FPGA Based Implementation of Different Input Size FFT Algorithms

THESIS SUBMITTED

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS

FOR THE AWARD OF THE DEGREE OF

MASTER OF TECHNOLOGY IN VLSI DESIGN AND MICROELECTRONICS TECHNOLOGY

BY

DIKSHA HAJARI

Roll No: 001910703017

Examination Roll No-M6VLS22016

Registration No- 150144 of 2019-20

Under the guidance of

PROF. JAYDEB BHAUMIK

Department of Electronics and Telecommunication Engineering

JADAVPUR UNIVERSITY, KOLKATA-700032

WEST BENGAL, INDIA

AUGUST-2022

DECLARATION

I hereby submit this thesis for partial fulfillment of Master of Technology degree in the stream of VLSI Design and Microelectronics Technology. I also declare that this thesis contains original work and related literature survey. All information in this document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that as required by these rules and conduct I have fully cited and referenced all materials and results that are not original with this work.

Name: Diksha Hajari

Roll No: 001910703017

Examination Roll No: M6VLS22016

Registration No: 150144 of 2019-20

Project Title: FPGA Based Implementation of Different Input Size FFT Algorithms

Date:

_____

Place: Kolkata

(Student Signature)

Faculty of Engineering &
Technology Jadavpur University

CERTIFICATE OF RECOMMENDATION

This is to certify that the thesis entitled "**FPGA Based Implementation of Different Input Size FFT Algorithms**" has been carried out by **Diksha Hajari** (Class Roll No: **001910703017,** Examination Roll No**: M6VLS22016** and Registration No**: 150144 of 2019-20**) under my guidance and supervision and be accepted in partial fulfillment of the requirement for the degree of **Master of Technology in VLSI and Microelectronics Engineering**, in the Department of **Electronics and Telecommunication Engineering**, Jadavpur University, Kolkata-700032.

……

Prof. Jaydeb Bhaumik
Supervisor
Department of Electronics and
Telecommunication Engineering,
Jadavpur University,
Kolkata- 700032

……

Prof. Manotosh Biswas

Head of the Department
Department of Electronics and
Telecommunication Engineering,
Jadavpur University,
Kolkata-700032

……..

Prof. Chandan Mazumdar

Dean
Faculty Council of Engineering and
Technology,
Jadavpur University,
Kolkata-700032

Faculty of Engineering & Technology
Jadavpur University

CERTIFICATE OF APPROVAL*

The forgoing thesis, entitled "**FPGA Based Implementation of Different Input Size FFT Algorithms**" is hereby approved as a creditable study on an engineering subject carried out and presented in a manner satisfactory to warrant its acceptance as a prerequisite to the degree for which it has been submitted. It is understood that by this approval the undersigned does not necessarily endorse or accept every statement made, opinion expressed, or conclusion drawn therein but approve the thesis only for the purpose for which it has been submitted.

Committee on Final Examination for
Evaluation of the Thesis:

_____
External Examiner

_____
Prof. Jaydeb Bhaumik
Supervisor

*Only in case the thesis is approved.

Acknowledgment:

I express my deep sense of gratitude to my Thesis Supervisor, **Prof. Jaydeb Bhaumik**, Department of Electronics & Telecommunication Engineering, Jadavpur University, Kolkata, for providing me the opportunity to carry out the Thesis work. I am grateful to him for the valuable insights and suggestions that he gave me throughout my M.Tech. Thesis.

I also thank, **Prof. Manotosh Biswas**, HOD, Department of Electronics and Telecommunication Engineering.

I also like to express my sincere gratitude to Ph.D. Research Scholar **Jhilam Jana** for providing all the necessary help, encouragement, and support in this thesis work.

I would like to express my sincere gratitude to all the faculty members of ETCE Department, Jadavpur University and all other teaching and non-teaching staff of the department for providing necessary support.

Last but not the least, this work would not have been possible without the love, support and encouragement of my family members, classmates, and all of them.

August-2022

Abstract:

In digital signal processing and communication system, the Fast Fourier Transform (FFT) and its inverse (IFFT) are crucial algorithms. Various types of FFT algorithms exist in Digital Signal Processing; among them, the most basic and widely used method is Cooley-Tukey's Radix-2 FFT technique, but radix-2^2 FFT algorithm maintains the basic butterfly structure of the radix-2 technique while having the same multiplicative complexity as the radix-4 approach. Several FFT architectures have already been introduced due to their wide range of applications where real-time data is present, such as medical diagnosis and seismic monitoring, etc. This thesis introduces a radix-2^2 Single-path Delay Feedback (SDF) pipeline architecture for 64, 128, 256, 512, and 1024-point FFT processors based on a Common Factor Algorithm (CFA). The complexity of the radix-2 butterfly is very low compared to the radix-4 butterfly, but the radix-2^2 CFA uses less twiddle factor than both radix-2 and radix-4 butterfly. These architectures are implemented in the FPGA platform using the Xilinx Vivado 2019.1 synthesis tool of two different FPGA device families. The synthesis results show that these architectures are enhanced in area, delay and logic power.

List of Figures:

3.1 Example of a Function with a finite discontinuity… 11

3.2 An 8-point DFT flow graph using two $N/2$ -point DFTs… 17

3.3 An 8-point DFT Flow graph computed by combining two $N/2$-point DFTs with merged W_N coefficients……18

3.4 Flowgraph of an 8-point Radix- 2 DIT FFT… 19

3.5 Flow graph of an 8-point Radix-2 DIT FFT using W_8coefficients……..19

3.6 Butterfly diagram of a Radix-2 Decimation in Frequency (DIF)….....24

3.7 Radix-4 DIT Butterfly…. 25

3.8 Butterfly flowgraph with decomposed Twiddle Factors…. 29

3.9 For $N = 16$ Radix-2^2 DIF FFT Flow graph…. 30

4.1 RTL Diagram of 64-point Radix-2^2 FFT Processor… 32

4.2 Architecture of 64-point Radix-2^2 FFT Processor…. 33

4.3 Butterfly Structure…. 35

4.4 Trivial Multiplier…..35

4.5 Block diagram of Complex Adder…..36

4.6 Complex Multiplier…..37

4.7 RTL Diagram of 128-point Radix-2^2 FFT Processor…..37

4.8 Architecture of 128-point Radix-2^2 FFT Processor…..37

4.9 RTL Diagram of 256-point Radix-2^2 FFT Processor…..38

4.10 Architecture of 256-point Radix-2^2 FFT Processor…..39

4.11 RTL Diagram of 512-point Radix-2^2 FFT Processor…..40

4.12 Architecture of 512-point Radix-2^2 FFT Processor…..40

4.13 RTL Diagram of 1024-point Radix-2^2 FFT Processor…..41

4.14 Architecture of 1024-point Radix-2^2 FFT Processor…..41

List of Tables:

Table 5.1: FPGA synthesis result of implemented FFT architecture of different input size...44

Table 5.2: Comparison of the implemented and the existing FFT architectures on Virtex-7..45

Table 5.3: Comparison of the implemented and the existing FFT architectures on Artix-7. 46

Contents:

Declaration ..... **i**

Certificate of Recommendation ..... **ii**

Certificate of Approval ..... **iii**

Acknowledgement ..... **iv**

Abstract ..... **v**

List of Figures ..... **vi-vii**

List of Tables ..... **viii**

CHAPTER 1: Introduction… **1**

Thesis Objective…..... 2

Thesis Contribution…..... 3

CHAPTER 2: Literature Review… **4**

CHAPTER 3: Mathematical Background… **9**

3.1 The Fourier Transform…..... 10

3.2 The Continuous Fourier Transform…..... 10

3.3 The Discrete Fourier Transform…..... 12

3.4 The Fast Fourier Transform…..... 14

3.4.1 History…..... 14

3.4.2 Simple Derivation…..... 15

3.5 Common FFT Algorithm…..... 19

3.5.1 Common-Factor Algorithms…..... 21

3.5.2 Radix-r vs. Mixed Radix…..... 22

3.5.2.1 Radix-2 Decimation in Time (DIT)..... 23

3.5.2.2 Radix-2 Decimation in Frequency (DIF)…..... 24

3.5.2.3 Radix-4… 24

3.5.2.4 Radix 2^i and Higher Radix FFT Algorithms 26

3.6 Radix 2^2 DIF FFT Algorithms.....27

CHAPTER 4: Implementation of Fast Fourier Transform Architectures 31

4.1 Implementation of Radix-2^2 SDF Pipeline 64, 128, 256, 512 and 1024 point FFT Architectures 32

4.1.1 General Operation… 32

4.2 Radix-2^2 SDF Pipeline 64-point FFT Architecture 32

A) Operation.....33

B) Formation 34

4.3 Radix-2^2 SDF Pipeline 128-point FFT Architecture… 37

A) Operation.....38

B) Formation 38

4.4 Radix-2^2 SDF Pipeline 256-point FFT Architecture… 38

A) Operation….....39

B) Formation… 39

4.5 Radix-2^2 SDF Pipeline 512-point FFT Architecture… 40

A) Operation….....40

B) Formation… 41

4.6 Radix-2^2 SDF Pipeline 1024-point FFT Architecture… 41

A) Operation….....42

B) Formation… 42

CHAPTER 5: FPGA Based Synthesis Results **43**

CHAPTER 6: Conclusion & Future Scope….....**47**

Conclusion.....48

Future Scope 48

References. **49**

CHAPTER 1:

Introduction

Fast Fourier Transform (FFT) algorithms are extensively utilised in digital signal processing and communication. In many DSP applications, the FFT method is regarded as one of the fundamental algorithms. Modern mobile communications, particularly those using orthogonal frequency division multiplexing (OFDM) transceiver systems, rely heavily on FFT. An efficient multicarrier technology for high-rate, high-speed data transmission in wired and wireless communication systems, such as wireless local area networks, is orthogonal frequency division multiplexing (OFDM). On IEEE 802.11 standards, the majority of contemporary WLANs are built. Several researchers are interested in implementing FFT architectures for quick and effective computational schemes. A few design strategies for FFT include memory-based, pipeline-based, and general-purpose DSP based. The most area-efficient method is memory-based, but it requires a lot of processing time. The main features of pipeline based architecture are regularity, modularity, local connectivity, and high throughput rate with a reduced clock frequency. There are seven types of Pipelined architecture: i) Single-Path Delay Feedback (SDF), ii) Single-Path Delay Commutator (SDC), iii) Single-Stream Feedforward (SFF), iv) Serial Commutator (SC), v) Multi-Path Delay Commutator (MDC), vi) Multi-Path Delay Feedback (MDF), and vii) Multi-Path Serial Commutator (MSC) which can be used to group all hardware FFT implementations. The SDF architecture is the most appropriate among these seven pipelined architectures because this architecture requires less delay and power. The suggested pipelined SDF architecture's controller is simpler than the other structures because the entire architecture and its components are controlled by a single count signal.

