DEPARTMENT OF ELECTRONICS AND
COMMUNICATION ENGINEERING NATIONAL
INSTITUTE OF TECHNOLOGY ROURKELA

FPGA Implementation of Fast Fourier Transform Core Using NEDA

ABHISHEK MANKAR

2013

FPGA Implementation of Fast Fourier Transform Core Using NEDA

A THESIS SUBMITTED IN PARTIAL FULFILLMENT
OF THE REQUIREMENTS FOR THE DEGREE OF

Master of Technology

In

VLSI DESIGN AND EMBEDDED SYSTEM

By

Abhishek Mankar

Department of Electronics and Communication Engineering

National Institute Of Technology

Rourkela

2011 – 2013

FPGA Implementation of Fast Fourier Transform Core Using NEDA

A THESIS SUBMITTED IN PARTIAL FULFILLMENT
OF THE REQUIREMENTS FOR THE DEGREE OF

Master of Technology
In
VLSI DESIGN AND EMBEDDED SYSTEM

by

Abhishek Mankar

211EC2085

Under the Guidance of
Prof. Sukadev Meher

Department of Electronics and Communication Engineering

National Institute Of Technology

Rourkela

2011 – 2013

To my parents.

DEPARTMENT OF ELECTRONICS AND
COMMUNICATION ENGINEERING
NATIONAL INSTITUTE OF TECHNOLOGY, ROURKELA
ORISSA, INDIA-769008

CERTIFICATE

This is to certify that the Thesis Report titled "**FPGA Implementation of Fast Fourier Transform Core Using NEDA**", submitted by **Mr. ABHISHEK MANKAR bearing roll no. 211EC2085** in partial fulfilment of the requirements for the award of **Master of Technology in Electronics and Communication Engineering** with specialization in "**VLSI Design and Embedded Systems**" during session 2011 - 2013 at National Institute of Technology, Rourkela is an authentic work carried out by him under my supervision and guidance.

Place: Rourkela
Date: 29TH May, 2013

Prof. (Dr.) SUKADEV MEHER

Acknowledgements

I would like to express my gratitude to my thesis guide **Prof. Sukadev Meher** for his guidance, advice and support throughout my thesis work. I am especially indebted to him for teaching me both research and writing skills, which have been proven beneficial for my current research and future career. Without his endless efforts, knowledge, patience, and answers to my numerous questions, this research would have never been possible. The experimental methods and results presented in this thesis have been influenced by him in one way or the other. It has been a great honour and pleasure for me to do research under supervision of Prof. Sukadev Meher. Working with him has been a great experience. I would like to thank him for being my advisor here at National Institute of Technology, Rourkela.

Next, I want to express my respects to **Prof. Samit Ari, Prof .A.K. Swain, Prof. D.P. Acharya, Prof. P. K. Tiwari, Prof. N. Islam, Prof. K.K Mahapatra, Prof. S. K. Patra, Prof. S. K. Behera , Prof. Poonam Singh** for teaching me and also helping me how to learn. They have been great sources of inspiration to me and I thank them from the bottom of my heart.

I would like to thank to all my faculty members and staff of the Department of Electronics and Communication Engineering, N.I.T. Rourkela, for their generous help for the completion of this thesis.

I would like to thank all my friends and especially my classmates for thoughtful and mind stimulating discussions we had, which prompted to think beyond the obvious. I’ve enjoyed their companionship so much during my stay at NIT, Rourkela.

I am especially indebted to my parents for their love, sacrifice, and support. My parents are my first teachers, after I came to this world and I have set of great examples for me about how to live, study and work. I am grateful to them for guiding my steps on the path of achievements since my infanthood.

Abhishek Mankar

CONTENTS

Chapter 1 Introduction

Chapter 2 Fast Fourier Transform

Chapter 3 NEW Distributed Arithmetic Technique

Chapter 4 Discrete Fourier Transform using NEDA

Chapter 5 16 – point Radix – 4 Complex FFT

Chapter 6 Radix – 2^2 folded pipelined complex FFT

Chapter 7 Conclusions

Abstract

Transforms like Discrete Fourier Transform (DFT) are a major block in communication systems such as OFDM, etc. This thesis reports architecture of a DFT core using new distributed arithmetic (NEDA) algorithm. The advantage of the proposed architecture is that the entire transform can be implemented using adder/subtractors and shifters only, thus minimising the hardware requirement compared to other architectures. The proposed design is implemented for 16 – bit data path (12 – bit for comparison) considering both integer representation as well as fixed point representation, thus increasing the scope of usage. The proposed design is mapped on to Xilinx XC2VP30-7FF896 FPGA, which is fabricated using 130 nm process technologies. The maximum on board frequency of operation of the proposed design is 122 MHz.

NEDA is one of the techniques to implement many digital signal processing systems that require multiply and accumulate units. FFT is one of the most employed blocks in many communication and signal processing systems. The FPGA implementation of a 16 point radix-4 complex FFT is proposed. The proposed design has improvement in terms of hardware utilization compared to traditional methods. The design has been implemented on a range of FPGAs to compare the performance. The maximum frequency achieved is 114.27 MHz on XC5VLX330-2FF1760 FPGA and the maximum throughput observed is 1828.32 Mbit/s and minimum slice delay product observed is 9.18. The design is also implemented using synopsys DC synthesis for both 65 nm and 180 nm technology libraries.

The advantages of multiplier-less architectures are reduced hardware and improved latency. The multiplier-less architectures for the implementation of radix-2^2 folded pipelined complex FFT core are based on NEDA. The number of points considered in the work is sixteen and the folding is done by a factor of four. The proposed designs are implemented on Xilinx XC5VSX240T-2FF1738 FPGA. Proposed designs based on NEDA have reduced area over 83% The observed slice-delay product for NEDA based designs are 2.196 and 5.735.

LIST OF FIGURES

LIST OF TABLES

ACRONYMS

DA	Distributed Arithmetic
DFT	Discrete Fourier Transform
FPGA	Field Programmable Gate Array
FFT	Fast Fourier Transform
NEDA	NEW Distributed Arithmetic

Chapter -1

Introduction

Introduction

1.1 Motivation

If an N – point DFT is implemented directly, the requirement of arithmetic operations is of the order of $O(N^2)$ that is N^2 multiplications and N (N-1) additions. The order of arithmetic operations for FFT is O(NlogN) that is NlogN additions and (N/2)logN multiplications. Thus FFT implementation of DFT is preferred. Depending on inputs being real or complex, the structures of adders and multipliers are built. Distributed arithmetic (DA) [1] has become one of the most efficient tools in VLSI implementation of digital signal processing (DSP) architectures. It efficiently computes inner products of vectors, which is a key requirement in many DSP systems. One of the key computational blocks in DSP is multiply/accumulate (MAC), which is implemented by a standard adder unit and a multiplier. Using DA, MAC unit can be implemented by pre computing all possible products and using a ROM to store them. The con of using DA is in its exponential increase of the size of ROM with increase in internal precision and number of inputs. An approach to overcome this drawback is by distributing the coefficients to inputs. One of such examples is new distributed arithmetic (NEDA) [2]. As in [2], it can be used to implement any transform that is based on fourier basis. This approach helps in finding out the redundancy in computing vector inner product thus reducing the number of computational blocks, especially adders of a multiplier thus making it more hardware complex.

Many architectures for DFT, based on DA, is implemented in [3] – [5]. Discrete Hartley Transform (DHT), considered as sister transform of DFT, also was implemented using DA [6]. The disadvantage of all these implementations is that they use ROM or RAM which makes the designs with increased architecture. Other approach in implementing DFT is based on employing CORDIC units [7] – [8]. Even this has cons as the structure of a CORDIC unit is comparable to that

Techniques in designing DSP systems such as folding, pipelining have always improved performance of the systems in terms of area of hardware, latency, frequency, etc. To determine the control circuits systematically in DSP architectures, the folding transformation is used. In folding, time multiplexing of multiple algorithm operations to a single functional unit is done. Thus, in any DSP architecture, folding provides a means for trading time for area. In general, folding can be used to reduce the number of hardware functional units by a factor of N at the expense of increasing the computation time by a factor of N [4]. To avoid excess amount of registers used in these architectures that occur while folding, there are techniques that compute the minimum number of registers needed to implement a folded DSP architecture. These techniques also help in allocation of data in these registers. Pipelining transformation in DSP architectures reduces critical path, which can be pro-sequenced to either decrease the power consumption or to enhance the sample frequency or clock speed. By introducing the pipelining latches along the data-path, critical path in particular, pipelining technique reduces the effective critical path.

1.2 Problem Description

Fast Fourier Transform (FFT) is used to build various image processing systems and application specific Digital Signal Processing (DSP) hardware. Currently almost all proposed designs for FFT use ROMs or memory for complex twiddle multiplications. Proper techniques must be followed to eliminate the need of multipliers in FFT design. One of the most frequently used and significant method to eliminate the multipliers used in FFT design is using New Distributed Arithmetic (NEDA) for twiddle multiplications. While using NEDA technique, one must do precise shifting to reduce the number of adders.

While implementing FFT cores for long data sequence on FPGA, the number of bonded inputs and outputs (IOBs) are always a matter of concern. Precise techniques must be used to lessen the number of IOBs. We can reduce the number of IOBs if we can divide the long data sequence into a group of short data sequences of same length. By this not only throughput will be increased but also the number of IOBs will be reduced by a great factor. This can be achieved by folding technique. But by using folding technique, the computational time will be increased in a linear manner as well as the latency. To avoid excess amount of registers that are generated in these architectures while folding, there are techniques to minimise the number of registers needed to implement DSP architectures through folding.

Various proposed FFT algorithms have used pipelining in order to increase throughput and speed. This current work has used both folding and pipelining so that speed and throughput can be increased while not effecting the computational time. In a normal parallel design, the outputs starts coming after a very few clock cycles. The number of clock cycles to get the output increases as we incorporate pipelining and folding technique. Again while using pipelining and folding technique, there may be clock mismatch giving rise to undefined outputs in some clock cycles and wrong outputs in the other clock cycles. So, very strict attention must be paid while using folding and pipelining technique.

NEDA has been used in this work to design various FFT cores. Higher radix FFT algorithms also have been used to design the various FFT cores. Radix-4, radix-2^2 butterfly sections has been optimized for better performance. The proposed designs in the present work are efficient in terms of area, speed and power. The present designs have been tested on various FPGAs. ASIC implementation of the proposed architectures in 0.18μm process technology using Synopsys DC for logic synthesis and Cadence SoC Encounter for physical design has also been conducted. The physical design of proposed architectures has been made in such a way that the timing constraints are met after both placement as well as routing.

1.2 Organisation of this Thesis

This thesis has been divided into several chapters. A brief overview of the problem targeted in the current work has been presented in this chapter. The second chapter describes the various FFT algorithms used to design the FFT cores. The third chapter presents the overview of NEDA technique. The four chapter presents the Discrete Fourier Transform. The five chapter presents the 16-Point Radix-4 Complex FFT. Chapter six presents three proposed architectures for different FFT cores.

Chapter 2

Fast Fourier Transform

2.1 INTRODUCTION:

Today’s electronic systems mostly run on batteries thus making the designs to be hardware efficient and power efficient. Application areas such as digital signal processing, communications, etc. employ digital systems which carryout complex functionalities. Hardware efficient and power efficient architectures for these systems are most required to achieve maximum performance.

Fast Fourier Transform (FFT) is one of the most efficient ways to implement Discrete Fourier Transform (DFT) due to its reduced usage of arithmetic units. DFT is one of those primary tools that are used for the frequency analysis of discrete time signals and to represent a discrete time sequence in frequency domain using its spectrum samples. The analysis (forward) and synthesis (inverse) equations of an N point FFT are given below.

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

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

(2.1.1)

Where, $W_N^{kn} = e^{-j\frac{2\pi}{N}kn}$. As evident from equation (2.1.1), the basis of both synthesis and analysis equations remains same thus increasing the scope of the architecture to both analyze and synthesize. Due to increased employability of FFT in modern electronic systems, higher radix FFTs such as radix – 4, radix – 8, radix – 2^k, split radix, etc. are designed for improved timing and reduced hardware. The basic difference of the mentioned methods lies in the structure of their butterfly units.

2.2 The Discrete Fourier Transform (DFT):

Transforms such as discrete fourier transform (DFT) are a major block in many communication systems like OFDM, etc. DFT is also considered as one of the major tools to perform frequency analysis of discrete time signals. A discrete time sequence can be represented by samples of its spectrum in the frequency domain, using DFT.

The Discrete Fourier Transform is a continuous Fourier transform for the case of discrete functions. Given a real sequence of {x_n}, the DFT expresses them as a sequence {X_k} of complex numbers, The mathematical representation of the transform is:

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

(2.2)

Using more simple notation of equation (2.2) is,

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

(2.2.1)

Introducing the term

$$W_N = e^{\frac{-j2\pi}{N}} \tag{2.2.2}$$

Expanding the equ (2.2.2) we get the real and imaginary form.

$$W_N = \cos\frac{2\pi}{N} - j\,\sin\frac{2\pi}{N} \tag{2.2.3}$$

which is an Nth root of unity.

Finally the division of the core into real and imaginary coefficients is based on the following change made to equation (2.2.1).

$$X(k) = \sum_{n=0}^{N-1} x(n) \cos\,(\frac{2\pi kn}{N}) - j \sum_{n=0}^{N-1} x(n) \sin\left(\frac{2\pi kn}{N}\right)$$

(2.2.4)

The inverse of DFT is given as below.

$$x(n) = \left(\frac{1}{N}\right) \sum_{k=0}^{N-1} X(k) e^{\left(\frac{j2\pi kn}{N}\right)}, \mathrm{k} = 0, 1 \ldots \mathrm{N} - 1$$

(2.2.5)

Using compact notation of equ (2.2.5)

$$x(n) = \left(\frac{1}{N}\right) \sum_{k=0}^{N-1} X(k) W_N^{-nk}, \quad \mathrm{k} = 0, 1 \ldots \mathrm{N} - 1$$

(2.2.6)

As seen from equations (2.2) and (2.2.5), the forward and inverse transforms can be implemented using same kernel with few modifications, thus reducing the requirement of hardware. Many efficient ways have been put up for direct implementation of DFT due to its computational complexity. Fast fourier transform (FFT) is one of the most efficient and common ways to implement DFT. Reduced computational complexity and low latency are two driving factors for implementing DFT using FFT. Other FFT architectures that have been developed are based on radix – 4, radix – 2, hybrid, split radix.

If an N – point DFT is implemented directly, the requirement of arithmetic operations is of the order of O (N^2) that is N^2 multiplications and N (N-1) additions. The order of arithmetic operations for FFT is O (NlogN) that is NlogN additions and (N/2)logN multiplications. Thus FFT implementation of DFT is preferred. Depending on inputs being real or complex, the structures of adders and multipliers are built.

2.3 Divide – and – Conquer Approach to Computation of the DFT:

The development of computationally efficiently algorithm for the DFT is made possible to adopt a divide and conquer approach. This approach has decomposed an N point DFT into smaller DFT. This process is basically more efficient algorithm known collectively as FFT algorithm.

Let us consider N-point DFT , where N can be a factored as a product of two integer, that is,

$$N = LM \qquad (2.3.1)$$

Assumption that N is not a prime number is not restrictive, so we can pad any sequence with zeros to ensure a factorization of the form (2.3.1).

