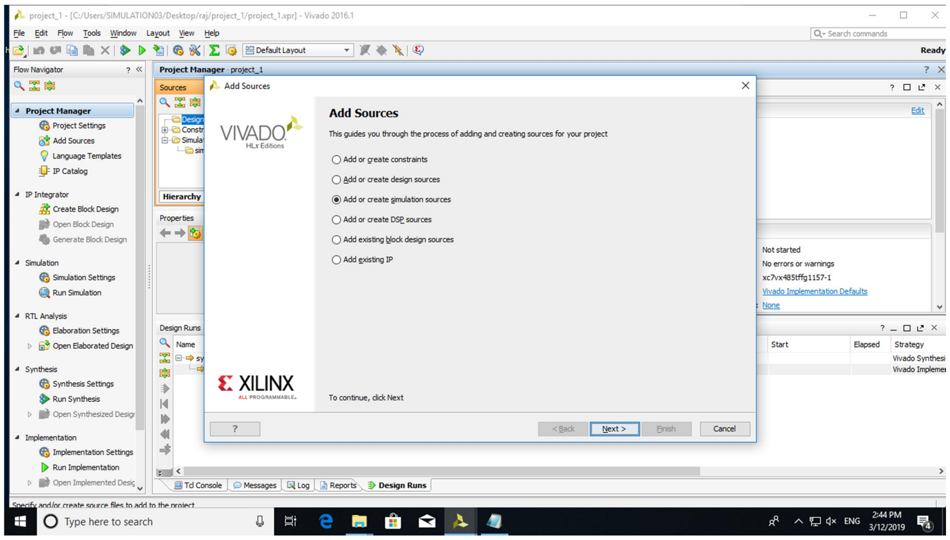
I/O Port Definitions

Port Name	Direction	Bus	MSB	LSB
	input	☐	0	0

Fig 6.14: Module defining with ports

The System Verilog HDL source file generated will be added to your project in the “Design Sources” folder as shown. Double click it and it will open up in a new tab for you to view/edit. All the code here was generated by the previous “Define Module” window, and for this example you only need to manually enter the three highlighted lines between the “begin” and “end” keywords

If we want to create a simulation source we have to select a new simulation source by right clicking the add source block in the panel



Add Sources

This guides you through the process of adding and creating sources for your project

- ( ) Add or create constraints
- ( ) Add or create design sources
- [✓ Add or create simulation sources]
- ( ) Add or create DSP sources
- ( ) Add existing block design sources
- ( ) Add existing IP

To continue, click Next

Not started
No errors or warnings
xc7vx485tffg1157-1
Vivado Implementation Defaults
None

Specify and/or create source files to add to the project

Fig 6.15: Creating the simulation sources

BIBLIOGRAPHY

[1] Mahdavi, M., Edfors, O., Owall, V., & Liu, L. (2019), " A Low Latency FFT/IFFT Architecture for Massive MIMO Systems Utilizing OFDM Guard Bands", IEEE Transactions on Circuits and Systems I: Regular Papers, 1–12.

[2] Enis ÇERRI, Marsida IBRO, "FFT Implementation on FPGA using Butterfly Algorithm", International Journal of Engineering Research & Technology (IJERT), ISSN: 2278-0181, Vol. 4 Issue 02, February-2015.

[3] Arioua.M., Belkouch.S., Agdad, M., & Hassani, M. M. (2011). "VHDL implementation of an optimized 8-point FFT/IFFT processor in pipeline architecture for OFDM systems" International Conference on Multimedia Computing and Systems, 2011.

[4] Ajay.A., & Lourde.R.M.,"VLSI implementation of an improved multiplier for FFT computation in biomedical applications", IEEE International Conference on Electro/Information Technology (EIT) ,2015.

[5] Anant G. Kulkarni1, Dr. M. F. Qureshi2, Dr. Manoj Jha, "Discrete Fourier Transform: Approach to Signal Processing",International Journal of Advanced Research in Electrical, Electronics and Instrumentation Engineering, Vol. 3, Issue 10, October 2014.

[6] Wey.C., Lin.S., & Tang, W. (2007)," Efficient memory-based FFT processors for OFDM applications", In IEEE International Conf. on Electro-Information Technology, 345–350. May.

[7] E.Bidet, C.joanblanq, P.senn," A Fast Single Chip Implementation of 8192 Complex Points FFT",IEEE custom integrated circuits conferece,1994.

[8] Wey.C.L., Tang.W.C., & Lin.S.Y, "Efficient Memory-Based FFT Architectures for Digital Video Broadcasting (DVB-T/H)",International Symposium on VLSI Design, Automation and Test (VLSI-DAT) ,2007.

[9] Anbarasan.A., & Shankar.K, "Design and implementation of low power FFT/IFFT processor for wireless communication". International Conference on Pattern Recognition, Informatics and Medical Engineering (PRIME-2012).

[10] Tang Song-Nien, and Yuan-Ho Chen. 2019. "Area-Efficient FFT Kernel with Improved use of GI for Multi-standard MIMO-OFDM Applications", *Applied Sciences* 9, no. 14: 2877.

[11] F. Qureshi, S.A. Alam and O.Gustafsson,"4k-point FFT Algorithms based on optimized twiddle factor multiplication for FPGA's" in proc.2010 Asia Pacific Conference on Postgraduate Research in Microelectronics and Electronics (Prime Asia),sep.2010.

[12] A.C. Geevarghese, Madheswaran.M," FPGA implementation of IFFT Architecture with enhanced pruning algorithm for low power applications", Science Direct Microprocessors and Micro systems,Volume 71, https://doi.org/10.1016/j.micpro.2019.06.010.

PUBLICATION

[1] B.Somasekhar, S.Srinivas, D.Durga Sai Prasanth, V.Purna Satya Srinivas, K.Anil Kumar, G.Chaitanya,"Implementation of DIT-FFT Algorithm using Verilog", Manuscript submitted to International Journal of Electrical and Electronics Research (IJEER).