Thesis Objective:

Fast Fourier Transform is a main building block in digital signal processing. Its applications vary from OFDM-based Digital Modems to ultrasound, CT image, and RADAR signal processing algorithms. It can also be used for spectral analysis, matched filtering, image processing, etc. Radix has an influence on FFT design. Since we have seen that if the radix number increases butterfly complexity also increases but twiddle factors reduce and if radix number decreases butterfly complexity also decreases but number of twiddle factor increases. The suggested FFT block is designed utilising the simplified 2-point

DFT butterfly structure and the radix-2^2 CFA, which utilises lesser twiddle factors than radix-4.

The main objectives of the thesis are: i) Design and implementation of FFT architectures based on radix-2^2 Common Factor Algorithm, ii) Implemented FFT architecture has lesser delay and consume lower logic power compared to existing architectures and iii) Implemented FFT architectures and existing architectures have been synthesized in FPGA platforms with the help of Xilinx Vivado 2019.1 synthesis tool using Artix-7, Virtex-6 and Virtex-7 FPGA device families.

Thesis Contribution

Design and implementation of different existing FFT architectures have been discussed in this thesis. Also, performance of FPGA-based implementation of these FFT architectures are presented and compared with respect to area, power consumption and delay. The complete thesis is divided into seven chapters including this introductory chapter. The organization of this thesis is as follows.

Chapter 2 presents the literature survey of the related field.

Chapter 3 describes the mathematical background of FFT including radix-2, radix-4, radix-2^2, and Common factor algorithm (CFA).

Chapter 4 provides implementation of Fast Fourier Transform (FFT) architectures based on radix-2^2 CFA.

Chapter 5 represents the FPGA implementation using Verilog HDL with the help of Xilinx Vivado 2019.1 Synthesis tool. FPGA-based synthesis results are presented in this section. Also, a comparison of the performance of different existing FFT architectures and implemented architecture in terms of different parameters such as area, delay and power consumption has been presented.

Chapter 6 concludes the thesis with some possible future work

CHAPTER 2:

Literature Survey

Literature Survey:

In the field of digital signal processing and communication, the application of FFT is particularly efficient and extensive. The Discrete Fourier Transform (DFT) is a simple algorithm that can be implemented quickly. The importance of DSP algorithms has risen dramatically in recent years. Discrete Fourier Transform (DFT) and Fast Fourier Transform (FFT) are two of the most prominent DSP approaches (FFT). DFT is widely utilised in applications like convolution, linear filtering, and so on. FFT is another efficient approach for computing DFT. In the realm of communication systems, such as audio broadcasting and digital video, FFT processor plays a significant role.

For real-time digital signal processing applications, a high-performance FFT processor is required. The split-radix 2/4, 2/8, and 2/16 FFT algorithms were proposed to minimize processing complexity. Sung et al. have proposed a reusable intellectual property (IP)-based split-radix FFT core for Orthogonal Frequency Division Multiplexing (OFDM) systems, such as Ultra-Wide Band (UWB), Asymmetric Digital Subscriber Line (ADSL) etc. The architectures are mostly built on a reusable IP 128-point split-radix FFT core based on coordinate rotation digital computer (CORDIC) [1]. FFT complex multiplications are computed using the pipelined CORDIC arithmetic unit, and the needed twiddle factors are generated using ROM-free twiddle factor generator rather than storing them in a huge ROM area [2, 3]. Yu et al. proposed the FFT technique based on the node's rotation factor and data address [4]. They use the Verilog HDL platform for data address design and realisation, as well as PLD software for compilation. They came to the conclusion that FFT with FPGA is simple to expand and has real-time data capability. Zhou et al [5] have proposed pipeline FFT architectures that are optimized for FPGA. In this paper, they have implemented radix-4 SDC and radix-2^2 SDF pipeline FFT processors on Spartan-3, Virtex-4, and Virtex-E FPGAs. On the Spartan-3, 16-bit 1024-point FFT with the radix-2^2 SDF architecture needed 2802 slices and a maximum clock frequency of 95.2 MHz. They have also presented that on Spartan-3, radix-4 SDC used 4409 slices and operated at 123.8 MHz. The numbers changed to 235.6 MHz and 2256 slices for radix-2^2 SDF on the Virtex-4 device and 219.2 MHz and 3064 slices for radix-4 SDC, respectively. Thus, they have shown that 1024-point radix-2^2 SDF architecture is better than the radix-4 SDC architecture in terms of throughput per area and measure of efficiency. The discrete Fourier transform is computed using FFT, which is a fast technique

(DFT). The operation requires a large number of N^2 complex multiplications and $N(N-1)$ additions and has a high processing requirement. This makes computation and implementation extremely challenging. Higher length FFT can be obtained from short length structures. Instead of employing 8-point FFTs to generate N-point FFTs, 4-point FFTs were used to obtain VLSI structure. Chandra et al. have proposed an architecture which is a higher order FFT that is divided into three phases, each of which is a radix-2 based 4-point FFT to reduce the number of operations [6]. To limit the number of complex multiplications after each 4-point FFT and retain the pipeline style of design computations, each stage in this architecture uses 8 complex additions/subtractions. The suggested architecture's performance is measured in terms of relative error. In terms of speed, the proposed architecture is the perfect compromise.

A 512-point feedforward FFT design for wireless personal area networks has been presented by Ahmed al. The architecture handles an 8-sample continuous flow in parallel, resulting in a throughput of 2.64 Giga-Samples/s [7]. The FFT is calculated using radix-8 butterflies in three phases. In comparison to prior systems based on radix-2, this radix greatly reduces the number of rotators. Furthermore, the proposed design consumes the bare minimum of memory for a 512-point 8-parallel FFT. Experiments reveal that, in addition to great throughput, the design is also efficient in terms of area and power consumption. Naidu et al. have proposed a 64-point, 128-point, and 256-point radix-8 highly pipelined FFT architecture [8]. A radix-8 FFT is used in the processor by USFFT64 [8]. It breaks DFT into two smaller 8-length DFTs. On the FPGA chip XC4SX25-12, the proposed FFT is implemented in Xilinx ISE 14.7 using Verilog coding. In terms of device utilisation and RTL schematic, power analysis for 64-point, 128-point, and 256-point FFTs, the simulation results are obtained. Verilog coding of the FFT implementation on Vivado has been presented by Sodhro et al [9]. FFT offers an advantage over DFT since it requires less computation. The butterfly diagram was used to create the FFT of input signals. FFT takes inputs and produces an output by adding and multiplying them. For a larger number of points, DFT must do O (N^2) addition and multiplication. In digital signal processing, particularly in the domain of digital signal and image processing, the FFT is a valuable and efficient tool. It's also been utilised to extract required information from signals in new technologies like the internet of medical things. Due to the twiddle factor, the Verilog version of FFT includes complex number addition and multiplication. It

is a quick and efficient method that involves fewer computations and produces results quickly. To reduce the multiplication complexity Bansal et al. proposed a fast FFT architecture for 128 point utilising radix-2^2 CFA [10]. Utilizing radix-2^2 CFA, swapping technique, reusing the twiddle bank, and bypassing the W_N^0 twiddle factor in place of multiplication, it improves performance by employing 41% less twiddle factors and multiplications than the standard design. They have implemented the suggested design by created the VHDL code and synthesised using the Xilinx FPGA device xc7vx330t-3ffg1761. ModelSim PE Student Edition $10.4a$ is used to simulate VHDL code that is intended to be synthesised into a Xilinx Virtex-7 FPGA. Nakhate et al. introduced the fast performance reconfigurable pipeline variable points FFT processor design [11]. The variable N points in this design can have values of $8, 16, 32, 64, 128, 256, 512$ sample points. Therefore, the proposed reconfigurable design can be used rather than using distinct designs for different points of OFDM applications. The radix-2^2 Common Factor Algorithm (CFA) and SDF architecture are used in the suggested implementation design. Compared to radix-4 and radix-2, the radix-2^2 CFA uses the relatively simple radix-2 butterfly structure, which minimizes the number of twiddle factors. Compared to other architectures, SDF architecture consumes less memory and fully utilizes multipliers. The suggested design is successfully synthesized using Xilinx ISE 14.1 synthesis tool and the results are then validated using ModelSim and a MATLAB simulation tool. The application of FFT in MIMO-OFDM (multiple input multiple output orthogonal frequency division multiplexing) systems is presented by Sridhanya et al. [12]. For each stage, they employed radix Ns (number of data streams) butterflies. Memory scheduling and a multipath delay commutator are employed as hardware implementations, reducing storage and maximising power savings. They replaced the twiddle factor with a complex multiplier, resulting in significant energy savings. FFT is an efficient approach for computing the DFT, which is commonly used in DSP and communication. The butterfly module's complex multiplication and addition units absorb the majority of the hardware resources in FFT. Multiplication, in comparison to addition, is a more difficult process due to this reason Liu et al. have proposed an approximate FFT processor where the multiplier in the complex multiplication unit of the FFT is approximated using the approximate pipeline radix-4 SDF FFT (FFTcmplx2, FFT-cmplx4) and approximate parallel FFT (AFFTcmplx1, AFFT-cmplx2, AFFT-cmplx) with different approximate levels [13]. Joseph et al. introduced the FPGA