Now x(n) is mapped into the rectangular array x(l,m) and X(k) is mapped into corresponding rectangular array X(p,q) .DFT can be expressed as a double sum over the elements of the rectangular array multiplied by corresponding phase factor.

Then

$$X(p,q) = \sum_{m=0}^{M-1} \sum_{l=0}^{L-1} x(l,m) W \, \mathrm{N}(\mathrm{Mp}+\mathrm{q})\,(\mathrm{mL}+\mathrm{l}) \qquad (2.3.2)$$

But

$$W_N^{(Mp+q)(mL+l)} = W_N^{MLmp} \, W_N^{mLq} \, W_N^{Mpl} \, W_N^{lq} \qquad (2.3.3)$$

However , $W_N^{Nmp} = 1$, $W_N^{mqL} = W_{N/L}^{mq} = W_M^{mq}$, and , $W_N^{Mpl} = W_{N/M}^{pl} = WLpl$.

Simplified the equation (2.3.2) can be expressed as:

$$X(p,q) = \sum_{m=0}^{M-1}\{ W_N^{lq} [\sum_{m=0}^{M-1} x(l,m) W_M^{mq}]\}\ W_L^{lp} \qquad (2.3.4)$$

The expression of (2.3.4) involves the computation of DFT of length M and length L.

2.3.1 Radix – 4 FFT algorithms:

When the number of data points N in the DFT is a power of 4 (i.e; N = 4^v) **,** and it is more efficient to employ a radix -4 algorithm.

Now describe the radix -4 decimation – in – time FFT algorithm, now we selecting L= 4 and M= N/4 in the divide – and – conquer approach describe in section (2.2). now for this region the choice of L and M. we have l , p = 0,1,2,3; m , q= 0,1,2,….N/4 – 1;n = 4m + l; and k= (N/4)p + q. the decimate the N point input sequence into four sub-sequences, x(4n), x(4n +1) ,x(4n + 2) and x(4n + 3), n= 0,1,……N/4.

By applying (2.3.4) we get

$$X(p,q) = \sum_{l=0}^{3} [W_N^{lq}\ F(l,q)]\ W_4^{lp}; \qquad p = 0,1,2,3 \qquad (2.3.1.1)$$

Now ,

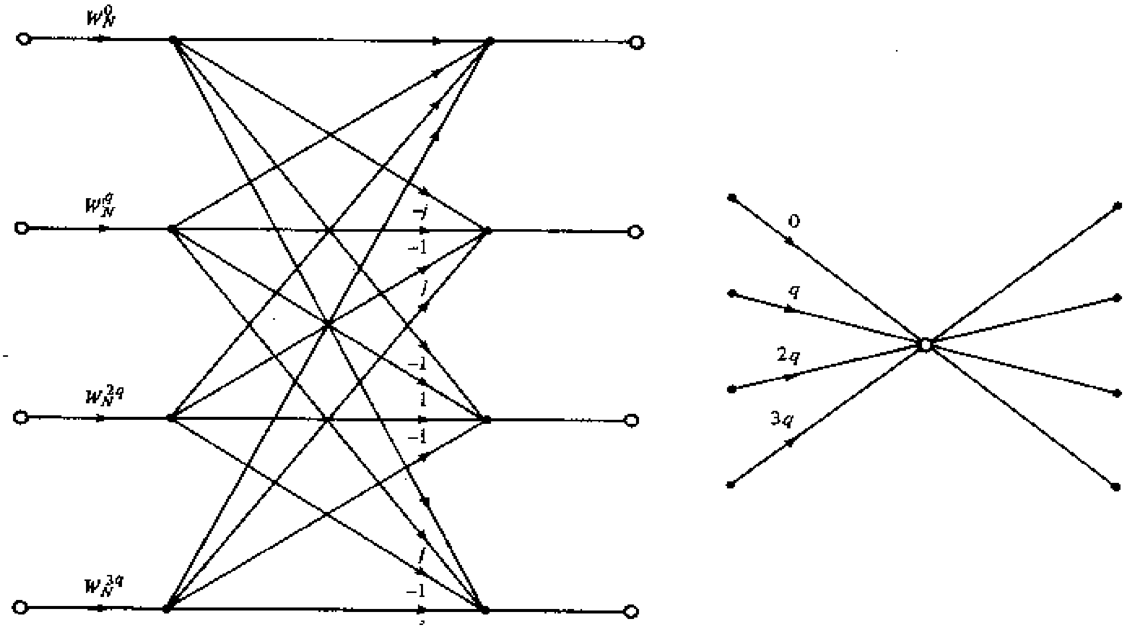
$$F(l,q) = \sum_{m=0}^{\left(\frac{N}{4}\right)-1} x(l,m) W_{N/4}^{mq}, \qquad l= 0\ , 1, 2\ ,3 \text{ and } q =0,1,2,\ldots,N/4 - 1 \qquad (2.3.1.2)$$

$$x(l,m) = x(4m + l) \qquad (2.3.1.3)$$

$$X(p,q) = X(N/4p + q) \qquad (2.3.1.4)$$

The four N/4 – point DFT’s obtained from (2.3.1.2) are combined according to (2.3.1.1) to yield the N – point DFT. The expression in (2.3.1.1) for combining the N/4 point DFT define a radix -4 decimation – in –time butterfly. The radix – 4 butterfly is in figure 2.1 (a) and more compact form in figure 2.1 (b). Note that $W_N^0 = 1$, each butterfly involves three complex multiplication and 12 complex addition.

The decimation – in time procedure can be can be repeated recursively v times. Hence the FFT algorithm consist of v stage , where each stage contain N/4 butterflies. The algorihm is



(a) (b)

Figure 2.1 Basic butterfly computation in a radix – 4 FFT algorithm

3vN/4 = (3N/8)log$_2$N complex multiplication and (3N/2)log$_2$N complex addition.

Now finally the multiplication reduced by 25% but the number of additions has increased by 50% from Nlog$_2$N to (3N/2)log$_2$N.

Now a radix – 4 decimation – in – frequency FFT algorithm can be obtained by selecting L = N/4 , M = 4; l,p = 0,1,….N/4-1;m , q = 0,1,2,3; q = 0,1,2,3; n = (N/4)m + l ;and k = 4p + q. now take these value and put in equation (2.2.4) ,

$$X(p,q) = \sum_{l=0}^{\left(\frac{N}{4}\right)-1} G(l,q) W_{N/4}^{lp} \tag{2.3.1.5}$$

Where

$$G(l,q) = W_N^{lq}\,F(l,q)\,, \qquad q = 0,1,2,3 \text{ and } l = 0,1,2,3\ldots\ldots N/4 - 1. \qquad (2.3.1.6)$$

And

$$F(l,\,q) = \sum_{m=0}^{3} x(l,m)\,W_4^{mq}\,, \qquad q = 0,1,2,3 \text{ and } l = 0,1,2,3\ldots\ldots N/4 - 1\,. \qquad (2.3.1.7)$$

Note that X(p,q) = X(4p + q) , q = 0,1,2,3. The N – point DFT is decimated into four N/4 point DFT and we have a decimation – in – frequency algorithm. The computation equation (2.3.1.6) and (2.3.1.7) define the basic radix – 4 butterfly for the decimation – in – frequency algorithm.

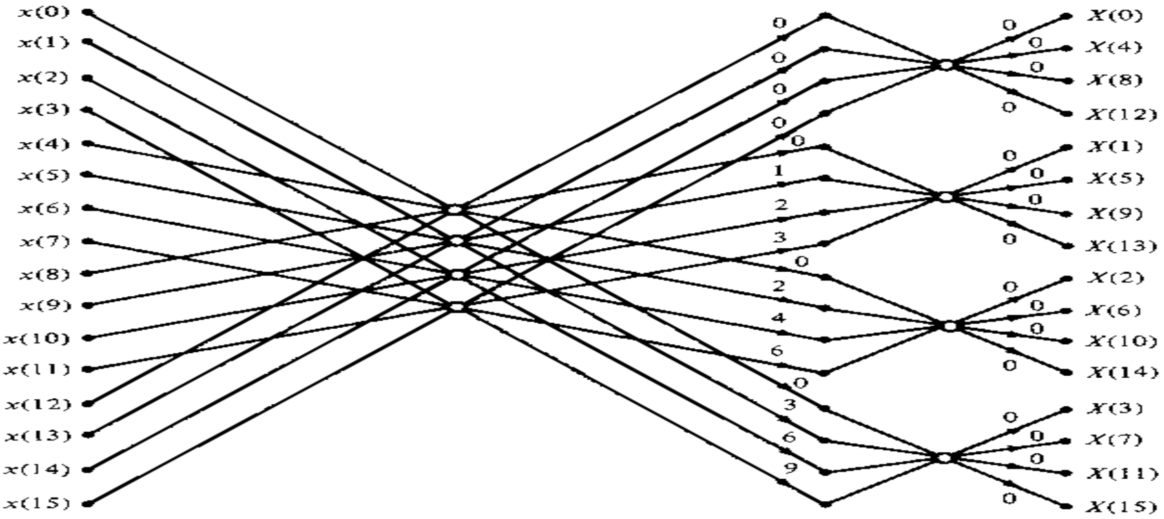


Figure 2.2 Sixteen- point, radix – 4 decimation – in – frequency algorithm with input in normal order and output in digit – reversed order.

A 16 point radix – 4 decimation – in – frequency FFT algorithm shown in figure (2.2). the inputs is in normal order but the outputs comes in digit reversed order. It has exactly same as computational complexity as the decimation – in frequency algorithm.

2.4 Radix-2^2 FFT:

The radix-2^2 FFT algorithm is well known for its simple structure as that of radix-2 FFT algorithm and computational complexity as that of radix-4 FFT algorithm. Figure 2.4 shows the data flow graph of radix-

2^2 16-point DIF FFT core using NEDA blocks. The generalized expression for the radix-2^2 FFT algorithm is given below.

The butterfly diagram for the radix-2^2 FFT algorithm is shown in figure 2.

$$
\begin{aligned}
&X(4k_1 + 2l_1 + l_2) \\
&\quad = \sum_{n=0}^{\frac{N}{4}-1} \left[x(n) + x\left(n + \frac{N}{4}\right) W_4^{2l_2+l_1} + x\left(n + \frac{N}{2}\right) W_4^{4l_2+2l_1} \right. \\
&\quad \left. + x\left(n + \frac{3N}{4}\right) W_4^{6l_2+3l_1} \right] W_N^{n(2l_2+l_1)} W_{\frac{N}{4}}^{nk_1} \\
&\quad = \sum_{n=0}^{\frac{N}{4}-1} \left[[x(n) + (-1)^{l_1} x\left(n + \frac{N}{2}\right)] + (-1)^{l_2}(-j)^{l_1}[x\left(n + \frac{N}{4}\right) \right. \\
&\quad \left. + (-1)^{l_1} x\left(n + \frac{3N}{4}\right)] W_4^{6l_2+3l_1}] \right] W_N^{n(2l_2+l_1)} W_{\frac{N}{4}}^{nk_1}
\end{aligned}
\tag{2.4.1}
$$

where

l_1, l_2=0, 1 and k=0, 1,....., N/4-1

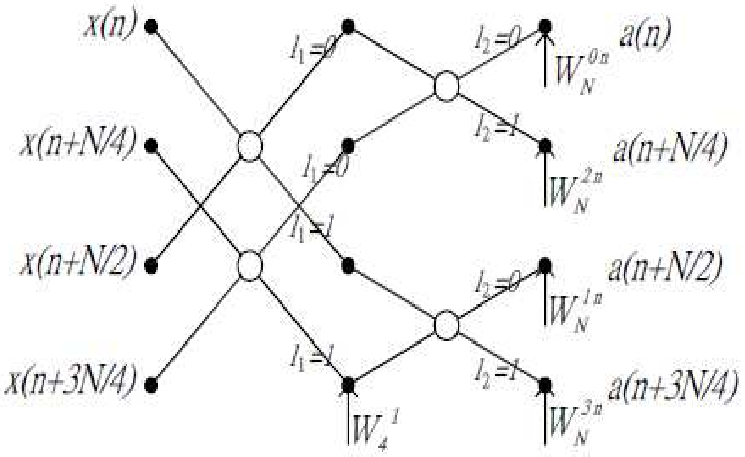


Figure 2.3. Butterfly diagram of radix-2^2 FFT algorithm

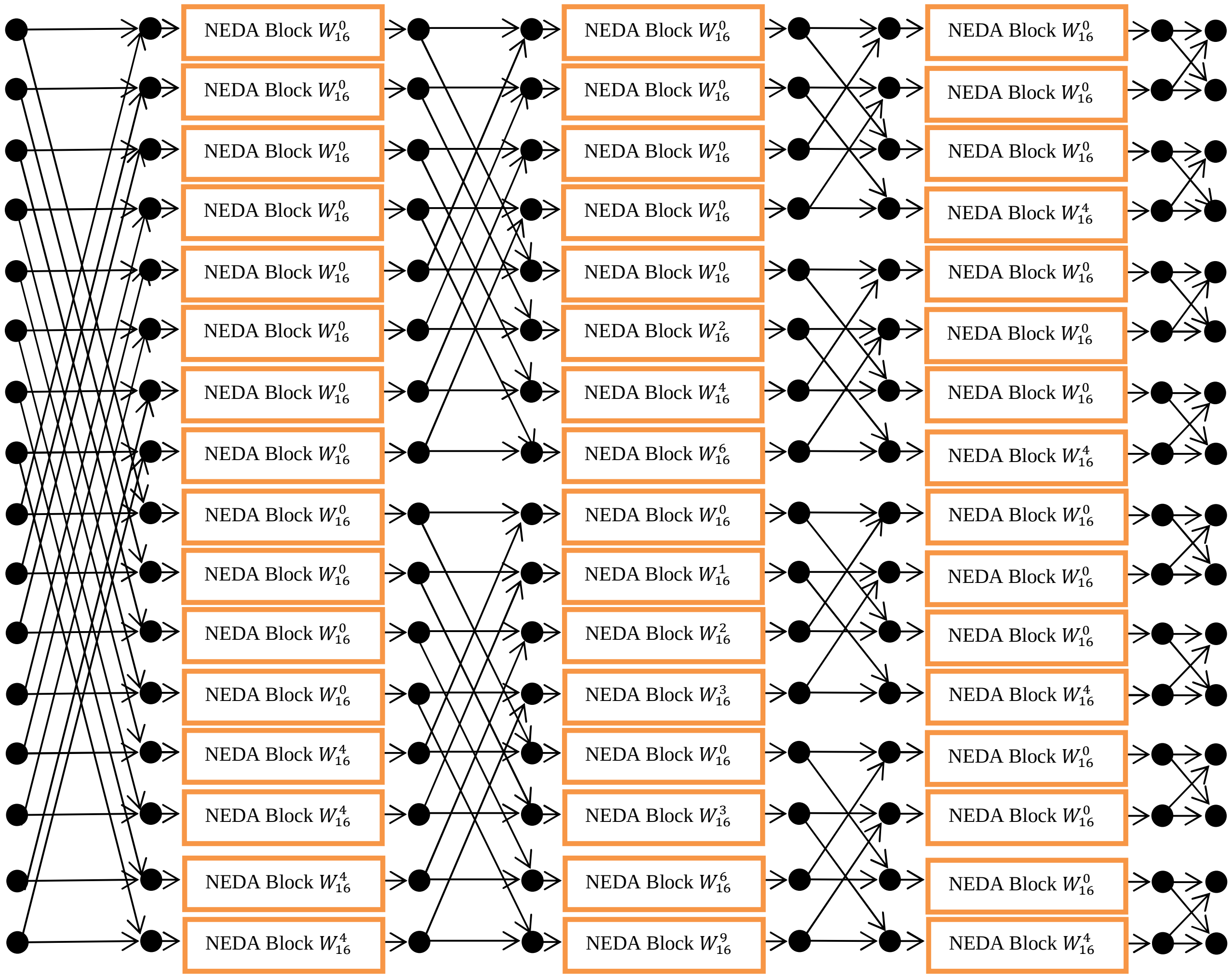


Figure 2.4. Data flow graph of 16-point radix-2^2 DIF FFT using NEDA (Left most column of dark bubbles denote inputs from 0 to 15 in normal order and right most column of dark bubbles denote outputs from 0 to 15 in bit reversal order).

Chapter 3

NEW Distributed Arithmetic Technique

3.1 Distributed Arithmetic (DA):

Distributed Arithmetic (DA) was invented about 30 years ago and has since seen widespread applications in area of VLSI implementation of DSP algorithms . DA has become one of the most efficient tools in implementation of multiply and accumulate (MAC) unit in several DSP systems. Most of the applications, for example discrete cosine transform (DCT), discrete wavelet transform (DWT) calculation, are commonly implemented using DA based approach as they all are hardware intensive with multipliers and MAC units. MAC unit is implemented using DA by pre computing all possible products and then storing them in a read only memory (ROM). In simple words, DA computes the inner product of two multi-dimensional vectors. Thus, increase in the number of dimensions increases the memory requirement to store all the obtained products. This is due to the reason that, increase in number of dimensions increases the number of obtained partial products. The elimination increased memory requirement is possible only if one or both of the inputs has a fixed set of coefficients.

DA inner-product generation, consider the calculation of the following sum of products,

$$Y = \sum_{i=1}^{k} C_i X_i \tag{3.1}$$

Where C_i is the fixed coefficients value and X_i is input data words.

If each X_i is a 2's-complement binary number scaled(for convenience, not as necessity) such that $|X_i| < 1$, then we may express each X_i as

$$X_i = - a_{k0} + \sum_{n-1}^{N-1} a_{kn} 2^{-n} \tag{3.2}$$

where the a_{kn} are the bits, 0 or 1, a_{k0} is the sign bit, and a_k.N-1 is the least significant bit (LSB). Now let us combine Equations 3.1 and 3.2 in order to express Y in terms of the bits of X_i :

$$Y = \sum_{i=1}^{k} C_i [- a_{k0} + \sum_{n-1}^{N-1} a_{kn} 2^{-n}] \tag{3.3}$$

Equation (3.3) is the conventional form of expressing the inner product. Direct mechanization of this equation defines a "lumped" arithmetic computation. Let us instead interchange the order of the summations, which gives us:

$$Y=\sum_{n-1}^{N-1}[\sum_{i=1}^{k} C_i a_{kn}] 2^{-n} + \sum_{i=1}^{k} C_I (- a_{k0}) \tag{3.4}$$

This is the crucial step: Equation (3.4) defines a distributed arithmetic computation. Consider the bracketed term in Equation (3.4):

$$\sum_{i=1}^{k} C_i \, a_{kn} \tag{3.5}$$

Because each $\mathbf{a}_{kn}$ may take on values of 0 and 1 only, expression (3.5) may have only 2K possible values. Rather than compute these values on line, we may pre compute the values and store them in a ROM. The input data can be used to directly address the memory and the result, i.e., the $\sum_{i=1}^{k} C_i \, a_{kn}$, can be dropped into an accumulator. After N such cycles, the memory contains the result, y.

3.2 Why NEDA:

Distributed arithmetic technique is used to implement the sum of product terms and in this technique we used ROM, Adder and Shifter for implementing purpose, but in NEDA technique we used only Adder and Shifter and eliminated the ROM.

So, the size of the architecture is reduced with respect to Distributed arithmetic technique, and also enhanced the speed and throughput of the architecture.

3.3 Overview of NEW Distributed Arithmetic Technique:

NEw Distributed Arithmetic (NEDA) is being used in many digital signal processing systems that possess MAC units as their computational blocks. Transforms such as DWT, DCT, etc. have many MAC like structures that in turn require much hardware. Implementation of such transforms using multiplier-less and ROM-less architectures using blocks like NEDA improves performance of the system in terms of power, area and speed. The mathematical deduction of NEDA is discussed below.

Inner product computation of any two sequences can be represented using summation as

$$Z = \sum_{i=1}^{k} M_i \, X_i$$

(3.3.1)

Where the inputs M_i are constant fixed coefficients and the inputs X_i are varying with time as a function. The matrix product representation of equation (3.3.1) may be given as

$$Z = [M_1 \quad M_2 \quad \cdots \quad M_k] \begin{bmatrix} X_1 \\ X_2 \\ \vdots \\ X_k \end{bmatrix}$$

(3.3.2)

Considering both M_i and X_i in signed 2's complement number system, they can be expressed in the form as shown below

$$M_i = -M_i^m 2^m + \sum_{k=n}^{m-1} M_i^k \, 2^k$$

(3.3.3)

Where $M_i = 0 \; or \; 1$, $k = n, n+1, \ldots, m$and M_i^m is the sign bit, as it is the most significant, and M_i^n is the least significant bit. Substituting equation (3.3.3) in equation (3.3.2) results in the following matrix product which is modelled according to the required design depending on the values of the fixed inputs.

$$Z = [-2^0 \quad 2^{-1} \quad \cdots \quad 2^{-12}] \begin{bmatrix} M_1^0 & \cdots & M_k^0 \\ \vdots & \ddots & \vdots \\ M_1^{12} & \cdots & M_k^{12} \end{bmatrix} \begin{bmatrix} X_1 \\ X_2 \\ \vdots \\ X_k \end{bmatrix}$$

(3.3.4)

The matrix containing M_i^k is a sparse matrix, which means that the values are either 1 or 0. The number of rows in M matrix defines the precision of fixed coefficients used. Equation (3.3.4) is rearranged as shown below.

$$Z = \begin{bmatrix} -2^{0} & 2^{-1} & \cdots & 2^{-12} \end{bmatrix} \begin{bmatrix} W_0 \\ W_1 \\ \vdots \\ W_{12} \end{bmatrix}$$

(3.3.5)

Where $$\begin{bmatrix} W_0 \\ W_1 \\ \vdots \\ W_{12} \end{bmatrix} = \begin{bmatrix} M_1^0 & \cdots & M_k^0 \\ \vdots & \ddots & \vdots \\ M_1^{12} & \cdots & M_k^{12} \end{bmatrix} \begin{bmatrix} X_1 \\ X_2 \\ \vdots \\ X_k \end{bmatrix} \quad (3.3.6)$$

In each row, the W matrix consists of sums, related to the partial products, of the inputs depending on the coefficient values. An example that shows the NEDA operations is elucidated below. Consider to evaluate the value of equation (3.3.7).

$$Y = \begin{bmatrix} \cos\frac{\pi}{8} & \cos\frac{\pi}{4} \end{bmatrix} \begin{bmatrix} X_1 \\ X_2 \end{bmatrix} \quad (3.3.7)$$

Equation (3.3.7) can be expressed in the form of equation (3.3.4) as a matrix product has been shown in equation (3.3.8).

Thus implementing equation (3.3.10) further reduces number of adders compared to implement equation (3.3.9). Multiplication with $2^{-i}, i \in Z^{+}$ can be realized with the help of arithmetic shifters. In equation (3.3.10), the first row of X matrix shifts right by 1 bit, second row by 2 bits and so on. More precisely, the shifts carried out are arithmetic right shifts. The output Y can be realized as a column matrix when we need the partial products. Thus, NEDA based architecture designs have less critical path compared to traditional MAC units without multipliers as well as memory

$$Y = \begin{bmatrix} -2^{0} & 2^{-1} & \cdots & 2^{-12} \end{bmatrix} \begin{bmatrix} 0 & 0 \\ 1 & 1 \\ 1 & 0 \\ 1 & 1 \\ 0 & 1 \\ 1 & 0 \\ 1 & 1 \\ 0 & 0 \\ 0 & 1 \\ 1 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} X_1 \\ X_2 \end{bmatrix} \tag{3.3.8}$$

Equation (3.3.8) may be rewritten as

$$Y = \begin{bmatrix} -2^{0} & 2^{-1} & \cdots & 2^{-12} \end{bmatrix} \begin{bmatrix} 0 \\ X_1 + X_2 \\ X_1 \\ X_1 + X_2 \\ X_2 \\ X_1 \\ X_1 + X_2 \\ 0 \\ X_2 \\ X_1 \\ 0 \\ 0 \\ 0 \end{bmatrix} \tag{3.3.9}$$

Applying precise shifting, we rewrite equation (3.3.9) as

$$Y = \begin{bmatrix} 2^{-1} & 2^{-2} & 2^{-3} & 2^{-4} & 2^{-5} & 2^{-6} & 2^{-8} & 2^{-9} \end{bmatrix} \begin{bmatrix} X_1 + X_2 \\ X_1 \\ X_1 + X_2 \\ X_2 \\ X_1 \\ X_1 + X_2 \\ X_2 \\ X_1 \end{bmatrix} \tag{3.3.10}$$

.

Chapter 4

Discrete Fourier Transform

4.1 Introduction:

In chapter 2 we completely discussed about DFT, which has been reintroduced here for convenience. Transforms such as discrete fourier transform (DFT) are a major block in many communication systems like OFDM, etc. DFT is also considered as one of the major tools to perform frequency analysis of discrete time signals. A discrete time sequence can be represented by samples of its spectrum in the frequency domain, using DFT. The mathematical representation of the transform is shown below.

$$X(k) = \sum_{n=0}^{N-1} x(n) e^{(-\frac{j2\pi kn}{N})} \quad \text{n=0,1...N-1} \tag{4.1}$$

The inverse of DFT is given as below.

$$x(n) = (1/N) \sum_{k=0}^{N-1} X(k) e^{(\frac{j2\pi kn}{N})} \quad k = 0, 1 \ldots N-1 \tag{4.2}$$

As seen from equations (4.1) and (4.2), the forward and inverse transforms can be implemented using same kernel with few modifications, thus reducing the requirement of hardware. Many efficient ways have been put up for direct implementation of DFT due to its computational complexity. Fast fourier transform (FFT) is one of the most efficient and common ways to implement DFT.

If an N – point DFT is implemented directly, the requirement of arithmetic operations is of the order of O (N^2) that is N^2 multiplications and N (N-1) additions. The order of arithmetic operations for FFT is O (NlogN) that is NlogN additions and (N/2)logN multiplications. Thus FFT implementation of DFT is preferred. Depending on inputs being real or complex, the structures of adders and multipliers are built.

4.2 8- point Discrete Fourier Transform (DFT):

4.2 (a) 8 – point Real Inputs

The design of proposed DFT core is divided into two stages. First stage divides the computation based on real and imaginary coefficients, carries out the computation based on NEDA. Second stage integrates the two sub modules to make the design of the core complete.

The division of the core into real and imaginary coefficients is based on the following change made to equation (4.1).

$$X(k) = \sum_{n=0}^{7} x(n)\cos\left(\frac{2\pi kn}{N}\right) - j\sum_{n=0}^{7} x(n)\sin\left(\frac{2\pi kn}{N}\right) \qquad (4.3)$$

Equation (4.3) is re written as follows.

$$X(k) = X_C(k) - jX_S(k) \qquad (4.4)$$

where,

$$X_C(k) = \sum_{n=0}^{7} x(n)\cos\left(\frac{2\pi kn}{N}\right)$$

$$X_S(k) = \sum_{n=0}^{7} x(n)\sin\left(\frac{2\pi kn}{N}\right)$$

(4.5)

The sub modules are designed by exploring symmetry property of the DFT thus reducing number of computational blocks. Now the proposed architecture designed for 8 point DFT. For this reason equation (4.3) was represented for 8 point DFT.

The proposed architecture considers both real inputs and complex inputs. So we first explain the real inputs. In real inputs case we use symmetry property technique but in complex inputs we can’t use the symmetry property. For this reason it increases the count of slices, LUTs and area. But it is more preferred due to its high speed and low power.

First, we explained the method to find the outputs from the equation (4.3) using real inputs;

$$X(0) = x(0) + x(1) + x(2) + x(3) + x(4) + x(5) + x(6) + x(7) \tag{4.6}$$

$$X(1) = x(0) + x(1)\, e^{\frac{-j2\pi}{8}} + x(2)\, e^{\frac{-j4\pi}{8}} + x(3)\, e^{\frac{-j6\pi}{8}} + x(4)\, e^{\frac{-j8\pi}{8}} + x(5)\, e^{\frac{-j10\pi}{8}} + x(6)\, e^{\frac{-j12\pi}{8}} + x(7)\, e^{\frac{-j14\pi}{8}} \tag{4.7}$$

Now, expanding the equation (4.7) into real and imaginary form, results in the complex outputs.

$$X(1) = x(0) + x(1)\cos\frac{\pi}{4} - j\, x(1)\sin\frac{\pi}{4} + x(2)\cos\frac{\pi}{2} - j\, x(2)\sin\frac{\pi}{2} + x(3)\cos\frac{3\pi}{4} - j\, x(3)\sin\frac{3\pi}{4} + x(4)\cos\pi - j\, x(4)\sin\pi + x(5)\cos\frac{5\pi}{4} - j\, x(5)\sin\frac{5\pi}{4} + x(6)\cos\frac{3\pi}{2} - j\, x(6)\sin\frac{3\pi}{2} + x(7)\cos\frac{7\pi}{4} - j\, x(7)\sin\frac{7\pi}{4}$$

From the above equation, we obtain the real and imaginary coefficients as shown below.

$$X_R(1) = (x(0) - x(4) + K\,(x(1) - x(3) + x(5) - x(7))) \tag{4.8}$$

$$X_I(1) = (-x(2) + x(6) - K\,(x(1) + x(3) - x(5) + x(7))) \tag{4.9}$$

Equation (4.8) and (4.9) explain the real output and imaginary output values respectively. Finally we combine these two equation to we get the output X(1).

$$X(1) = X_R(1) + j\, X_I(1) \tag{4.10}$$

Similar way to calculate the values of X(2), X(3) and X(4) and rest of the outputs are calculated by using symmetry property,

$$X(k) = X(N - k)^{*} \tag{4.11}$$

For example we find the output value of X(5). Now the proposed architecture defined the 8-point DFT, so N=8 and k=5 then put the value in equation (4.11)

$$X(5) = X(8 - 5)^{*} = X(3)^{*}$$

The output value of X(5) is an complex conjugate of X(3).