implementation of radix-2 FFT processor based on radix-4 CORDIC. For generating twiddle factors this is used in radix-2 FFT algorithm. These twiddle factors are required for generating the FFT [21]. To reduce hardware complexity, four radix-4 Booth multipliers with varying approximation levels have been proposed. The pipeline FFT and parallel FFT based on the proposed approximation multipliers have been implemented. Performance of proposed and existing designs has been analysed in terms of slices, slice flip-flops, delay and power. As a result, the proposed FFT designs can be implemented in low-power DSP systems.

CHAPTER 3:

Mathematical Background

3.1 Fourier Transform

This chapter begins with an introduction to the Fourier transform in its two most common forms: continuous Fourier transform and Discrete Fourier Transform. There are numerous excellent texts on the Fourier transform and its properties (Bracewell, 1986 [14]; Oppenheim and Schafer, 1989 [15]; Roberts and Mullis, 1987 [16]; Strum and Kirk, 1989 [17]; DeFatta et al., 1988 [18]; Jackson, 1986 [19]) that can be consulted for more information. The remainder of the chapter introduces a set of algorithms used to efficiently compute the DFT; these algorithms are known as Fast Fourier Transform (FFT) algorithms.

3.2 Continuous Fourier Transform

Equation 3.1 defines the continuous Fourier transform. It operates on the function $f(x)$ to produce $F(s)$, also known as the Fourier transform of $f(x)$.

$$F(s) = \int_{-\infty}^{\infty} f(x) e^{-i2\pi xs} dx \tag{3.1}$$

The imaginary quantity $\sqrt{-1}$ is represented by the constant i. The independent variables x and s are real-valued and have values ranging from $-\infty$ to ∞. The independent variable of f is frequently used to represent time, it is commonly referred to as the "time" variable and denoted t. This title may be misleading because f can be a function of a variable with arbitrary units and is, in fact, commonly used with distance units. Similarly, the independent variable of F is frequently denoted as f and referred to as the "frequency" variable; its units are the inverse of 's units.

The functions $f(x)$ and $F(s)$, are complex-valued in general.

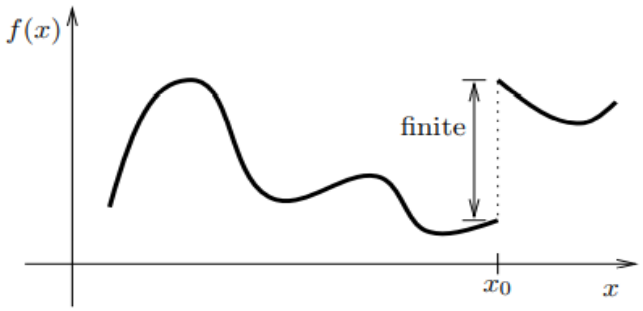

Fig 3.1 Example of a function with a finite discontinuity

Fourier integral is defined in this way, not every function $f(x)$ has a transform $F(s)$. While there are no known necessary and sufficient conditions for convergence, there are two conditions that are sufficient for convergence (Bracewell, 1986) are:

Condition 1:

There is an integral of $|f(x)|$ from $-\infty$ to ∞, i.e.,

$$\int_{-\infty}^{\infty} |f(x)|dx < \infty \qquad (3.2)$$

Condition 2:

Discontinuities in $f(x)$ are bounded. The function shown in Fig. 3.1, for example, is discontinuous at $x = x_0$, but only over a finite distance, so it meets the second condition.

Because these conditions are only necessary, many functions that do not meet these requirements have Fourier transforms. This category includes useful functions such as $\sin(x)$, the Heaviside step function $H(x)$, $\sin c(x)$, and the "generalised" Dirac delta "function"(x). The inverse Fourier transform, which converts $F(s)$ back into $f(x)$, is the counterpart to equation 3.1 and is defined as,

$$f(x) = \int_{-\infty}^{\infty} F(s)e^{i2\pi xs}ds \qquad (3.3)$$

While the continuous Fourier transform's generality is elegant and useful for studying Fourier transform theory, it has limited direct practical application. The reason for the limited practical use is that any signal $f(x)$ that exists over a finite interval has a spectrum $F(s)$ that extends to infinity, and any spectrum with finite bandwidth has an infinite-duration $f(x)$. Furthermore, the functions

$f(x)$ and $F(s)$ are continuous (at least stepwise) or analogue functions, which means they are defined over all values of their independent variables (except at bounded discontinuities, as described in condition 2). Because digital computers have limited memory, they cannot store or process an infinite number of data points required to describe an arbitrary, infinite function. The discrete Fourier transform is a special case of the Fourier transform that is used in practical applications. This transform is defined for finite-length $f(x)$ and $F(s)$ with a finite number of samples and is discussed in the following section.

3.3 Discreate Fourier Transform

The discrete Fourier transform is applied to an N-point sequence of numbers known as $x(n)$. This sequence can be thought of as a uniformly sampled version of the continuous function finite period $f(x)$. Equation 3.4 defines the DFT of $x(n)$ as an N -point sequence written as $X(k)$. In general, the functions $x(n)$ and $X(k)$ are complex. The integer indices n and k are real.

$$X(k) = \sum_{n=0}^{N-1} x(n) e^{-\frac{i2\pi nk}{N}} \quad k = 0,1, \ldots . N-1 \tag{3.4}$$

We can also write it by Substituting $W_N = e^{-i2\pi/N}$

$$X(k) = \sum_{n=0}^{N-1} x(n) W_N^{nk} \quad k = 0,1, \ldots N-1 \tag{3.5}$$

$$W_N = e^{-i2\pi/N} \tag{3.6}$$

$$= \cos(\frac{2\pi}{N}) - i \sin(\frac{2\pi}{N}) \tag{3.7}$$

Because $(W_N)^N = e^{-i2\pi} = 1$, the variable W_N is often referred to as a "N^{th} root of unity."

Another unique property of W_N is that it is periodic; that is, for any integer m, ${W_N}^n = {W_N}^{n+mN}$. This relationship ${W_N}^{n+mN} = {W_N}^n {W_N}^{mN}$ is use to express the periodicity because,

$${W_N}^{mN} = (e^{-i2\pi/N})^{mN}, \; m = -\infty, \ldots, -1,0,1, \ldots, \infty \tag{3.8}$$

$$= e^{-i2\pi m} \tag{3.9}$$

$$= 1 \tag{3.10}$$

In a manner similar to the inverse continuous Fourier transform, the Inverse DFT (IDFT), which transforms the sequence $X(k)$ back into $x(n)$, is,

$$x(n) = \frac{1}{N}\sum_{k=0}^{N-1} X(k)e^{\frac{i2\pi nk}{N}}, \qquad n = 0,1,..,N-1 \tag{3.11}$$

$$= \frac{1}{N}\sum_{k=0}^{N-1} X(k)W_N^{-nk}, \qquad n = 0,1,..,N-1 \tag{3.12}$$

from equations (3.4) and (3.11), $x(n)$ and $X(k)$ are explicitly defined over only the finite interval from 0 to $N-1$. However, because $x(n)$ and $X(k)$ are periodic in N (i.e., $x(n) = x(n+mN)$ for any integer m) and $X(k) = X(k+mN)$ for any integer m), they exist for all n and k. The number of operations required to compute the DFT of a sequence is an important feature of the DFT. According to equation 2.5, each of the DFT's N outputs is the sum of N terms consisting of $x(n){W_N}^{nk}$ products. When the term ${W_N}^{nk}$ is assumed to be a pre-computed constant, the DFT requires $N(N-1)$ complex additions and N^2 complex multiplications. As a result, calculating the DFT of a length- N sequence requires approximately $2N^2$ or $O(N^2)$ operations.

The IDFT is assumed to require the same amount of computation as its forward counterpart in this analysis because it differs only by a multiplication of the constant $1/N$ and a minus sign in the exponent of e. By modifying the pre-computed W_N term, the negative exponent can be handled without any additional computation. Another important feature of DFT algorithms is the amount of memory required for computation. Each term of the input sequence must be preserved until the final output term is computed using equation (3.5). As a result, at the very least, $2N$ memory locations are required for the direct calculation of the DFT.

3.4 Fast Fourier Transform

The Fast Fourier Transform is a group of efficient DFT computation techniques. It always yields the same outcomes as the computation of the direct form of the DFT using equation 3.4. The phrase “Fast Fourier Transform” was developed to describe the rapid DFT algorithm made famous by Cooley and Tukey's seminal article (1965). Almost every DFT was computed using an ON^2 technique previous to the release of that work. Because of this new efficient family of algorithms, the DFT’s popularity rapidly increased after the report was published.