4.2 **(b) 8 – point Complex Inputs**

Next part we take complex inputs value and find the outputs of the DFT

General representation of the inputs is,

$$x_{in}(n) = x(n) + j\,y(n) \qquad (4.12)$$

Putting the values of equ (4.12) in equ (4.1), we get complex outputs values as,

$$X(k) = \sum_{n=0}^{N-1} (x(n) + j\,y(n))e^{(-\frac{j2\pi kn}{N})} \quad n = 0, 1 \ldots N\text{-}1 \qquad (4.13)$$

Now putting different values of k between 0 and 7 to get the outputs as follows:

$$X(0) = (x(0) + x(1) + x(2) + x(3) + x(4) + x(5) + x(6) + x(7)) + j\,(y(0) + y(1) + y(2) + y(3) + y(4) + y(5) + y(6) + y(7)) \qquad (4.14)$$

$$X(1) = x(0) + j\,y(0) + (x(1) + j\,y(1))\,e^{\frac{-j\pi}{4}} + (x(2) + j\,y(2))\,e^{\frac{-j2\pi}{4}} + (x(3) + j\,y(3))\,e^{\frac{-j3\pi}{4}} + (x(4) + j\,y(4))\,e^{\frac{-j4\pi}{4}} + (x(5)+jy(5))e^{\frac{-j5\pi}{4}} + (x(6+jy(6))e^{\frac{-j6\pi}{4}} + (x(7)+jy(7))e^{\frac{-j7\pi}{4}} \qquad (4.15)$$

Now realize the equation (4.15) and find the real and imaginary part of the outputs,

$$X_R(1) = [x(0) + y(2) - x(4) - y(6)] + K\,[x(1) + y(1) - x(3) + y(3) - x(5) - y(5) + x(7) - y(7)] \qquad (4.16)$$

$$X_I(1) = [y(0) - x(2) - y(4) + x(6)] + K\,[y(1) - x(1) - y(3) - x(3) - y(5) + x(5) + y(7) + x(7)] \qquad (4.17)$$

Equation (4.16) and (4.17) represent the real and imaginary part of X(1) respectively. Combining these two equations, we get the complex output of X(1) as

$$X(1) = X_R(1) + j\,X_I(1)$$

In similar fashion we get the rest of the output valves. In the case of complex inputs, we can’t use symmetry property. So we find the rest of the outputs as follows:

X(2) = [x(0) + y(1) – x(2) – y(3) + x(4) + y(5) – x(6) – y(7)] + j [y(0) – x(1) – y(2) + x(3) + y(4) – x(5) – y(6) + x(7)]

X(3) = [x(0) – y(2) – x(4) + y(6) + K (y(1) – x(1) + y(3) + x(3) + x(5) – y(5) – x(7) – y(7))] + j [y(0) + x(2) – y(4) – x(6) + K (- y(1) – x(1) + y(3) – x(3) + y(5) + x(5) – y(7) + x(7)]

X(4) = [x(0) – x(1) + x(2) – x(3) + x(4) – x(5) + x(6) + x(7)] + j [y(0) – y(1) + y(2) – y(3) + y(4) – y(5) + y(6) – y(7)]

X(5) = [x(0) + y(2) – x(4) – y(6) + K (- x(1) – y(1) + x(3) – y(3) + x(5) + y(5) + y(7) – x(7))] + j [y(0) – x(2)]

X(6) = [x(0) – y(1) – x(2) + y(3) + x(4) – y(5) – x(6) + y(7)] + j [y(0) + x(1) – y(2) – x(3) + y(4) + x(5) – y(6) – x(7)]

X(7) = [x(0) – y(2) – x(2) + y(6) + K (x(1) – y(1) – x(3) – y(3) – x(5) + y(5) + x(7) + y(7))] + j [y(0) + x(2) – y(4) – x(6) + K (y(1) + x(1) + x(3) – y(3) – y(5) – x(5) – x(7) + y(7))]

The value of K is getting from NEDA technique that was explained in chapter 3

Where K = $\cos\frac{\pi}{4} = 0.7071 = \begin{bmatrix} -2^{0} & 2^{-1} & \cdots & 2^{-12} \end{bmatrix} \begin{bmatrix} 0 \\ 1 \\ 0 \\ 1 \\ 1 \\ 0 \\ 1 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$

The figure 4.1, it is evident that the core is divided into two sub modules which compute coefficients for real and imaginary parts separately. From the lower level of RTL schematic, the number of adder/subtractors required for computing terms is 31. In addition to the above, the number of shifters used in the proposed design is 11. As there are no multipliers, though there is a huge increase in number of adders, it greatly reduces the hardware complexity.

The inputs and outputs of the design are registered thus making the design sequential and also synchronous.

Compared to the traditional implementation of FFT that consumes 24 adder/subtractors and 12 multipliers, the proposed design, though has 79 adder/subtractors in total, the hardware is greatly reduced as there are no multipliers.

The width of the inputs is taken to be 16 bits and the same is maintained throughout the design, which makes the design have a constant data path of 16 bits. Thus the width of the outputs is also 16 bits. The average of the inputs is limited to 4095 as the width of the data path is fixed.

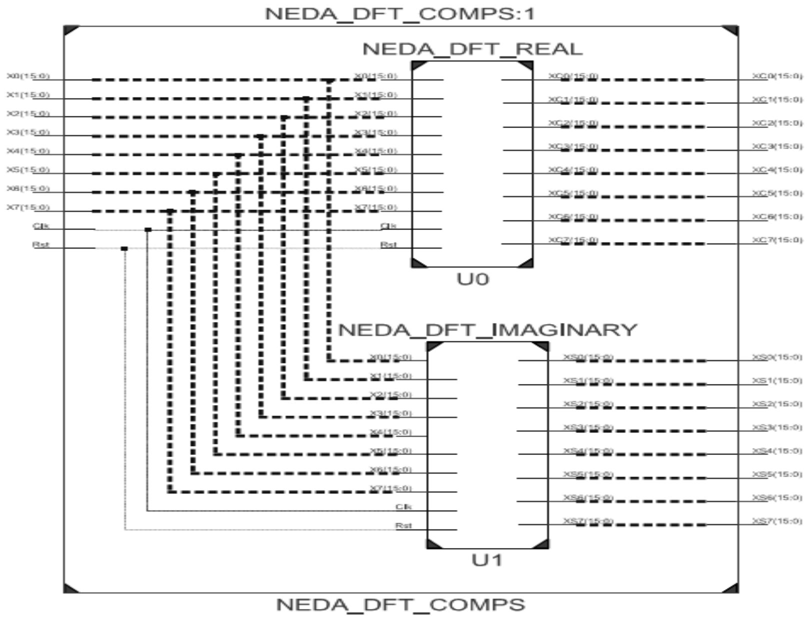


Figure 4.1. RTL schematic of top module of the proposed design

4.3 RESULTS:

The design is coded in VHDL using the software Xilinx ISE 10.1. The design is mapped onto the Xilinx made XC2VP30-7FF896 FPGA. All inputs and outputs are declared in signed two's complement number system. As mentioned earlier, the data path width considered in the design is 16 bits and thus, the total number of bonded IOBs required for the proposed design is 386. The inputs are considered both in integer and fixed point. As the width of the data path is fixed, no overflow should occur in carrying out the arithmetic operations.

Table 4.1 compares the results of the proposed design, taken using Xilinx ISim, to the results obtained by MATLAB. The inputs and outputs are declared using fixed point representation.

TABLE 4.1. COMPARISON OF OUTPUTS OBTAINED USING MATLAB AND XILINX ISE, FOR FIXED POINT REPRESENTATION

Inputs	**Matlab Outputs**	**Xilinx Outputs**
20.1367	113.64	113.6523
21.1289	1.37 – 5.18j	1.3554 – 6j
31.0703	4.95 – 36.07j	4.9492 – 37.9296j
5.0003	6.77 – 3.41j	6.7539 – 6.75j
16.0664	21.27	21.2617
20.0003	6.77 + 3.41j	6.7539 + 3.375j
15.1835	4.95 + 36.07j	4.9492 + 36.0703j
0.0585	1.37 + 5.18j	1.3554 + 5.1562j

From table 4.1, it is evident that the proposed design is producing near exact outputs. Thus from the table 4.1, the input sequence considered is of 8 samples. The range of inputs has to be maintained on an average as large variations among the inputs effect the outputs in the case of Xilinx outputs. Table 4.2 compares the results of the proposed design, taken using Xilinx ISim, to the results obtained by MATLAB. The inputs and outputs are declared using integer representation.

From table 4.2, it is evident that the difference between the outputs obtained through MATLAB and those obtained through Xilinx ISE is permissible.

Table 4.3 compares the results of the proposed design, taken using Xilinx ISim, to the results obtained by MATLAB. The inputs and outputs are declared using complex point representation.

From table 4.3, it is evident that the difference between the outputs obtained through MATLAB and those obtained through Xilinx ISE is permissible.

Table 4.4 compares the number of arithmetic units required for the proposed design with that of traditional methods. Thus from the table, it is evident that the hardware required for the proposed design is less, though the number of adders is more, since there are no multipliers in the proposed design. The input sequence is assumed to be real in nature.

TABLE 4.2. COMPARISON OF OUTPUTS OBTAINED USING MATLAB AND XILINX ISE, FOR INTEGER POINT REPRESENTATION

Inputs	**Matlab Outputs**	**Xilinx Outputs**
35	167	167
33	50.53 + 16.27j	49 + 19j
18	-13 – 18j	-13 – 18j
1	-14.53 – 41.73j	-17– 38j
17	67	67
1	-14.53 + 41.73j	-17 + 38j
47	-13 + 18j	-13 + 18j
15	50.53 – 16.27j	49 – 19j

TABLE 4.3. COMPARISON OF OUTPUTS OBTAINED USING MATLAB AND XILINX ISE, FOR COMPLEX POINT REPRESENTATION

Inputs	Matlab Outputs	Xilinx Outputs
32 + 3j	486 + 6j	486 + 6j
34 + 3j	- 31.5 + 102.4j	- 32 + 96j
45	- 30 + 44j	- 30 + 44j
55	-30.17 + 18.15j	- 31 + 16j
65	-32	- 32
75	- 34.4 – 16.4j	- 38 – 19j
85	- 36 – 38j	- 36 – 38j
95	- 35.8 – 92.18	- 39 – 94j

The proposed method of implementing DFT can be increased to higher point according to the requirements of the system. If the outputs are expected to be more accurate, then the precision of the W matrix has to be increased.

Table 4.5 compares the proposed design with the designs given in [4].

From table 4.5, it is clear that the proposed design has occupied far less hardware than other designs mentioned. In addition to that, the proposed design also has higher clock frequency and better throughput. The device used to map for generating the comparison is EP2C35F672C6 from Altera’s Cyclone II family. The width of the inputs, outputs and the data path considered is 12 bits. The maximum frequency obtained is through the slow model using Altera’s TimeQuest Timing Analyzer.

Table 4.6 shows the device utilization summary of the proposed design in XC2VP30-7FF896 FPGA

TABLE 4.4. COMPARISON OF THE NUMBER OF ARITHMETIC UNITS REQUIRED TO PERFORM 8 POINT DFT

	Direct implementation	**FFT implementation**	**Proposed design for fixed and integer point(NEDA implementation)**	**Proposed design for complex point(NEDA implementation)**
Adder/Subtractor	56	24	31	106
Multiplier	64	12	0	0

TABLE 4.5. COMPARISON OF THE PROPOSED DESIGN WITH THE ONES MENTIONED IN[4].

	[#LC]	**[#DSP]**	**Clock frequency [MHz]**	**Throughput [Mbit/s]**
FFT_LC	4723 (14%)	-		
FFT_DSP	1554 (5%)	70 (100%)	48.93	587.16
FFT_DA	7222 (22%)	-	74.36	892.32
DFT_NEDA for fixed and integer point (Proposed)	869 (3%)	-	79.64	955.68
DFT_NEDA for complex point (Proposed)			122.704	1963.264

.

TABLE 4.6. DEVICE UTILIZATION SUMMARY OF THE PROPOSED DESIGN IN XILINX XC2VP30-7FF896 FPGA

Logic Utilization	**Used**		**Available**	**Utilization**	
	Fixed point	complex point		Fixed point	complex point
Number of slices	295	925	13696	2%	6%
Number of slice Flip Flops	304	760	27392	1%	2%
Number of 4 input LUTs	478	1602	27392	1%	5%

Figure 4.2 shows the simulation window obtained in Xilinx Isim simulator. Figure 4.3 shows the Post Place and Route Simulation output of Xilinx Isim for integer representation of inputs. Figure 4.4 shows the Chip Scope Pro Analyzer Simulation output of Xilinx Isim for integer representation of inputs.

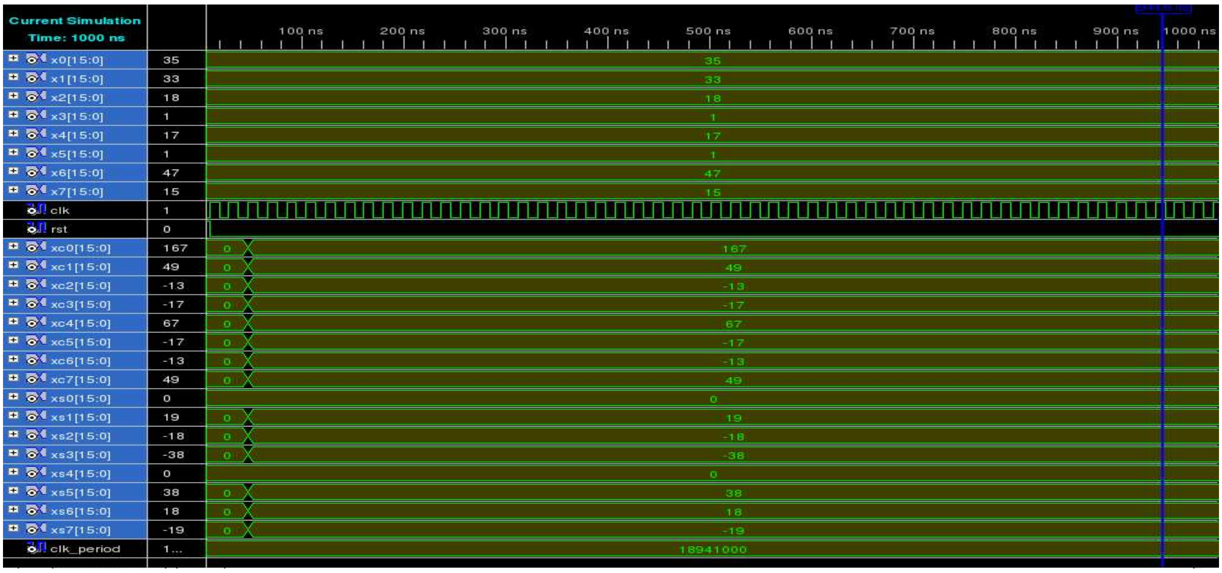


Figure 4.2. Simulation output of Xilinx Isim for integer representation of inputs

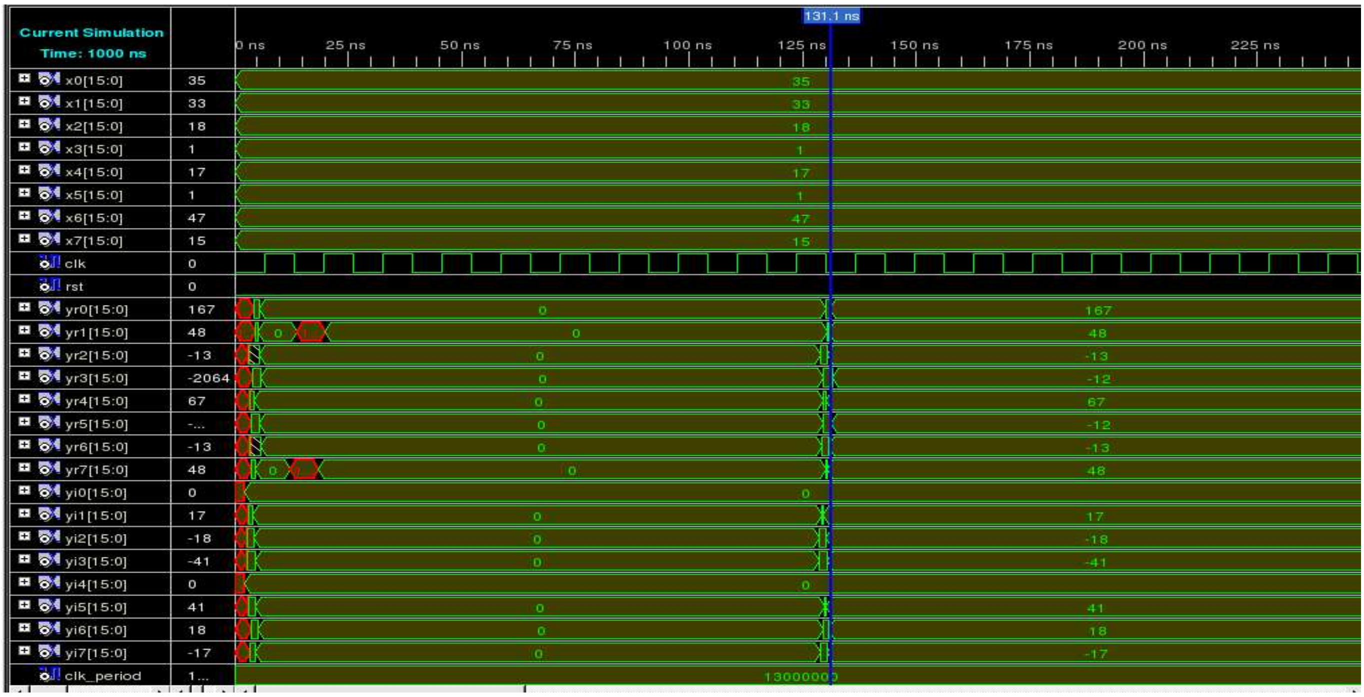
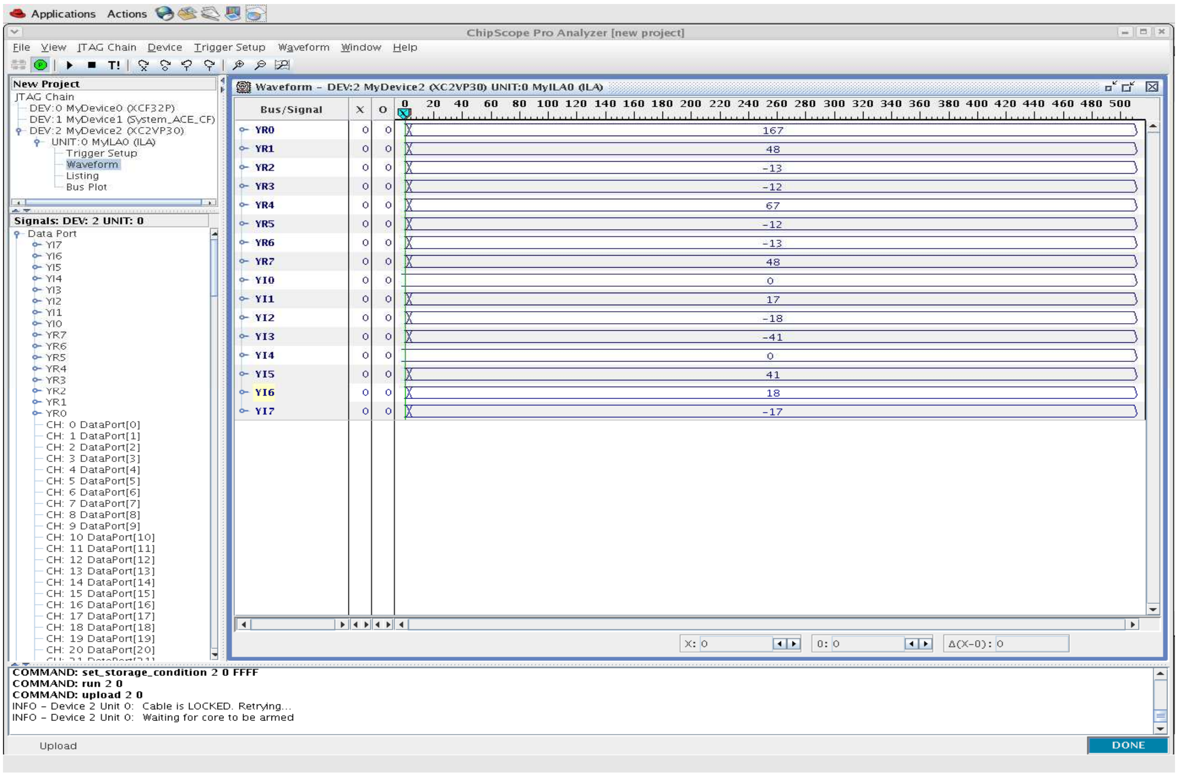


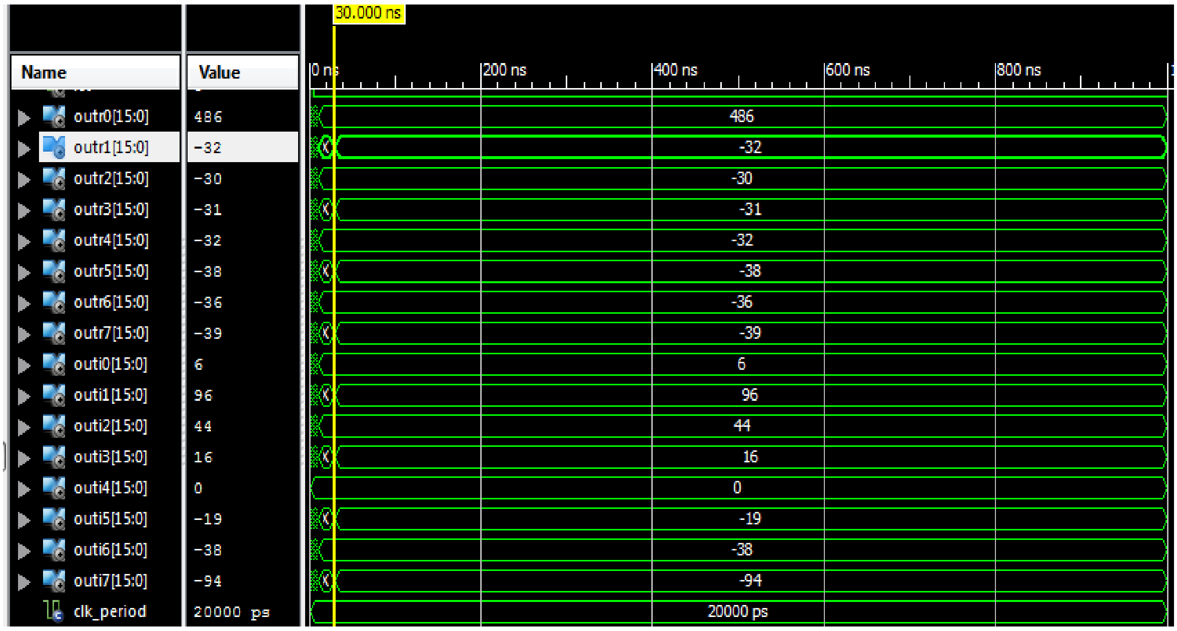
Figure 4.3. Post Place and Route Simulation output of Xilinx Isim for integer representation of inputs



```
COMMAND: set_storage_condition 2 0 FFFF
COMMAND: run 2 0
COMMAND: upload 2 0
INFO - Device 2 Unit 0:  Cable is LOCKED. Retrying...
INFO - Device 2 Unit 0:  Waiting for core to be armed
```

Figure 4.4. Chip Scope Pro Analyzer output for integer representation of inputs

Figure 4.5 shows the simulation output of Xilinx Isim for complex representation of inputs.



Name	Value
outr0[15:0]	486
outr1[15:0]	-32
outr2[15:0]	-30
outr3[15:0]	-31
outr4[15:0]	-32
outr5[15:0]	-38
outr6[15:0]	-36
outr7[15:0]	-39
outi0[15:0]	6
outi1[15:0]	96
outi2[15:0]	44
outi3[15:0]	16
outi4[15:0]	0
outi5[15:0]	-19
outi6[15:0]	-38
outi7[15:0]	-94
clk_period	20000 ps

Figure 4.5. Simulation output of Xilinx Isim for complex representation of input

4.4 SUMMARY:

This chapter has reported architecture of a DFT core, which is employed in many communication systems, using NEDA, a ROM less and multiplier less method. The proposed design is hardware efficient as compared to other traditional methods as well as architectures that are built using only DA. The proposed design is implemented on Xilinx XC2VP30-7FF896 FPGA, which occupies an area ratio of 6% of the total configurable area. The proposed design in total has 79 adders with zero multipliers and no ROM, thus making possible in implementing higher point DFT. The maximum frequency of operation of the proposed design on the mapped FPGA is 122.704 MHz. The proposed design is also compared to other designs mentioned in [4], which shows great reduction in hardware (44.08%) and improvement in maximum clock frequency (65%).

Chapter 5

16-Point Radix-4 Complex FFT

5.1 Introduction:

Fast Fourier Transform (FFT) is one of the most efficient ways to implement Discrete Fourier Transform (DFT) due to its reduced usage of arithmetic units. DFT is one of those primary tools that are used for the frequency analysis of discrete time signals and to represent a discrete time sequence in frequency domain using its spectrum samples. The analysis (forward) and synthesis (inverse) equations of an N point FFT are given below.

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

$$x[n] = \frac{1}{N}\sum_{k=0}^{N-1} X[k]\, W_N^{-kn} \tag{5.1}$$

Where, $W_N^{kn} = e^{-j\frac{2\pi}{N}kn}$. As evident from equation (5.1), the basis of both synthesis and analysis equations remains same thus increasing the scope of the architecture to both analyse and synthesize. Due to increased employability of FFT in modern electronic systems, higher radix FFTs such as radix – 4, radix – 8, radix – 2^k, split radix, etc. are designed for improved timing and reduced hardware. The basic difference of the mentioned methods lies in the structure of their butterfly units. In section 5.2 and 5.3 we discussed the proposed design and results and discussion respectively. Finally in section 5.4 we conclude the proposed architecture.

5.2 16-point complex FFT:

The implementation of 16-point complex FFT using radix-4 method has been proposed. Complex multiplications required during the process have been implemented by using NEDA. According to the radix-4 algorithm, to implement 16-point FFT, eight radix-4 butterflies are required. Four radix-4 butterflies are used in the first stage and the other four being used in the second/final stage. The input is taken in normal order and the output in bit-reversal order. The output of each radix-4 butterfly is multiplied by the respective twiddle factors [7].

In the shown block diagram, the first stage consists of four radix-4 butterflies. The inputs to the butterflies are x(n), x(n+4), x(n+8), x(n+12) where n is 0 for first butterfly, 1 for second butterfly, 2 for the third butterfly, and 3 for the last butterfly, all of first stage. The twiddle factors are given byW_{16}^{0}, W_{16}^{q}, W_{16}^{2q}, $W_{16}^{3q}$where q is 0 for first butterfly, 1 for second butterfly, 2 for third butterfly, and 3 for the last butterfly, all of first stage. The outputs of first stage are multiplied with respective twiddle factors and are given as inputs to the second stage. As proposed in our design, the complex twiddle multiplications required at the stage-1 output have been implemented by using NEDA blocks. Overall 9 NEDA blocks are required at the output of first stage of the 16 point FFT processor.

In the second stage, 4 more radix-4 butterfly blocks are used. The first radix-4 butterfly in the second stage takes the first output of the 4 radix-4 butterfly blocks used in the first stage. The second radix-4 butterfly in the second stage takes the second output of the 4 radix-4 butterfly blocks followed by the NEDA block (if required). This process continues for the rest radix-4 butterfly blocks present in the second stage. There is no need of using any NEDA block after second stage as the twiddle factor W_{16}^{0} that is 1 is multiplied to the outputs of the second stage. The final output comes in a bit-reversal order.

The advantage of using radix-4 algorithm is that it retains the simplicity of radix-2 algorithm and gives the output with lesser complexity [8]. The NEDA block shown in the proposed block diagram does the complex multiplication of the output of the first stage and the respective twiddle factor. The twiddle factor values used here are as follows.

$$W_{16}^{1} = \cos\frac{\pi}{8} - j\sin\frac{\pi}{8} = 0.9238 - j0.3826$$

$$W_{16}^{2} = \cos\frac{\pi}{4} - j\sin\frac{\pi}{4} = 0.7071 - j0.7071$$

$$W_{16}^{3} = \cos\frac{3\pi}{8} - j\sin\frac{3\pi}{8} = 0.3826 - j0.9238$$

$$W_{16}^{6} = \cos\frac{3\pi}{4} - j\sin\frac{3\pi}{4} = -0.7071 - j0.7071$$

$$W_{16}^{9} = \cos\frac{9\pi}{8} - j\sin\frac{9\pi}{8} = -0.9238 + j0.3826 \qquad (5.2)$$

The product of a complex number and a twiddle factor is given by $(R + jI)(\cos\theta + j\sin\theta) = (Rcos\theta - I\,sin\,\theta) + j(R\,sin\,\theta + I\,cos\,\theta)$. For a constant θ, cosine and sine values are constant. So, we can find $Rcos\theta$, $I\,sin\,\theta$, $R\,sin\,\theta$, and $I\,cos\,\theta$.