3.4.1 History

Although Cooley and Tukey are widely credited with discovering the FFT, they actually “rediscovered” it. Some earlier known discoverers are mentioned by Cooley, Lewis, and Welch (1967). They cite Danielson and Lanczos’ (1942) paper, which describes a type of FFT algorithm and its application to X-ray scattering experiments. Danielson and Lanczos cite two papers by Runge in their article (1903; 1905). Runge and Konig’s (1924) papers and lecture notes describe two methods for reducing the number of operations required to calculate a DFT: one exploits symmetry, and the other uses the periodicity of the DFT kernel $e^{i\theta}$. Symmetry in this context refers to the property $R\{e^{i\theta}\} = R\{e^{-i\theta}\}$ and $J\{e^{i\theta}\} = -J\{e^{-i\theta}\}$; or simply $e^{i\theta} = (e^{-i\theta})^*$, where x^* is the complex conjugate of x. By utilizing symmetry, the DFT can be computed more efficiently than a direct process, but only by a constant factor; the algorithm remains $O(N^2)$. Runge and Konig briefly discuss a method for further reducing computational requirements by utilising e^{i}’s periodicity. The periodicity of $e^{i\theta}$ is demonstrated by noting that $e^{i\theta} = e^{i\theta+2\pi m}$, where m can be any integer. Taking advantage of the DFT kernel’s periodicity allows for much larger gains in efficiency, such that the complexity can be reduced to less than $O(N^2)$, as shown in Sec. 2.3.2. Because the transforms used by Runge and Konig were relatively short (all computation was done by hand), both methods resulted in comparable reductions in computational complexity, and Runge and Konig actually emphasised the method exploiting symmetry. Heideman et al. (1984), published a paper fifteen years after Cooley and Tukey’s paper, providing even more insight into the history of the FFT, including work dating back to Gauss (1866). Gauss’ work is thought to have been completed in October or November of 1805, two years before Fourier’s seminal work.

3.4.2 Simple Derivation

This section introduces the FFT by deriving one of its simplest forms. First, the DFT, equation 2.5, and the definition of W_N, equation 2.7, are repeated below for convenience.

$$X(k) = \sum_{n=0}^{N-1} x(n) W_N^{nk}, \qquad k = 0,1,..,N-1 \tag{3.13}$$

$$W_N = e^{-i2\pi/N} \tag{3.14}$$

In this example, N is chosen to be a power of two, i.e., $N = 2^m$, where m is a positive integer. The length N is thus an even number, and $x(n)$ can be divided into two sequences of length $N/2$, one containing the even members of x and the other containing the odd members. Equation 3.13 is divided into even-indexed and odd-indexed summations.

$$X(k) = \sum_{n_{even}=0}^{N-2} x(n) W_N^{nk} + \sum_{n_{odd}=1}^{N-1} x(n) W_N^{nk} \tag{3.15}$$

The result of substituting 2^m for n in the even-indexed summation and 2^{m+1} for n in the odd-indexed summation (with $m = 0,1,..,\frac{N}{2} - 1$) is,

$$X(k) = \sum_{m=0}^{N/2-1} x(2m) W_N^{2mk} + \sum_{m=0}^{N/2-1} x(2m+1) W_N^{(2m+1)k} \tag{3.16}$$

$$= \sum_{m=0}^{N/2-1} x(2m) (W_N^2)^{mk} + \sum_{m=0}^{N/2-1} x(2m+1) (W_N^2)^{mk} W_N^k \tag{3.17}$$

From equation (3.14) we can write,

$$W_N^2 = (e^{-i2\pi/N})^2 \tag{3.18}$$

$$= e^{-i2\pi 2/N}$$

$$= e^{-i2\pi/(N/2)}$$

$$= W_{N/2} \tag{3.19}$$

$$X(k) = \sum_{m=0}^{N/2-1} x(2m)W_{N/2}^{mk} + W_N^k \sum_{m=0}^{N/2-1} x(2m+1)W_{N/2}^{mk} \tag{3.20}$$

$$X(k) = \sum_{m=0}^{N/2-1} x_{even}(m)W_{N/2}^{mk} + W_N^k \sum_{m=0}^{N/2-1} x_{odd}(m)W_{N/2}^{mk} \tag{3.21}$$

$$k = 0,1,\ldots\ldots,N-1$$

where $x_{even}(m)$ is a sequence of even-indexed members of $x(n)$ and $x_{odd}(m)$ is a sequence of odd-indexed members of $x(n)$. The terms on the right are now understood to be the $(N/2)$-point DFTs of $x_{even}(m)$ and $x_{odd}(m)$.

$$x(k) = DFT_{N/2}\{x_{even}(m), k\} + W_N^k . DFT_{N/2}\{x_{even}(m), k\} \tag{3.22}$$

At this point, using only equations 3.22 and 3.13, no significant computational savings have been realised. Each $X(k)$ term still requires $2.O\left(\frac{N}{2}\right) = O(N)$ operations, which means that all N terms still require $O(N^2)$ operations. However, as stated in Section 3.2, the DFT of a sequence has a periodic length ($N/2$ in this case), which means that the $(N/2)$-point DFTs of $x_{even}(m)$ and $x_{odd}(m)$must be calculated for only $N/2$ of the N values of k. To put it another way, the $(N/2)$-point DFTs are computed for $k = 0,1,\ldots\ldots,N/2-1$ and then "re-used" for$k = \frac{N}{2}, \frac{N}{2}+1,\ldots\ldots,N-1$. The N terms of $X(k)$ can be determined using $O\left(\left(\frac{N}{2}\right)^2\right) + O\left(\left(\frac{N}{2}\right)^2\right) = O\left(\frac{N^2}{2}\right)$ operations, as well as $O(n)$ operations for multiplication by the W_N^k terms, which are referred to as "twiddle factors". When applied to large N, this $O(\frac{N^2}{2}+N)$ algorithm saves nearly half the number of operations.

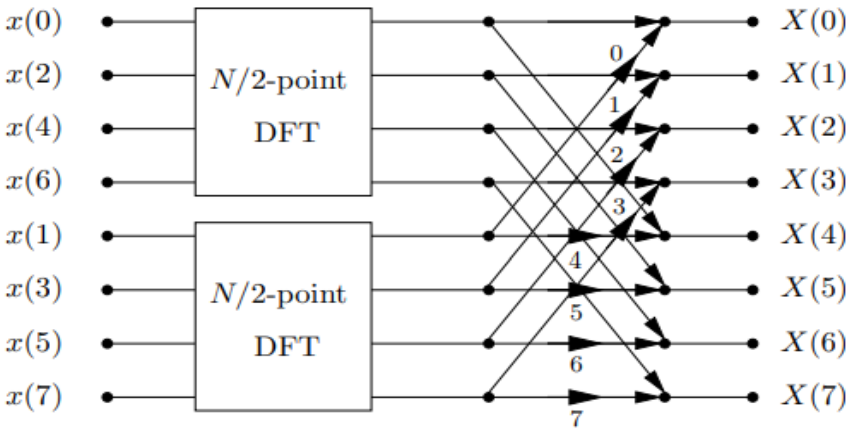

Fig 3.2 (An 8-point DFT flow graph was calculated from two $N/2$ -point DFTs. Integers next to large arrowheads indicate that the corresponding signal has been multiplied by W_8^k, where k is the given integer. Signals that follow arrows that lead to a dot are summed into that node.)

The dataflow of this algorithm for $N = 8$ is depicted graphically in Figure 3.2. Memory locations are represented by the vertical axis. The N -element sequences $x(n)$ and $X(k)$ have N memory locations. Computation stages are represented by the horizontal axis. Data Processing begins on the left with the input sequence $x(n)$ and proceeds from left to right until the output $X(k)$ is realised on the right.

The following relationship can be used to reduce the N multiplications by $W_N^k, k = 0,1, \ldots, N-1$

$$W_N^{x+\frac{N}{2}} = W_N^x W_N^{\frac{N}{2}} \tag{3.23}$$

$$= W_N^x (e^{-i2\pi/N})^{N/2} \tag{3.24}$$

$$= W_N^x e^{-i2\pi N/2N} \tag{3.25}$$

$$= W_N^x e^{-i\pi} \tag{3.26}$$

$$= -W_N^x \tag{3.27}$$

In the context of the example shown in Fig. 3.2 where $N = 8$, equation 3.27 reduces to $W_8^{x+4} = -W_8^x$ and allows the dataflow diagram of Fig. 3.2 to be transformed into the one shown in Fig. 3.3. It is worth noting that the calculations following the W_N multiplications are now 2-point DFTs.

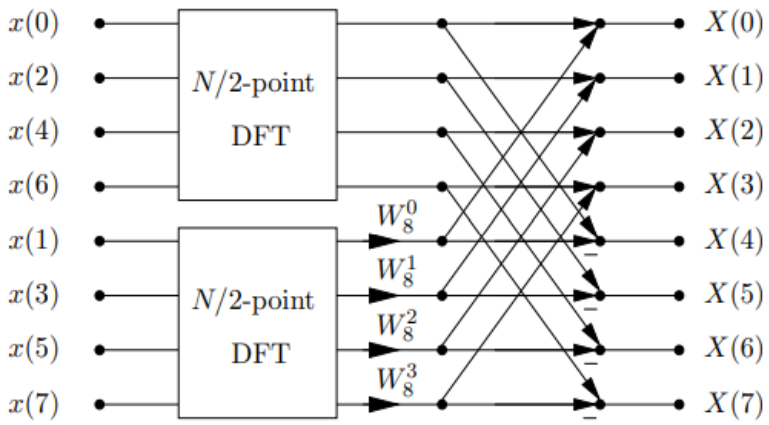

Fig 3.3 (Flow graph of an 8-point DFT computed by combining two $N/2$-point DFTs with merged W_N coefficients. A minus sign $(-)$ next to an arrow indicates that the signal is subtracted from the node rather than added to it.)