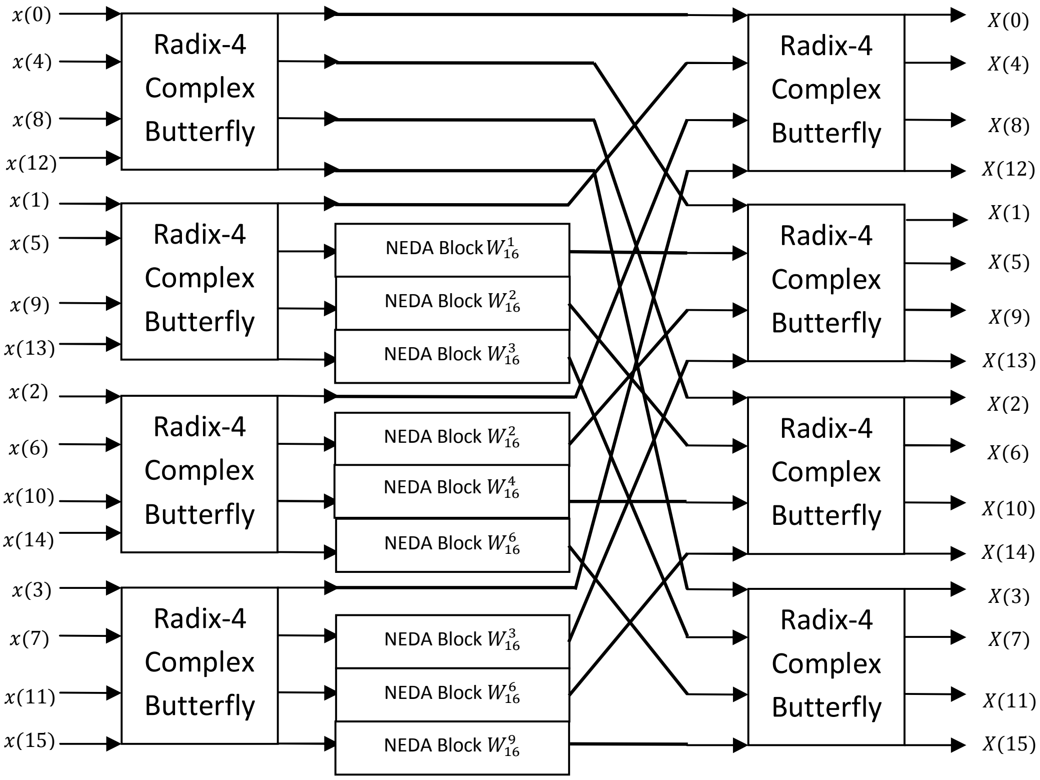


Figure 5.1. Block diagram of the proposed architecture

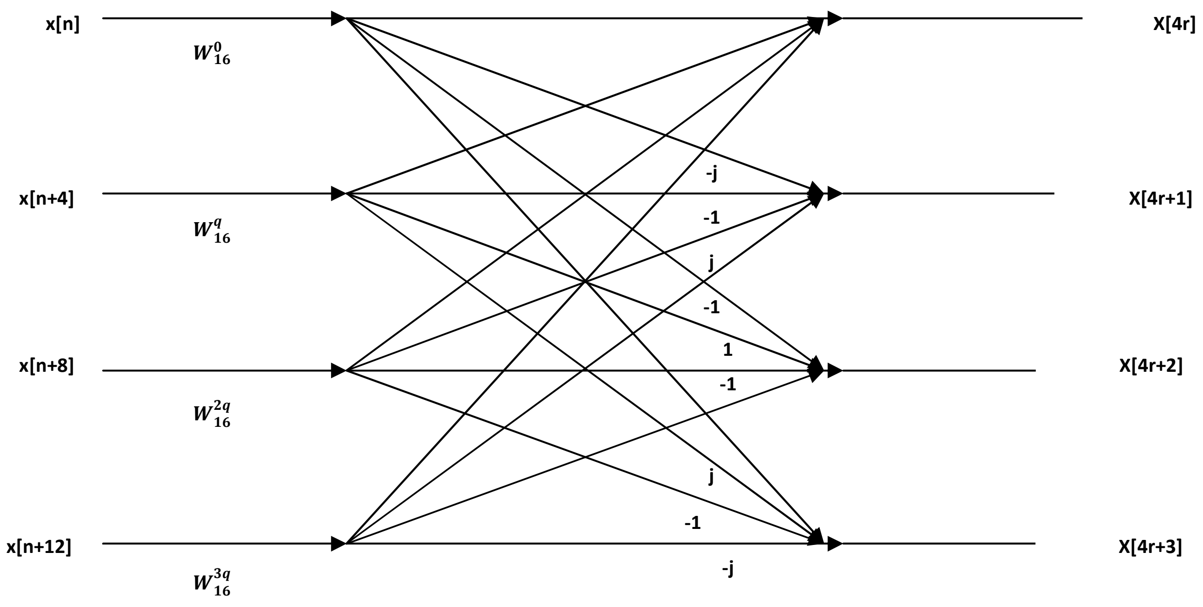


Figure 5.2. Radix-4 butterfly structure.

5.3 Results:

The proposed architecture has been implemented using Xilinx ISE v10.1. To map the design, the FPGAs selected are XC2VP100-6FF1704, XC2V8000-5FF1517 and XC5VLX330-2FF1760. The results are checked for integer inputs. The design is coded in VHDL, declaring all inputs and outputs in signed two's complement number system. The width of data path in the proposed design is 16-bits which includes the width of inputs and outputs. The number of real adders and multipliers required for calculation of an N-point FFT using radix-4 algorithm are given by $3N\log_2 N$ and $(3N/2)\log_2 N$ respectively. A comparison between the number of arithmetic units required in conventional and proposed design is depicted in table5.2. Table 5.3 figures out the performance metrics of the proposed design on different FPGAs, which include throughput and slice delay product. Table 5.4 shows device utilization summary of proposed design on different FPGAs. From table 5.3, it is clear that the number of slices occupied in XC5VLX330-2FF1760 is less compared to other two FPGAs.

TABLE 5.1. COMPARISON OF OUTPUTS OBTAINED USING MATLAB AND XILINX ISE

MATLAB outputs	**Xilinx ISim Outputs**
1783	1783
-171.6-j783.9	-185-j781
-286.7+j149.3	-287+j152
-96.3-j447	-103-j444
-943+j228	-943+j228
477.8+j687.8	483+j693
298.7-j604.7	299-j602
-337.9+j375	-336+j381
719	719
-337.9-j375	-335-j379
298.7+j604.7	299+j602
477.8-j687.8	485-j682
-943-j228	-943-j228
-96.3+j447	-91+j443
-286.7-j149.3	-287-j152
-171.6+j783.9	-174+j769

TABLE 5.2. COMPARISON OF NUMBER OF ARITHMETIC UNITS REQUIRED TO PERFORM 16-POINT RADIX-4 COMPLEX FFT

Arithmetic unit	**Conventional design**	**Proposed design**
Real adder/Subtractor	192	269
Real multiplier	96	0

Table 5.5 compares the power analysis of the proposed design on different FPGAs. The software tool used to analyze power is Xilinx Xpower Analyzer. The proposed design has also been implemented in synopsys for both 65 nm and 180 nm technologies. Table 5.6 shows the power report and area report of the proposed

design in ASIC using synopsys for TCBN65GPLUSTC and FSA0A_C_GENERIC_CORE_FF1P98VM40C libraries. The flow used is synopsys DC synthesis.

TABLE 5.3. PERFORMANCE METRICS OF PROPOSED DESIGN ON DIFFERENT FPGAs.

	XC2VP1000s6FF1704	**XC2V8000-5FF1517**	**XC5VLX330-2FF1760**
Frequency (MHz)	61.831	54.478	119.27
Throughput (Mbit/s)	989.296	871.648	1828.32
Slice delay product	38.64	44.49	9.18

TABLE 5.4. DEVICE UTILIZATION SUMMARY OF PROPOSED DESIGN ON DIFFERENT FPGAs

Logic Utilization	**Used**			**Utilization**		
	XC2VP10006FF1704	XC2V8000-5FF1517	XC5VLX330-2FF1760	XC2VP10006FF1704	XC2V8000-5FF1517	XC5VLX330-2FF1760
Number of Slices	2389	2424	1096	5%	5%	2%
Number of Slice Flip Flops	1913	1904	1894	2%	2%	1%
Number of 4-input LUTs	3972	4054	3469	4%	4%	1%

TABLE 5.5. POWER ANALYSIS OF PROPOSED DESIGN ON DIFFERENT FPGAs

Frequency (MHZ)	**Total Quiescent (W)**			**Total Dynamic Power (W)**			**Total Power (W)**		
	XC2VP100-6FF1704	XC2V8000-5FF1517	XC5VLX330-2FF1760	XC2VP100-6FF1704	XC2V8000-5FF1517	XC5VLX330-2FF1760	XC2VP100-6FF1704	XC2V8000-5FF1517	XC5VLX330-2FF1760
12.5	0.20438	0.13785	3.32919	0.30797	0.47152	0.07869	0.51234	0.60937	3.40789
20	0.20438	0.13785	3.55222	0.37262	0.57793	0.57921	0.57699	0.71578	4.13143
30	0.20438	0.13785	3.68024	0.40755	0.69395	0.83959	0.61193	0.83180	4.51983
40	0.20438	0.13785	3.81776	0.45378	0.79855	1.09989	0.65815	0.93640	4.91766
50	0.20438	0.13785	3.96607	0.52452	0.93893	1.36013	0.72889	1.07678	5.32617

TABLE 5.6. POWER AND AREA REPORT OF PROPOSED DESIGN FOR SYNOPSYS DC IMPLEMENTATIONS

	TECHNOLOGY LIBRARY	
	TCBN65GPLUSTC	**FSA0A_C_GENERIC_CORE_FF1P98VM40C**
Total Dynamic Power (mW)	22.7303	302.5547
Total Cell Area	21039.480469	161733.406250

5.4 SUMMARY:

This chapter has reported architecture of 16-point radix-4 complex FFT core using NEDA which is a ROM less and multiplier less method. The proposed design is efficient in terms of hardware as compared to other traditional methods. The proposed design has been implemented on different FPGAs to compare the performance metrics. Number of real adders/subtractors required to implement the proposed design is 269.

The maximum frequency obtained is 119.27 MHz on XC5VLX330 FPGA. The throughput and slice delay product obtained in XC5VLX330 FPGA are 1828.32 Mbit/s and 9.18. In terms of power consumption, XC2V8000 FPGA shows better result than others. The proposed design occupied 21039.480469 units of cell area for 65 nm technology and 161733.406250 units of cell area for 180 nm technology, when synthesized using synopsys DC. Power reports using synopsys show that the consumption is 22.7303 mW for 65 nm technology and 302.5547 mW for 180 nm technology.

Chapter 6

Radix-2^2 Folded Pipelined Complex FFT

6.1 Introduction:

In this chapter we implemented 16 – point FFT using radix–2^2. Previous chapter also discussed a 16–point using radix–4. But radix–2^2 shows better performance with respect to radix–4. Techniques in designing DSP systems such as folding, pipelining have always improved performance of the systems in terms of area of hardware, latency, frequency, etc. To determine the control circuits systematically in DSP architectures, the folding transformation is used. In folding, time multiplexing of multiple algorithm operations to a single functional unit is done. Thus, in any DSP architecture, folding provides a means for trading time for area. In general, folding can be used to reduce the number of hardware functional units by a factor of N at the expense of increasing the computation time by a factor of N [4]. To avoid excess amount of registers used in these architectures that occur while folding, there are techniques that compute the minimum number of registers needed to implement a folded DSP architecture. These techniques also help in allocation of data in these registers. Pipelining transformation in DSP architectures reduces critical path, which can be pro-sequenced to either decrease the power consumption or to enhance the sample frequency or clock speed. By introducing the pipelining latches along the data-path, critical path in particular, pipelining technique reduces the effective critical path. This technique has been used in areas such as compiler synthesis and architecture design [21].

6.2 Radix-2^2 Complex FFT:

6.2(a). Architecture – I:

The proposed architecture – I is designed that the complex rotators are replaced using NEDA blocks. The advantage of NEDA blocks lies in its simplest form of add-shift architecture that occupies very less area. Figure 6.9 shows the proposed architecture – I.

The data flow remains same in proposed architecture – II as in that of architecture – I. Since the NEDA blocks are computationally simple, they are implemented using combinatorial processes. Thus, there is no need of the delay traverse blocks in proposed architecture – II. This is a good sign since it reduces the latency of the proposed architecture.The R2CBF has been implemented as a radix-2 block. The blocks R2CBFWN and R2CBFW2N are similar to R2CBFWC and R2CBFW2C with NEDA blocks. The blocks are shown in figure 6.1 and figure 6.2 respectively.

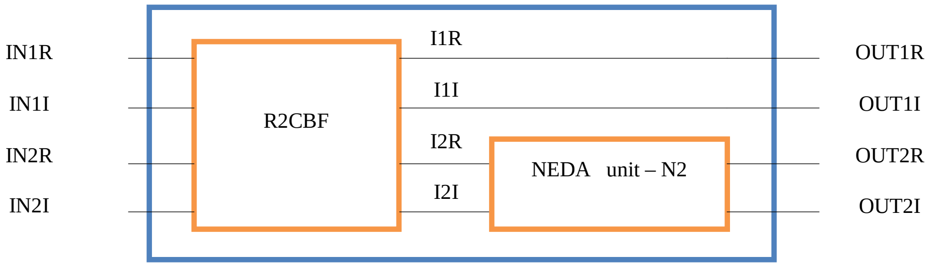


Figure 6.1. Internal architecture of R2CBFWN block

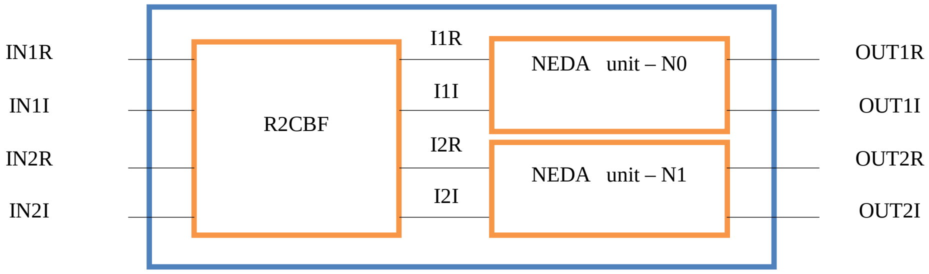


Figure 6.2. Internal architecture of R2CBFW2N

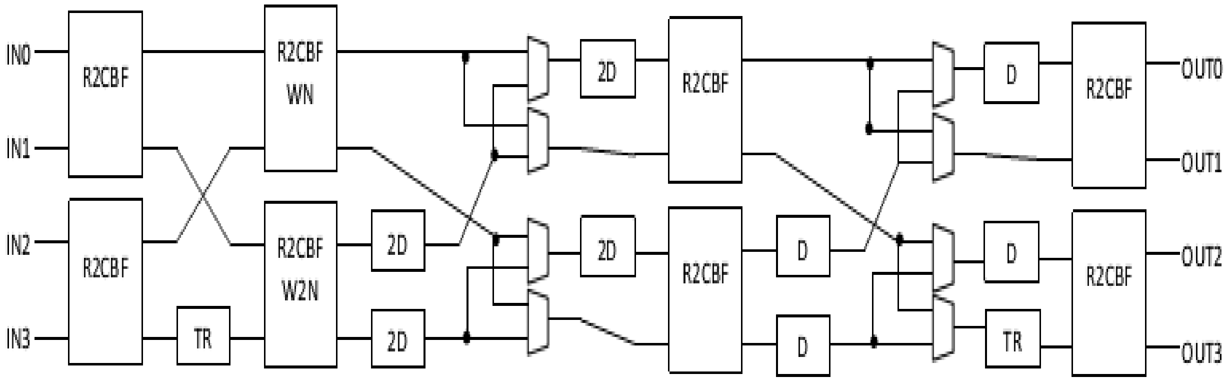
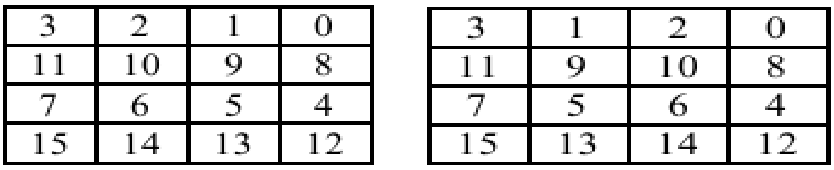


Figure 6.3. Radix-2^2 4-parallel 16-point feedforward complex FFT core using NEDA

The data flow and the order of input outputs are as mentioned in figure 6.6.The multiplications done with twiddle factors vary even in architecture – II for each clock cycle and repeat after every 4 clock cycles. The twiddle factors that are considered for multiplication in NEDA blocks for each clock cycle are listed in table 6.2.

The order of inputs taken and outputs produced by the proposed architecture – I is shown in figure 6.4.



3	2	1	0
11	10	9	8
7	6	5	4
15	14	13	12

(a)

3	1	2	0
11	9	10	8
7	5	6	4
15	13	14	12

(b)

Figure 6.4. Data flow order in the proposed architecture – I of (a) inputs and (b) outputs.