N was chosen as a power of two, if $N > 2$, both $x_{even}(m)$ and $x_{odd}(m)$ have an even number of members. As a result, they can be separated into sequences of even and odd members and computed using $\frac{N}{2}/2 = N/4$ -point DFTs. This procedure can be repeated until an even and odd separation results in sequences with two members. No further separation is required because the DFT of a 2-point sequence is trivial to compute, as will be demonstrated shortly. Figure 3.4 depicts the dataflow diagram for the $N = 8$ example. The recursive interleaving of the even and odd inputs to each DFT has scrambled the input order.

This separation process can be applied $\log_2(N) - 1$ times to yield $\log_2(N)$ stages. The resulting mth stage has $\frac{N}{2^{m+1}} . 2^m = N/2$ complex multiplications by some power of W (for $m = 0,1, \ldots, \log_2(N) - 1$). The final stage is simplified to 2-point DFTs. These are very simple to compute because, according to equation 3.13, the DFT of the 2-point sequence $x(n) = \{x(0), x(1)\}$ requires no multiplications and is calculated as follows:

$$X(0) = x(0) + x(1) \tag{3.28}$$

$$X(1) = x(0) - x(1) \tag{3.29}$$

Each of the $N/2$ 2-point DFTs requires one addition and one subtraction, and there are $N/2$ W_N multiplications per stage, each stage is calculated with roughly $2.5N$ or $O(N)$ complex operations. Summing things up, the calculation of this N-point FFT requires $O(N)$ operations for each of its $\log_2(N)$ stages, so the total computation required is $O(N)\log_2(N)$ operations.

All W coefficients are normally converted into equivalent W_N values to reduce the total number of W coefficients required. $W_4^0 = W_2^0 = W_8^0$ for the Fig. 3.4 diagram, and $W_4^1 = W_8^2$ for the Fig. 3.5 common dataflow diagram.

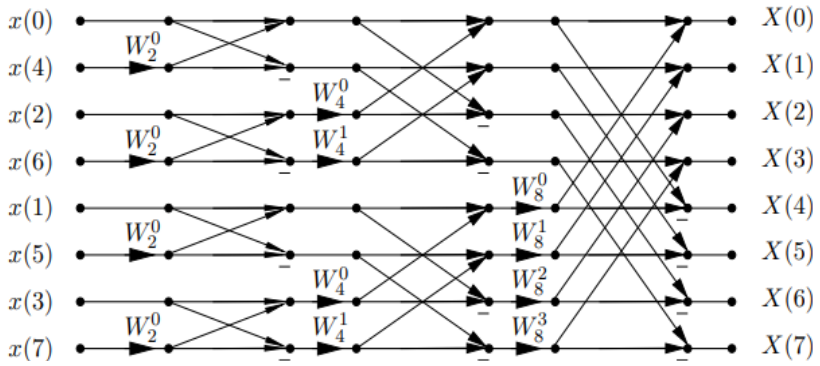

Fig-3.4 Flowgraph of an 8-point radix- 2 DIT FFT

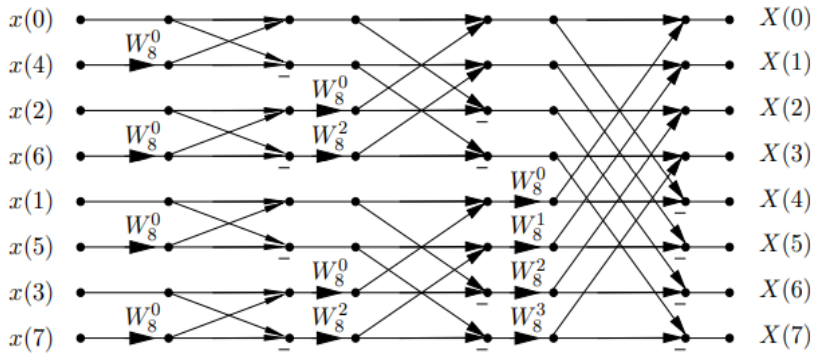

Fig 3.5 Flow graph of an 8-point radix-2 DIT FFT using W_8 coefficients

3.5 Common FFT Algorithm

A couple of the more common FFT algorithms are reviewed in this section. All methods start by selecting a highly composite value for N, or a number with numerous variables. The N' -element input sequence can be represented as a l-dimensional array if a specific N' with l factors is chosen. In some cases, it is possible to calculate the N'-point FFT by executing DFTs in each of the l dimensions and multiplying intermediate terms by the proper twiddle factors in between sets of DFTs.

The 1-dimensional input sequence x can be mapped in a number of different ways to a 2-dimensional array (Burrus, 1977; Van Loan, 1992). In the sequence x has N elements and they can be written as follows:

$$x(n) = [x(0), x(1), \ldots\ldots, x(N-1) \tag{3.30}$$

The given N is composite, it may be factored into N_1 and N_2, respectively.

$$N = N_1 N_2 \tag{3.31}$$

Using a one-to-one mapping, the inputs can be rearranged into a $N_1 N_2$ array that we refer as $\hat{x}$. Using n_1 and n_2 as the indexes in the dimensions of length N_1 and N_2, respectively, $\hat{x}$ can be expressed as follows:

$$\hat{x}(n_1 n_2) = \begin{bmatrix} \hat{x}(0,0) & \hat{x}(0,1) & \cdots & \hat{x}(0, N_2-1) \\ \hat{x}(1,0) & \hat{x}(1,1) & & \\ \vdots & & \ddots & \\ \hat{x}(N_1-1,0) & & & \hat{x}(N_1-1, N_2-1) \end{bmatrix} \tag{3.32}$$

Where $n_1 = 0,1, \ldots\ldots, N_1 - 1$, $n_2 = 0,1, \ldots\ldots, N_2 - 1$, and

$$x(n) = \hat{x}(n_1, n_2) \tag{3.33}$$

One of most popular methods for converting from 1- and 2-dimensional $\hat{x}$ is (Burrus, 1977),

$$n = An_1 + Bn_2 \bmod N \tag{3.34}$$

for mapping the x(n) inputs into the array $\hat{x}(n_1, n_2)$, and

$$k = Ck_1 + Dk_2 \bmod N \tag{3.35}$$

for the DFT output mapping $X(k)$ into the array $\hat{X}(k_1, k_2)$, where $\hat{X}$ represents the 2-dimensional map of $X(k)$. When Eqs. (3.33-3.35) are substituted into Eq. 3.13, the result is

$$\hat{X}(k_1, k_2) = \sum_{n_1=0}^{N_1-1} \sum_{n_2=0}^{N_2-1} \hat{x}(n_1, n_2) W_N^{(An_1+Bn_2)(Ck_1+Dk_2)} \tag{3.36}$$

$$= \sum_{n_1=0}^{N_1-1} \sum_{n_2=0}^{N_2-1} \hat{x}(n_1, n_2) W_N^{An_1Ck_1} W_N^{An_1Dk_2} W_N^{Bn_2Ck_1} W_N^{Bn_2Dk_2} \tag{3.37}$$

N produce different FFT algorithms by varying the $\hat{X}$ and $\hat{x}$.

The remaining sections of this chapter provide an introduction to the most popular FFT kinds and a closer look at two of its key characteristics: their butterfly architectures and flow graphs

3.5.1 Common-Factor Algorithms

The common-factor FFTs are arguably the most popular class of FFT algorithms. They are also known as Cooley-Tukey FFTs because they employ mappings first popularised in a 1965 paper by Cooley and Tukey. Their name derives from the fact that N_1 and N_2 in equation 3.31 share a common factor, which means that there exists an integer other than unity that divides N_1 and N_2 evenly.

$A, B, C,$ and D of equations 3.34 and 3.35 are set to $A = N_2, B = 1, C = 1,$ and $D = N_1$ for common-factor FFTs. After that, the equations can be written as,

$$n = N_2 n_1 + n_2 \tag{3.38}$$

$$k = k_1 + N_1 k_2 \tag{3.39}$$

Then equation (3.37) becomes,

$$\hat{X}(k_1, k_2) = \sum_{n_1=0}^{N_1-1} \sum_{n_2=0}^{N_2-1} \hat{x}(n_1, n_2) W_N^{N_2 n_1 k_1} W_N^{N_2 n_1 N_1 k_2} W_N^{n_2 k_1} W_N^{n_2 N_1 k_2} \tag{3.40}$$

$W_N^{N_2 n_1 N_1 k_2} = W_N^{N n_1 k_2} = 1$ for any values of n_1 and k_2. $W_N^{N_2 n_1 k_1} = W_{N_1}^{n_1 k_1}$ and $W_N^{n_2 N_1 k_2} = W_{N_2}^{n_2 k_2}$, based on reasoning similar to that used in equation 3.19. With these changes, equation 3.40 becomes,

$$\hat{X}(k_1, k_2) = \sum_{n_1=0}^{N_1-1} \sum_{n_2=0}^{N_2-1} \hat{x}(n_1, n_2) W_{N_1}^{n_1 k_1} W_N^{n_2 k_1} W_{N_2}^{n_2 k_2} \tag{3.41}$$

$$= \sum_{n_1=0}^{N_1-1} ([\sum_{n_2=0}^{N_2-1} \hat{x}(n_1, n_2) W_{N_2}^{n_2 k_2}] W_N^{n_2 k_1}) W_{N_1}^{n_1 k_1}$$