TABLE 6.1 TWIDDLE FACTORS CONSIDERED FOR MULTIPLICATION WITH NEDA UNITS

Clock cycle	**N2**	**N0**	**N1**
1	W_{16}^{0}	W_{16}^{0}	W_{16}^{0}
2	W_{16}^{2}	W_{16}^{1}	W_{16}^{3}
3	W_{16}^{4}	W_{16}^{2}	W_{16}^{6}
4	W_{16}^{6}	W_{16}^{3}	W_{16}^{9}

6.2 (b). Architecture – II:

The proposed architecture – II is implemented using folding but its architecture is slightly different when compared with the other two proposed architectures. It is designed using finite state machine (FSM) approach. Thus, the design is a fully pipelined one. The number of stages remains same that is four. The FSM is so designed that there is no requirement of the data shufflers that were present in the earlier proposed architectures. The architecture has intermediate registers to store values of intermediate outputs for every clock cycle.

The NEDA block after the first stage represents the multiplications of the outputs, of the first stage FFT, with the twiddle factors W^0. Thus this NEDA block does not constitute any adders and shifters. The outputs of every stage in the architecture are given to multiplexers that select the proper twiddle factor for multiplications. Table 6.3 shows the data flow graph in the proposed architecture – II. The order of inputs and outputs of the proposed architecture – II is shown in figure 6.5.

The contents of table – III are described here. For first four clock cycles the proposed architecture takes the inputs in the order mentioned in figure 6.5(a). In clock cycles 1 to 5, the outputs after first stage radix-2^2 computation are obtained. These are denoted by T0 to T15. In clock cycles 6 to 9, the registered signals are given to second stage radix-2^2 butterfly unit. The signals W12, W13, W14, and W15 correspond to the values that are obtained after multiplying with respective twiddle factors after the first stage. The outputs of the second state computations are stored from clock cycle 7 to clock cycle 10 and are represented by Z0 to Z15. During clock cycles 11 to 14, the inputs are given to the third stage radix-2^2 unit. The signals X5, X6, X7, X9, X10, X11, X13, X14, and X15 are the products obtained after multiplying with respective twiddle factors. The outputs of third stage are obtained during clock cycles 12 to 15. The inputs for final stage radix-2^2 computation are given from clock cycle 16. Signals V3, V7, V11, and V15 are obtained after twiddle factor multiplication in third stage. The outputs of the proposed architecture are obtained starting from the clock cycle 17 till clock cycle 20.

3	2	1	0
11	10	9	8
7	6	5	4
15	14	13	12

(a)

3	1	2	0
11	9	10	8
7	5	6	4
15	13	14	12

(b)

Figure 6.5. Data flow order in the proposed architecture – II of (a) inputs and (b) outputs

TABLE 6.2. DATA FLOW GRAPH OF INPUT-OUTPUTS OF PROPOSED ARCHITECTURE – II

Clock Cycle	**Input**	**Output**
1	IN0,IN8,IN4,IN12	
2	IN1,IN9,IN5,IN13	T0,T8,T4,T12
3	IN2,IN10,IN6,IN14	T1,T9,Y5,T13
4	IN3,IN11,IN7,IN15	T2,T10,T6,T14
5		T3,T11,TT7,T15
6	T0,T4,T8,W12	
7	T1,T5,T9,W13	Z0,Z4,Z8,Z12
8	T2,T6,T10,W14	Z1,Z5,Z9,Z13
9	T3,T7,T11,W15	Z2,Z6,Z10,Z14
10		Z3,Z7,Z11,Z15
11	Z0,Z2,Z4,X6	
12	Z1,Z3,X5,X7	D0,D2,D4,D6
13	Z8,X10,Z12,X14	D1,D3,D5,D7
14	X9,X11,X13,X15	D8,D10,D12,D14
15		D9,D11,D13,D15
16	D0,D1,D2,V3	
17	D4,D5,D6,V7	Y0,Y8,Y4,Y12
18	D8,D9,D10,V11	Y2,Y10,Y6,Y14
19	D12,D13,D14,V15	Y1,Y9,Y5,Y13
20		Y3,Y11,Y7,Y15

6.3 RESULTS :

ASIC AND FPGA IMPLEMENTATION AND COMPARISON:

The proposed architectures are designed for FPGAs using Xilinx ISE. The family of FPGA chosen is Virtex-5 and the device chosen is XC5VSX240T-2FF1738. All the proposed architectures are implemented for 16 inputs with a word length of 16 bits. All inputs and outputs are represented by signed 2’s complement number system.

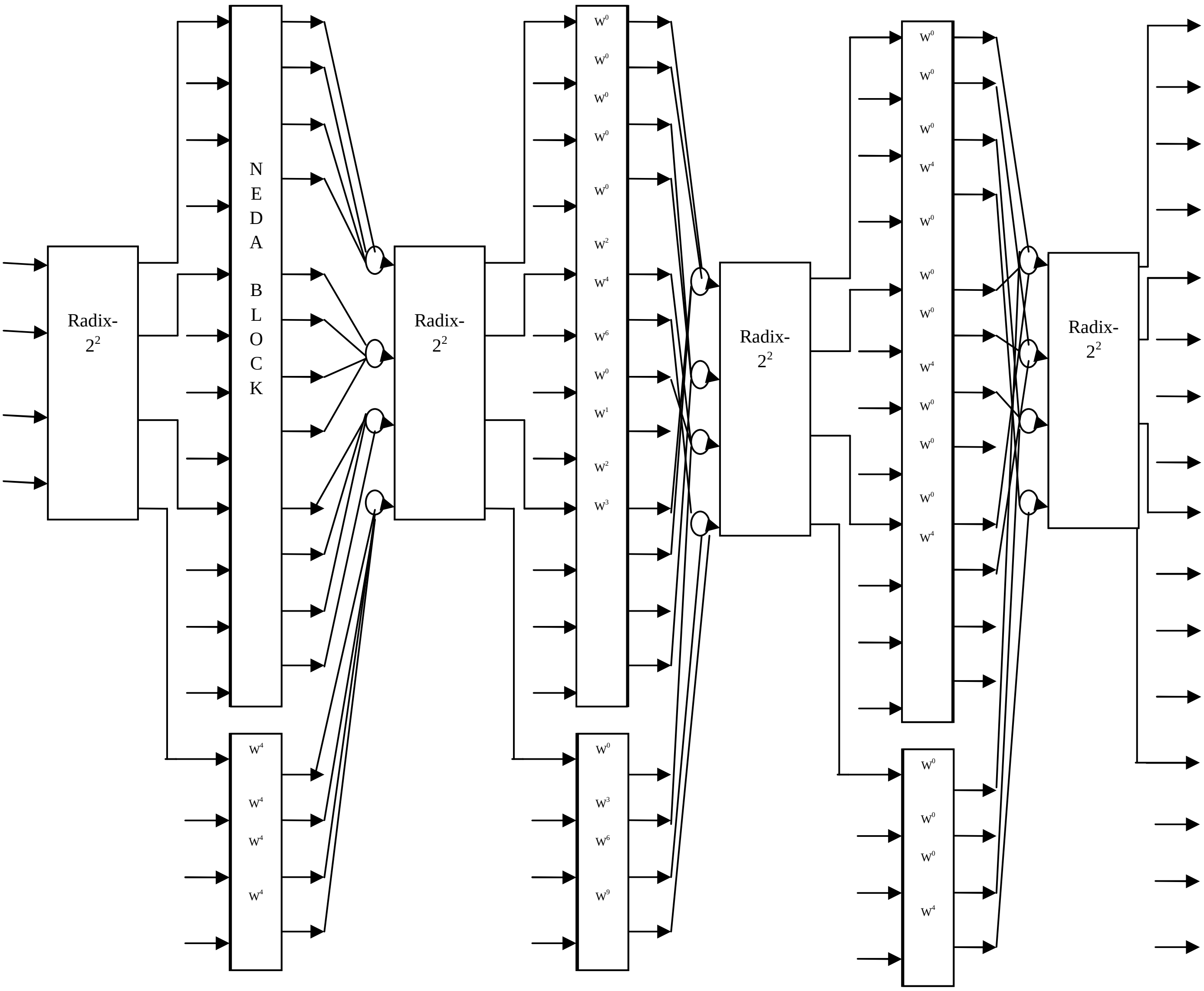


Figure 6.6. Architecture – II of 16 – point radix - 2^2 4 – parallel complex FFT core

Table 6.4 shows the device utilization summaries of the proposed architectures on the mapped FPGA. From table 6.4, it is clear that proposed architecture – I occupies least number of slices. Table 6.5 shows the comparison of proposed architectures in terms of FPGA hardware resources with the existing designs. From table 6.5, proposed architecture – II may be considered as the optimised design since it has the least slice-delay product of 2.196, among all proposed architectures mentioned in the table. Proposed architecture – II also has the highest throughput among the proposed one.

The proposed architectures are designed using Synopsys DC and Cadence SoC Encounter for ASIC design flow. The process technology used is 180nm. The designs are checked for design rule check (DRC) as well as for timing. Table 6.6 shows ASIC implementation results of the proposed architectures in 180nm technology.

TABLE 6.3. DEVICE UTILIZATION SUMMARY OF THE PROPOSED DESIGNS ON XC5VSX240T-2FF1738

Device Utilization	**Architecture – I**	**Architecture – II**
Number of occupied slices	575	1190
Number of slice LUTs	1882	2956
Number of slice registers	514	2491
Number of bonded IOBs	258	258

TABLE 6.4. COMPARISON OF ARCHITECTURES ON XC5VSX240T-2FF1738 FPGA

Design	**Area**		**Maximum Frequency (MHz)**	**Throughput (MS/s)**	**Slice-delay Product**
	Slices	**DSP48E**[*]			
[21]	386	12	458	1831	-
Architecture – I	575	0	100.263	401	5.735
Architecture – II	1190	0	541.947	2167	2.196

TABLE 6.5. ASIC IMPLEMENTATION RESULTS OF THE ARCHITECTURES

ASIC implementation results using Synopsys DC	**Process technology: 0.18µm**	
	Architecture – I	**Architecture – II**
Total cell area	200190.312500	291428.225070
Total dynamic power	10.7566 mW	2.6513 mW
Add-sub width	16 bits	16 bits
Slack at 50 MHz	13.85 ns	16.65 ns

Figure 6.7 and figure 6.8 show the physical layouts of architecture – I and architecture – II respectively.