(3.42)

The DFT can now be computed in a new way that is using reformulation of the input sequence into a 2-dimensional array:

1. Determine the N_1 N_2-point DFTs of the terms in equation 3.32.

2. Multiply the intermediate values $N_1 \times N_2 = N$ by the appropriate $W_N^{n_2 k_1}$ twiddler factors.

3. Determine the N_2 N_1-point DFTs of the intermediate terms in equation 2.32's columns.

Equation 3.43 shows the three components of this decomposition.

$$\hat{X}(k_1, k_2) = \sum_{n_1=0}^{N_1-1} ([\sum_{n_2=0}^{N_2-1} \hat{x}(n_1, n_2) W_{N_2}^{n_2 k_2}] W_N^{n_2 k_1}) W_{N_1}^{n_1 k_1}$$

(3.43)

Where

$$\sum_{n_1=0}^{N_1-1} ([\sum_{n_2=0}^{N_2-1} \hat{x}(n_1, n_2) W_{N_2}^{n_2 k_2}] W_N^{n_2 k_1}) W_{N_1}^{n_1 k_1} \qquad is \quad N_1 - point\ DFT$$

$$\sum_{n_2=0}^{N_2-1} \hat{x}(n_1, n_2)\, W_{N_2}^{n_2 k_2} \qquad is\ N_2 - point\ DFT$$

$$W_N^{n_2 k_1} \quad is\ twiddle\ factor$$

3.5.2 Radix-r vs. Mixed Radix

Radix-r algorithms are common-factor FFTs in which $N = r^k$, where k is a positive integer, and the butterflies used in each stage are the same. A radix-r FFT has $\log_r N$ stages and uses radix-r butterflies. In terms of re-mapping the

sequence into 2-dimensional arrays, the N-point sequence is first mapped into a $r \times (\frac{N}{r})$ array, and then $k-2$ decimations follow. A multi-dimensional mapping, on the other hand, stores the N input terms in a $\log_r N$-dimensional $(r \times r \times \ldots . \times r)$ array.

Mixed-radix FFTs, on the other hand, are FFTs in which the radices of the component butterflies are not all equal. In general, radix- r algorithms are preferred over mixed-radix algorithms because the structure of butterflies in radix- r designs is consistent across all stages, simplifying the design. In some cases, such as when $N = r^k$, the choice of N dictates that the FFT be mixed-radix. The following three subsections look at three different types of common-factor FFTs, which are most likely the most widely used FFT algorithms. Radix-2 decimation in time, radix-2 decimation in frequency, and radix-4 algorithms are used.

3.5.2.1 Radix-2 Decimation in Time (DIT)

If $N = 2^k$, $N_1 = \frac{N}{2}$, and $N_2 = 2$, then equation 3.38 and 3.39 become,

$$n = 2n_1 + n_2 \tag{3.44}$$

$$k = k_1 + \frac{N}{2}k_2 \tag{3.45}$$

The resulting algorithm is known as a radix-2 decimation in time FFT when applied to each of the log2 N stages. The radix-2 DIT butterfly and a sample flow graph are shown in Figures 3.6 and 3.7, respectively. Because the exact computation required for the butterfly is critical for hardware implementations, we go over it in depth here. In Figure 3.6, the radix-2 DIT butterfly's inputs are A and B, and its outputs are X and Y. W is a complex constant that can be considered pre-computed.

$$X = A + BW \tag{3.46}$$

$$Y = A - BW \tag{3.47}$$

We introduce the variable $Z = BW$ and rewrite the equations because it would almost certainly never make sense to compute the $B \times W$ term twice.

$$Z = BW \tag{3.48}$$

$$X = A + Z \tag{3.49}$$

$$Y = A - Z \tag{3.50}$$

Thus, one complex multiplication and two complex additions are required for the radix-2 DIT butterfly.

3.5.2.2 Radix-2 Decimation in Frequency (DIF)

The Decimation in Frequency (DIF) FFT is the second most popular radix-2 FFT algorithm. The name derives from the fact that the "frequency" values, $X(k)$, are decimated during each stage in one common derivation. Setting $N_1 = 2$, and $N_2 = N/2$. Equations 3.38 and 3.39 are other ways to determine their form.

$$n = \frac{N}{2n_1} + n_2 \tag{3.51}$$

$$k = k_1 + 2k_2 \tag{3.52}$$

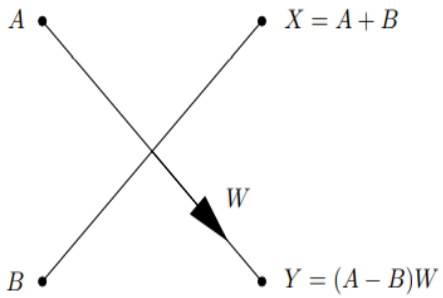

Fig 3.6 Butterfly diagram of a Radix-2 Decimation in Frequency (DIF)

A radix-2 DIF butterfly is depicted in Figure 3.9. W, is a complex constant, and A and B input. X and Y output.

$$X = A + B \tag{3.53}$$

$$Y = (A - B)W \tag{3.54}$$

The radix-2 DIF butterfly, like the DIT butterfly, necessitates one complex multiplication and two complex additions.

3.5.2.3 Radix-4

We can use a radix-4 common-factor FFT algorithm when $N = 4^k$ by recursively rearranging sequences into $N' \times N'/4$ arrays. The conception of a

radix-4 algorithm is analogous to the formation of a radix-2 FFT, and both DIT and DIF versions are possible. More information on radix-4 algorithms can be found in Rabiner and Gold (1975).

A time butterfly with radix-4 decimation is shown in Figure 3.10. The radix-4 butterfly develops similarly to the radix-2 butterfly by combining a 4-point DFT with the corresponding twiddle factors often in between DFT stages. The butterfly diagram's left side contains the four inputs, A, B, C, and D. The latter are multiplied by the complex coefficients W_b, W_c, and W_d, respectively. Since there is more than one in a single butterfly, these coefficients are all shown with distinct subscripts here to distinguish them from one another, even though they all have the same form as the W_N of Eq. 3.14.

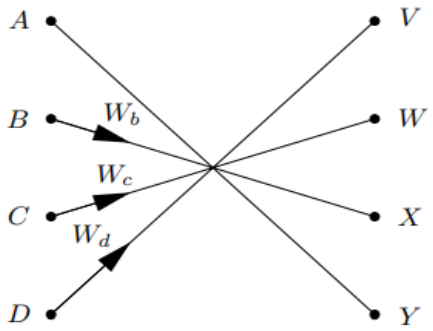

Fig 3.7 Radix-4 DIT Butterfly

V, W, X, and Y are the four outputs, which are calculated from,

$$V = A + BW_b + CW_c + DW_d \tag{3.55}$$

$$W = A - iBW_b - CW_c + iDW_d \tag{3.56}$$

$$X = A - BW_b + CW_c - DW_d \tag{3.57}$$

$$Y = A + iBW_b - CW_c - iDW_d \tag{3.58}$$

By introducing three new variables, the equations can be written more concisely.

$$B' = BW_b \tag{3.59}$$

$$C' = CW_c \tag{3.60}$$

$$D' = DW_d \tag{3.61}$$

resulting in,

$$V = A + B' + C' + D' \tag{3.62}$$

$$W = A - iB' - C' + iD' \tag{3.63}$$

$$X = A - B' + C' - D' \tag{3.64}$$

$$Y = A + iB' - C' - iD' \tag{3.65}$$

It's essential to remember that the radix-4 butterfly typically only needs three complex multiplies (Equations 3.59, 3.60, and 3.61). Multiplication by i is achieved by switching the real and imaginary components, as well as possibly negation.

One radix-4 butterfly may perform the job of four radix-2 butterflies, and the radix-4 butterfly only needs three complex multiplies as opposed to four multiplies for four radix-2 butterflies, giving radix-4 algorithms a computational advantage over radix-2 algorithms. The simple radix-4 butterfly takes 3 additions $\times$ 4 terms $= 12$ additions, but the radix-2 technique only needs 4 butterflies $\times$ 2 $= 8$ additions. However, by reusing intermediate values like $+CW_c$, $A - CW_c$, $BW_b + DW_d$, and $iBW_b - iDW_d$, a radix-4 butterfly can also be generated with eight adds. In the end, a radix-4 algorithm will need around the same amount of additions and roughly 75% as many multiplications as a radix-2 method.

3.5.2.4 Radix 2^i and Higher Radix FFT Algorithms

The multiplicative complexity of the twiddle factor can be reduced by using higher radices such as radix-8 or radix-16. However, as the radix increases, implementation complexity also increases. Torkeson et al. discussed radix- 2^2 and radix- 2^3 FFT algorithms in 1996. These algorithms perform the same number of nontrivial multiplications as the radix-4 and radix-8 algorithms. However, these algorithms differ in twiddle factors at different FFT stages while retaining the radix-2 algorithm's butterfly structure. Following these algorithms, several radix- 2^i algorithms for higher radices are developed, including radix- 2^4 , modified radix- 2^4 , radix- 2^5 , and modified radix- 2^5 algorithms. The goal of these radix-2^i algorithms is to produce more superficial

butterfly structures with lower multiplicative complexity. The following section explains how the radix-2^2 algorithm was developed, that must be extended to include higher radices.

3.6 Radix-2^2 DIF FFT Algorithm

By incorporating a twiddle factor decomposition technique into the divide and conquer approach, a hardware-oriented radix-2^2 algorithm is derived. The radix-2^2 algorithm has the same multiplicative complexity as radix-4 algorithm but retains the radix-2 algorithm's butterfly structure. To distinguish between the well-known radix-2/4 split radix algorithm and the mixed-radix-'4 + 2'algorithm, the concept of the radix-2^2 algorithm is used to reflect the structural relationship with the radix-2 algorithm and the exact computational requirement with a radix-4 algorithm.

The DFT of size N is defined as follows:

$$X(k) = \sum_{n=0}^{N-1} x(n) W_N^{nk}, \qquad 0 \le k < N \tag{3.55}$$

Where W_N is the Nth primitive root of unity, and its exponent is evaluated modulo N. Consider the first two steps of decomposition in the radix-2 DIF FFT together to better understand the new algorithm's derivation.

$$n =< \frac{N}{2} n_1 + \frac{N}{4} n_2 + n_3 > N$$

$$k =< k_1 + 2k_2 + 4k_3 > N \tag{3.56}$$

The Common Factor Algorithm (CFA) takes the form of by using a 3-dimensional linear index map.

$$X(k_1 + 2k_2 + 4k_3) =$$

$$\sum_{n_3=0}^{\frac{N}{4}-1} \sum_{n_2=0}^{1} \sum_{n_1=0}^{1} x(\frac{N}{2} n_1 + \frac{N}{4} n_2 + n_3) W_N^{\left(\frac{N}{2} n_1 + \frac{N}{4} n_2 + n_3\right)(k_1 + 2k_2 + 4k_3)}$$

$$= \sum_{n_3=0}^{\frac{N}{4}-1} \sum_{n_2=0}^{1} \{B_{\frac{N}{2}}^{k_1}(\frac{N}{4}n_2 + n_3) W_N^{\left(\frac{N}{4}n_2+n_3\right)k_1}\} W_N^{\left(\frac{N}{4}n_2+n_3\right)(2k_2+4k_3)}$$

(3.57)

Where the butterfly structure can be expressed as

$$B_{\frac{N}{2}}^{k_1}\left(\frac{N}{4}n_2 + n_3\right) = x\left(\frac{N}{4}n_2 + n_3\right) + (-1)^{k_1} + x\left(\frac{N}{4}n_2 + n_3 + \frac{N}{2}\right)$$

We can conclude the result is a normal radix-2 DIF FFT only if the expression inside the braces of equation (3.57) is calculated before further decomposition. The key concept behind the new algorithm is to apply the second step decomposition to the remaining DFT coefficients, including the "twiddle factor" $W_N^{\left(\frac{N}{4}n_2+n_3\right)k_1}$, in order to exploit the exceptional values in multiplication before constructing the next butterfly.

Analysing the composite twiddle factor and notice that,

$$W_N^{\left(\frac{N}{4}n_2+n_3\right)(k_1+2k_2+4k_3)} = W_N^{Nn_2k_3} W_N^{\frac{N}{4}n_2(k_1+2k_2)} W_N^{n_3(k_1+2k_2)} W_N^{4n_3k_3}$$

$$= (-j)^{n_2(k_1+2k_2)} W_N^{n_3(k_1+2k_2)} W_N^{4n_3k_3} \qquad (3.58)$$

Substituting equation (3.58) for equation (3.57) and expanding the summation with index n_2. After simplifying, we have a set of four DFTs with length $N/4$.

$$X(k_1 + 2k_2 + 4k_3) = \sum_{n_3=0}^{\frac{N}{4}-1} [H(k_1, k_2, n_3) W_N^{n_3(k_1+2k_2)}]\, W_{\frac{N}{4}}^{n_3k_3}$$

(3.59)

$H(k_1, k_2, n_3)$ is expressed in equation (3.60). This equation represents the first two stages of butterflies as BF I and BF II in Fig. 2 with only trivial multiplications in the SFG. Following these two stages, full multipliers are needed to compute the product of the decomposed twiddle factor $W_N^{n_3(k_1+2k_2)}$ in equation (3.59), as illustrated in Fig. 3.8. It should be noted that the order of the twiddle factors differs from that of the radix-4 algorithm.

$$H(k_1,k_2,n_3)=\left[x(n_3)+(-1)^{k_1}x\left(n_3+\frac{N}{2}\right)\right]+(-j)^{(k_1+2k_2)}\left[x\left(n_3+\frac{N}{4}\right)+(-1)^{k_1}x\left(n_3+\frac{3}{4}N\right)\right] \quad (3.60)$$

Where

$x(n_3)+(-1)^{k_1}x\left(n_3+\frac{N}{2}\right)$ and $x\left(n_3+\frac{N}{4}\right)+(-1)^{k_1}x\left(n_3+\frac{3}{4}N\right)$ is BF I

$\left[x(n_3)+(-1)^{k_1}x\left(n_3+\frac{N}{2}\right)\right]+(-j)^{(k_1+2k_2)}\left[x\left(n_3+\frac{N}{4}\right)+(-1)^{k_1}x\left(n_3+\frac{3}{4}N\right)\right]$ is BF II

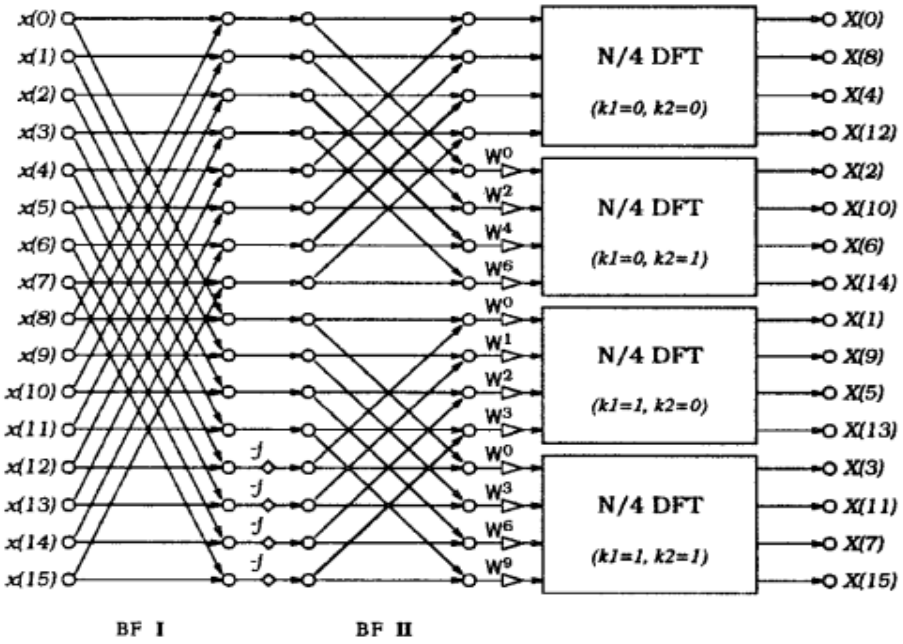

Fig 3.8 Butterfly flowgraph with decomposed Twiddle Factors

The complete radix-2^2 DIF FFT algorithm is obtained by applying this CFA procedure recursively to the remaining DFTs of length $N/4$ in equation (3.59). Fig. 3.9, represents an $N=16$ example where, small diamonds portray trivial multiplication by $W_N^{\frac{N}{4}}=-j$, which includes only real-imaginary swapping and sign inversion.

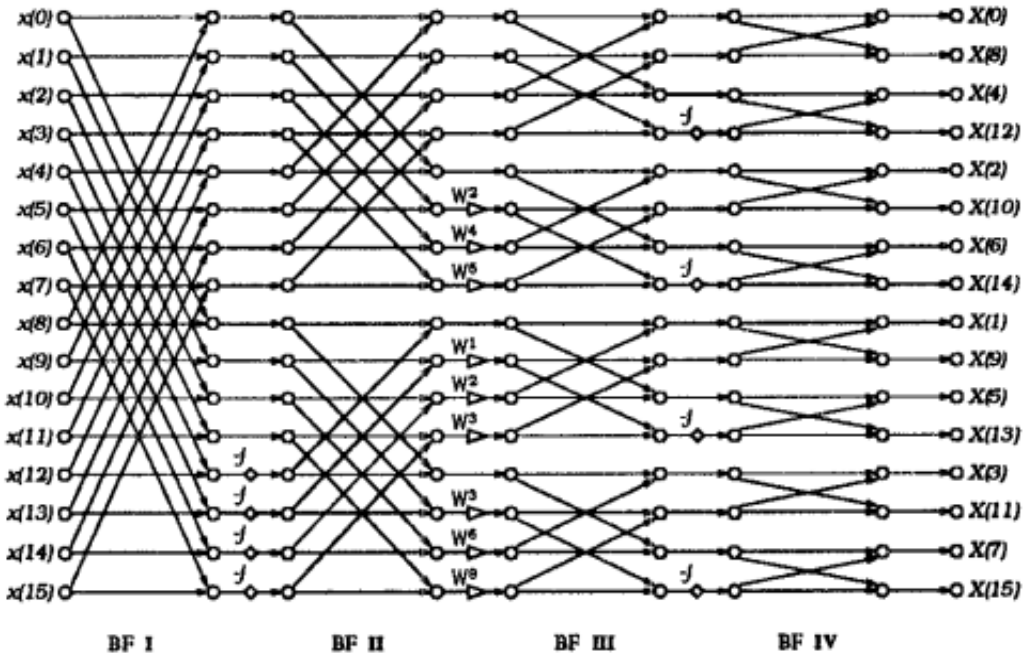

Fig 3.9 For $N = 16$ Radix-2^2 DIF FFT Flow graph

Radix- 2^2 algorithms have the same multiplicative complexity as radix-4 algorithms while retaining the radix-2 butterfly structures. Only every other stage has non-trivial multiplications due to the arrangement of the multiplicative operations. When considering pipeline/cascade FFT architecture, this is a significant structural advantage over other algorithms.