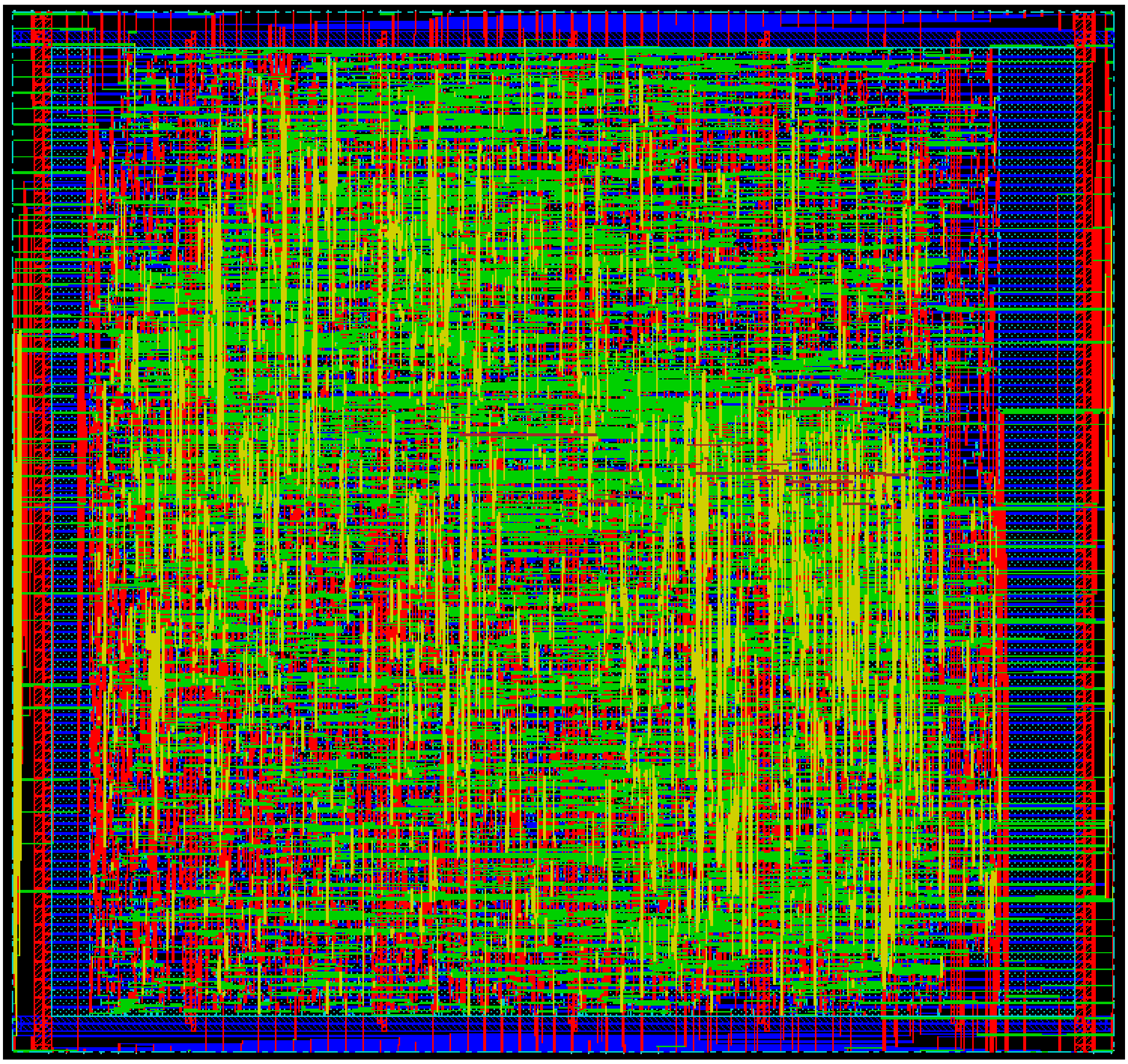


Figure 6.7. Physical layout of architecture – I

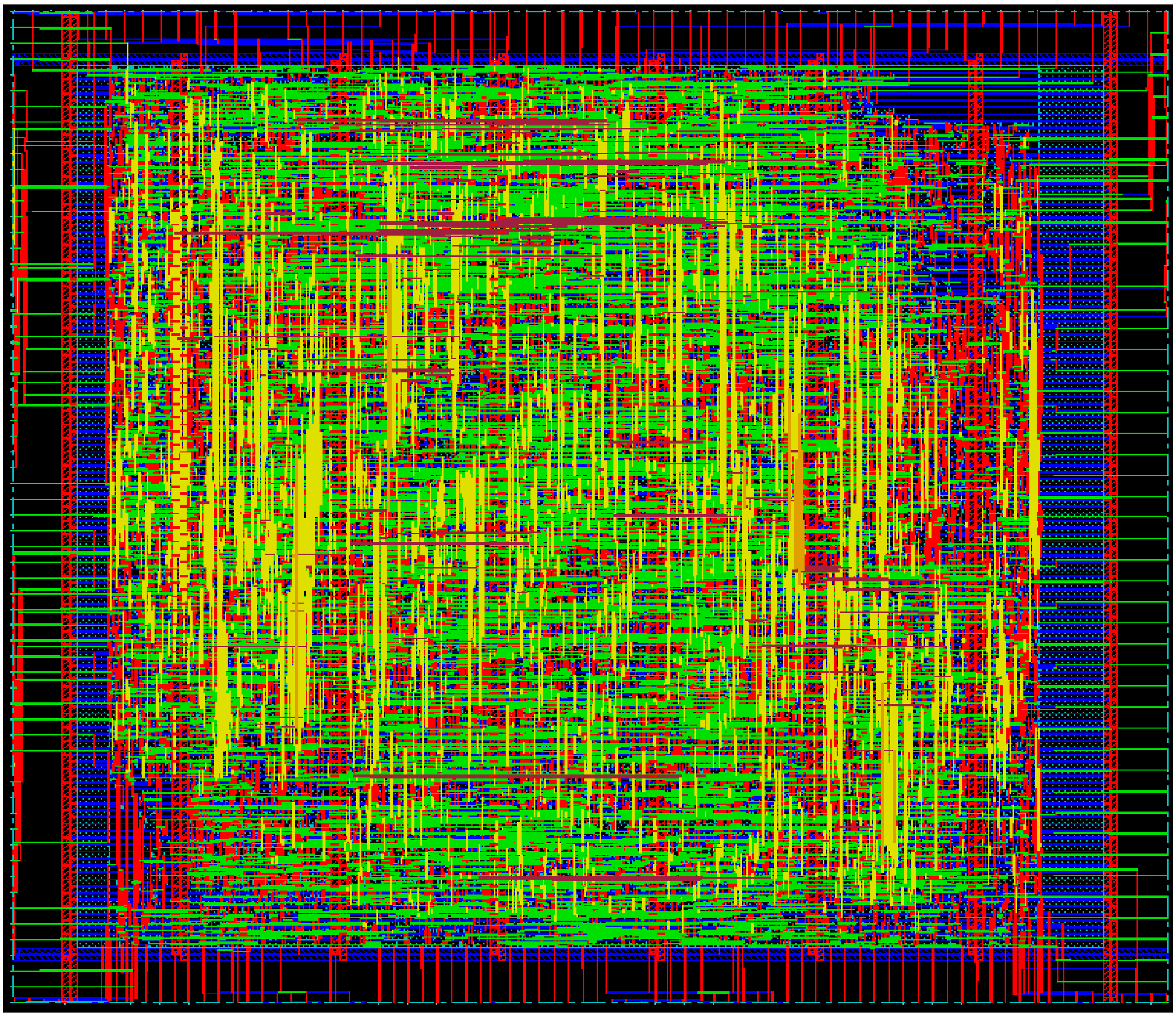


Figure 6.8. Physical layout of architecture – II

6.4 SUMMARY:

This chapter has reported multiplier-less architectures of radix-2^2 16-point 4-parallel complex FFT core. The algorithm that is considered for designing these is NEDA. Of the proposed architectures, architecture – III has shown better performance in ASIC implementation in terms of power (2.6513 mW) and slack (16.65 ns at 50 MHz) as well as in FPGA implementation in terms of operating frequency (541.947 MHz) and slice-delay product (2.196). All the architectures are designed using pipelining thus increasing the scope of their applicability. The device used for FPGA implementation is XC5VSX240T-2FF1738. The technology used for ASIC is 180 nm. The ASIC designs are compliant with their FPGA counterparts thus showing better scope in many applications.

Chapter – 7:

7.1 Conclusions:

While designing various FFT cores, due to the use of multiplexers, memory, or ROMs, there is a substantial increase in power consumption and area. In order to increase speed and throughput, folding and pipelining methods have been approached by various existing designs. But the prime disadvantage of those architectures is the use of multipliers for twiddle multiplications. This present work has proposed various multiplier-less FFT cores using NEDA technique. Both folding and pipelining techniques have also been used in the proposed designs.

By doing such, it can be concluded that there is a substantial increase in the performance of the proposed FFT cores in terms of speed, power and area as compared to other existing architectures. Radix-4, Radix – 2^2 FFT algorithms have been used in the different architectures. The architectures have been implemented in FPGA and ASIC. ASIC implementation of architectures has been done using Synopsys and Cadence tools. Three different FFT cores have been proposed and have undergone complete FPGA and ASIC flow.

We reported a DFT core, which is employed in many communication systems, using NEDA, a ROM less and multiplier less method. The proposed design is hardware efficient as compared to other traditional methods as well as architectures that are built using only DA. The design is implemented on Xilinx XC2VP30-7FF896 FPGA, which occupies an area ratio of 6% of the total configurable area. The complex DFT design in total has 106 adders with zero multipliers and no ROM, thus making possible in implementing higher point DFT. The maximum frequency of operation of the proposed design on the mapped FPGA is 122 MHz the proposed design is also compared to other designs mentioned in [4].

The present architecture of 16-point radix-4 complex FFT core using NEDA which is a ROM less and multiplier less method. The proposed design is efficient in terms of hardware as compared to other traditional methods. The proposed design has been implemented on different FPGAs to compare the performance metrics. Number of real adders/subtractors required to implement the proposed design is 269. The maximum frequency obtained is 119.27 MHz on XC5VLX330 FPGA. The throughput and slice delay product obtained in XC5VLX330 FPGA are 1828.32 Mbit/s and 9.18. In terms of power consumption, XC2V8000 FPGA shows better result than others. The proposed design occupied 21039.480469 units of cell area for 65 nm technology and 161733.406250 units of cell area for 180 nm technology, when synthesized using synopsys DC. Power reports using synopsys show that the consumption is 22.7303 mW for 65 nm technology and 302.5547 mW for 180 nm technology.

The multiplier-less architectures of radix-2^2 16-point 4-parallel complex FFT core are also proposed. The algorithm that is considered for designing these is NEDA. Of the proposed architectures, architecture – III has shown better performance in ASIC implementation in terms of power (2.6513 mW) and slack (16.65 ns at 50 MHz) as well as in FPGA implementation in terms of operating frequency (541.947 MHz) and slice-delay product (2.196). All the three architectures are designed using pipelining thus increasing the scope of their applicability. The device used for FPGA implementation is XC5VSX240T-2FF1738. The technology used for ASIC is 180 nm. The ASIC designs are compliant with their FPGA counterparts thus showing better scope in many applications.

7.2 Future Research:

The complete FPGA and ASIC flow of the proposed 512-point FFT using radix-16 algorithm is to be carried out. The multiplier-less architecture can also be used in different mathematical transforms like Discrete Cosine Transform (DCT) and Wavelet Transform.

The proposed designs can still be optimized to give better results by taking care of precise shifting in the NEDA blocks. Folding technique can also be used more efficiently in the proposed architectures. Different higher radix FFT algorithms can be combined to get better results.

Last but not the least, the multiplier-less concept along with the use of folding and pipelining can be extended to various types of computation extensive applications and might result in area, power consumption, and delay reduction.

REFERENCES:

[1] Stanley A. White, "Applications of Distributed Arithmetic to Digital Signal Processing: A Tutorial Review," *IEEE ASSP Magazine*, vol. 6, no. 3, pp. 4 – 19, Jul. 1989.

[2] Wendi Pan, Ahmed Shams, and Magdy A. Bayoumi, "NEDA: A NEw Distributed Arithmetic Architecture and its Application to One Dimensional Discrete CosineTransform," *Proc. IEEE Workshop on Signal Processing Syst.*, pp. 159–168, Oct. 1999.

[3] Richard M. Jiang, "An Area-Efficient FFT Architecture for OFDM Digital Video Broadcasting," *IEEE Trans. Consumer Elect.*, vol. 53, no. 4, pp. 1322–1326, Nov. 2007.

[4] M. Rawski, M. Vojtynski, T. Wojciechowski, and P. Majkowski, "Distributed Arithmetic Based Implementation of Fourier Transform Targeted at FPGA Architectures," *Proc. Intl. Conf. Mixed Design*, pp. 152 – 156, Jun. 2007.

[5] S. Chandrasekaran, and A. Amira, "Novel Sparse OBC based Distributed Arithmetic Architecture for Matrix Transforms," *Proc. IEEE Intl. Sym. Circuits and Syst.*, pp. 3207 – 3210, May 2007.

[6] Pramod K. Meher, Jagdish C. Patra, and M. N. S. Swamy, "High-Throughput Memory-Based Architecturefor DHT Using a New Convolutional Formulation," *IEEETrans.Circuits and Syst. – II*, vol. 54, no. 7, pp. 606 – 610, Jul. 2007.

[7] Pooja Choudhary, and Dr. Abhijit Karmakar, "CORDIC Based Implementation of Fast Fourier Transform," *Proc. Intl. Conf. Computer and Comm. Tech.*, pp. 550 – 555, Sept. 2011.

[8] Jayshankar, "Efficient Computation of the DFT of a 2N – Point Real Sequence using FFT with CORDIC based Butterflies," *Proc. IEEE TENCON 2008*, pp. 1 – 5, Nov. 2008.

[9] Ren-Xi Gong, Jiong-Quan Wei, Dan Sun, Ling-Ling Xie, Peng-Fei Shu and Xiao-Bi Meng, "FPGA Implementation of a CORDIC-based Radix-4 FFT Processor for Real-Time Harmonic Analyzer," *Intl. Conf. on Natural Computation*, pp. 1832 – 1835, Jul. 2011.

[10] Alban Ferizi, Bernhard Hoeher, Melanie Jung, Georg Fischer, and Alexander Koelpin, "Design and Implementation of a Fixed-Point Radix-4 FFT Optimized for Local Positioning in Wireless Sensor Networks," *Intl. Multi-Conf. Syst. Signals and Devices,* pp. 1 – 4, Mar. 2012.

[11] Li Wenqi, Wang Xuan, and Sun Xiangran, “Design of Fixed-Point High-Performance FFT Processor,” *Intl. Conf. Edu. Tech. and Comput.,* vol. 5, pp. 139 – 143, Jun. 2010.

[12] S.-N. Tang, J.-W. Tsai, and T.-Y. Chang, “A 2.4-GS/s FFT processor for OFDM-based WPAN applications,” *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 57, no. 6, pp. 451–455, Jun. 2010.

[13] S. Li, H. Xu, W. Fan, Y. Chen, and X. Zeng, “A 128/256-point pipeline FFT/IFFT processor for MIMO OFDM system IEEE 802.16e,” in *Proc. IEEE Int. Symp. Circuits Syst.,2010,* pp. 1488–1491.

[14] M. Garrido, K. K. Parhi, and J. Grajal, “A pipelined FFT architecture for real-value signals,” *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 56, no. 12, pp. 2634–2643, Dec.2009.

[15] Keshab K. Parhi, “VLSI Digital Signal Processing Systems:Design and Implementation”, Wiley, 1999.

[16] J. S. Walther, “A Unified Algorithm for Elementary Functions,” *Proc.Joint Spring Comput. Conf.*, vol. 38, pp. 379 – 385, Jul. 1971.

[17] J. E. Volder, “The CORDIC trigonometric computing technique,” *IRETrans. Electron.Comput.* vol. EC-8, pp. 330– 334, Sep. 1959.

[18] J. Villalba, J. A. Hidalgo, E. L. Zapata, E. Antelo, and J. D. Bruguera, “CORDIC architectures with parallel compensation of scale factor,” *Proc. Application Specific Array Processors Conf.,* pp. 258 – 269, Jul. 1995.

[19] Zhi-Xiu Lin and An-Yeu Wu, “Mixed-Scaling_Rotation CORDIC (MSR-CORDIC) Algorithm and Architecture for Scaling-Free High-Performance Rotational Operations,”*Proc. Acoustics, Speech, and Signal Processing Conf.*, vol. 2, pp. 653 – 656, Apr. 2003.

[20] N Prasad, Ayas Kanta Swain, and K. K. Mahapatra, “Design and error analysis of a scale-free CORDIC unit with corrected scale factor*,” Proc. IEEE PrimeAsia*, pp. 7 – 12, Dec. 2012.

[21] Mario Garrido, J. Grajal, M. A. Sánchez, and Oscar Gustafsson, “Pipelined Radix-2^k Feedforward FFT Architectures,” IEEE Trans. VLSI Syst., vol. 21, no. 1, pp. 23 – 32, Jan. 2013.

[22] Manohar Ayinala, Michael Brown, and Keshab K. Parhi, “Pipelined Parallel FFT Architectures via Folding Transformation,” *IEEE Trans. VLSI Syst.*, vol. 20, no. 6, pp. 1068 – 1080, Jun. 2012.

[23] J ohn G. Proakis, Dimitris G. Manolakis, “Digital Signal Processing: Principles, Algorithms, and Applications”, Prentice- Hall,1998.

[24] S. Aggarwal, P. K. Meher, and K. Khare, “Area-Time Efficient Scaling-Free CORDIC Using Generalized Micro- Rotation Selection,” *IEEE Trans*. VLSI Syst., vol. 20, no. 8, pp. 1542 – 1546, Aug. 2012.

[25] S.-N. Tang, J.-W. Tsai, and T.-Y. Chang, “A 2.4-GS/s FFT Processor for OFDM-Based WPAN Applications,” *IEEE Trans. Circuits Syst*. II: Exp. Briefs, vol. 57, no. 6, Jun. 2010, pp. 451 – 455.

[26] Z. Ismail, N. H. Ramli, Z. Ibrahim, T. A. Majid, G. Sundaraj, and W.H. W. Badaruzzaman, “Design Wind Speeds using Fast Fourier Transform: A Case Study,” Computational Intelligence in Control,Idea Group Publishing, 2012, ch. XVII.

[27] P. Duhamel, “Implementation of split-radix FFT algorithms for complex, real, and real-symmetric data,” *IEEE Trans. Acoust., Speech, Signal Processing*, vol. ASSP-34, Apr. 1986, pp.285-295.

[28] Robert Frey, “The FFT Analyzer in Mechanical Engineering Education,” *Sound and Vibration: Instrumentation Reference Issue*, Feb. 1999, pp. 1 – 3.

[29] Y. Chen, Y. Tsao, Y. Wei, C. Lin, and C. Lee, “An indexed- scaling pipelined FFT processor for OFDM-based WPAN applications*,” IEEE Trans. Circuits Syst. II*: Exp. Briefs, vol. 55, no. 2, Feb. 2008, pp. 146–150.

[30] M. Shin and H. Lee, “A high-speed four-parallel radix-2 FFT processor for UWB applications,” *Proc. IEEE Int. Symp. Circuits Syst*. (ISCAS), 2008, pp. 960–963.

[31] F. Arguello and E. Zapata, “Constant geometry split-radix algorithms*,” Journal of VLSI Signal Processing*, 1995.

[32] James W. Cooley and John W. Tukey, “An Algorithm for Machine Calculation of Complex Fourier Series,” *Mathematics of Computation*, vol. 19, 1965, pp.297-301.

Dissemination of Work

1. **Abhishek Mankar**, N Prasad, Ansuman DiptiSankar Das, Sukadev Meher, "Multiplier-less VLSI Architectures for Radix-2^2 Folded Pipelined Complex FFT Core," *International Journal of Circuit Theory and Applications* (Communicated to IJCTA), Mar. 2013.

2. **Abhishek Mankar**, N Prasad, and Sukadev Meher, "FPGA Implementation of Discrete Fourier Transform Core Using NEDA," *Proc. IEEE CSNT 2013*, Gwalior (IND), Apr. 2013.

3. Ansuman DiptiSankar Das, **Abhishek Mankar**, N Prasad, K. K. Mahapatra, and Ayas Kanta Swain, "Efficient VLSI Architectures of Split-Radix FFT using New Distributed Arithmetic," *Int. J. Soft Computing and Eng.*, vol. 3, no. 1, pp. 264–271, Mar. 2013.