CHAPTER 4:

Implementation of FFT Architectures

4.1 Implementation of Radix-2^2SDF Pipeline 64, 128, 256, 512, and 1024 Point FFT Architectures

4.1.1 General Operation:

The implemented FFT processor for N point FFT based on radix-2^2 Common Factor Algorithm. The number of stages is $\log_2 N$. Complex number values are stored in shift registers (SR). For a set of N points, the first shift register retains $N/2$ values, $N/4$ values are stored in the second shift register, the third shift register holds $N/8$ values, and so on until the last shift register holds one value. The twiddle generator generates twiddle factors for each $W_N^{n_3(k_1+2k_2)}$. The swapper $(-j)^{n_2k_1}$ exchanges signed real and imaginary values before flipping the sign of the exchanged imaginary values. Two input values are added, subtracted, and bypassed by the butterfly structure. The multiplication between an input complex value and twiddle factor is done by a complex multiplier.

4.2 Radix-2^2 SDF Pipeline 64-Point FFT Architecture

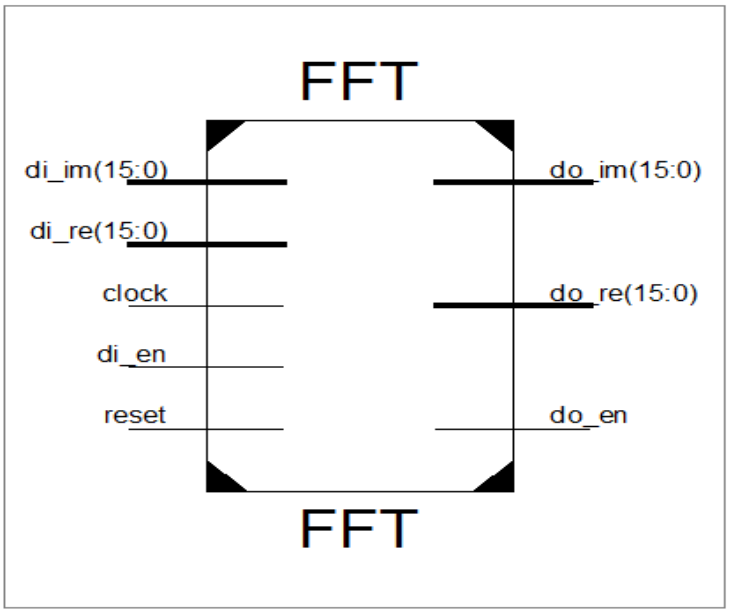

Fig 4.1 RTL diagram of 64 point FFT processor

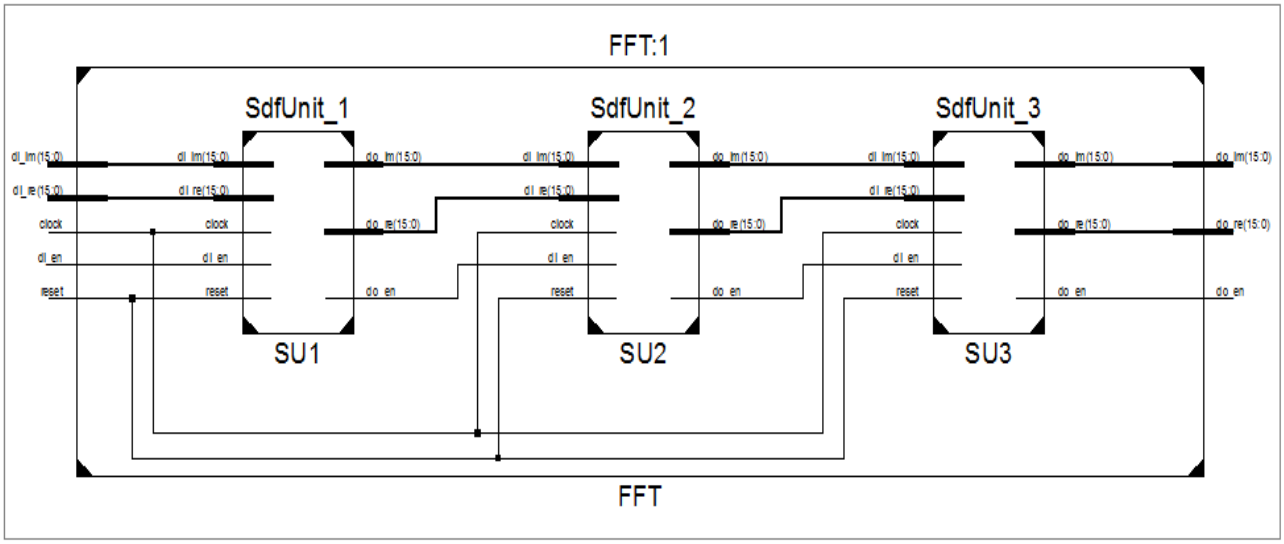

Fig 4.2 Architecture of 64 point FFT processor

A) Operation:

Figure 4.1 and 4.2 depicts the precise structure of the 64-point radix-2^2 FFT Processor, which comprises six stages Shift registers or SRs to store values. For a 64-point FFT, the first shift register stores 32 values, the second register stores 16, the third register stores 8, the fourth register stores 4, the fifth register stores 2, and the sixth shift register stores one value. Based on the values of n_3, k_1, and k_2 the twiddle generator generates the twiddle factor $W_{64}^{n_3(k_1+2k_2)}$ and $W_{16}^{n_3(k_1+2k_2)}$.. as per $W_N^{n_3(k_1+2k_2)}$. In the first stage swapping is performed as the swapper is connected to the output of the first stage. In the second stage calculations are performed as per 64-point sample using twiddle factor $W_{64}^{n_3(k_1+2k_2)}$. Again, in the third stage swapping is performed using the swapper $(-j)^{n_2k_1}$. In the fourth stage calculations are performed as per 16-point sample using twiddle factor $W_{16}^{n_3(k_1+2k_2)}$. 5th stage again perform swapping and 6th stage calculates as per 4-point sample. Addition, subtraction, and by-pass operations are performed by butterflies (BF). The complex multiplier calculates the twiddle factor values and the input complex value.

In each clock, the first 32 points of the 64-point FFT computation are stored in stage 1 register with the help of butterfly structure. It takes 32 clocks, then at the 33th clock, as x(32) is fed into stage 1, butterfly begins addition and subtraction between x(0) and x(32), the addition stored in stage 2 register and subtraction saved in stage 1 register. At the 34th clock, as x(34) entered into stage 1, the butterfly begins addition and subtraction between x(1) and x(33), addition result

stored in stage 2 register and subtraction saved in stage 1 register. This procedure is repeated until the 48th clock strikes. As soon as the x(48) point is entered into stage 1, again, the butterfly starts addition and subtraction between the x(16) and x(48) points. The addition is stored in stage 2 register, while the subtraction is stored in stage 1. At this clock, the butterfly starts addition and subtraction between the x(16) and x(48) points, and stores addition in the subsequent stage 3 register and subtraction in stage 2. This procedure run nonstop until the 63rd clock. At the 64th clock, as x(63) is fed into stage 1, the butterfly begins addition and subtraction between x(31) and x(63), addition stored in stage 2 register and subtraction stored in stage 1. At the same clock, stage 2, stage 3, stage 4, stage 5, and stage 6 butterflies begin addition and subtraction, with addition stored in stage 3, stage 4, stage 5, and stage 6 registers and subtraction stored in stage 2, stage 3, stage 4, stage 5, and stage 6 registers. The final FFT computation values are the output from stage 6.

B) Formation:

Different sub modules are required to form the FFT architecture which is as follows.

i) Butterfly Unit (BU):

The butterfly (BU) structure in Figure 4.3 operates between the nth and $(n + N/2)$th values. The Mux1, 2, 3, and 4 in the butterfly module alternate to position "0" on the first $N/2$ cycles. As soon as the shift registers are empty, the input data from the left is sent to them. The multiplexers flip to position "1" on the subsequent $N/2$ cycles; at that point, the butterfly begins addition and subtraction operations to compute the 2-point DFT using both the data stored in the shift registers and the incoming data. The results of subtraction are saved in the same register, while results of the first half no of addition are stored in the next stage register, and in the same stage, addition or subtraction operation is performed between the second half number of addition results and stored first half number. In order to compute a result per clock cycle, the butterfly element is coupled in a pipelined structure, which allows the process to continue from the current stage to the next stage and finally the final stage.

Fig 4.3 Butterfly structure

ii) Trivial Multiplier$(-j)$

Figure 4.4 shows a trivial $(-j)$ multiplier using real sign inversion and real imaginary swapping. The second compliment carries out the real sign inversion, and the multiplexor control signal 'S' directed the real imaginary swapping. The Mux1 and Mux2 multiplexors move to position "1" whenever the signal flow graph indicates that $(-j)$ multiplication is necessary. During that point, the first sign of the real value is inverted, then the inverted real value is swapped to the imaginary, and the imaginary value is switched back to the real. In order to optimize efficiency and decrease hardware logic, sign inversion and swapping are done in place of multiplication in the case of trivial $(-j)$ multiplication.

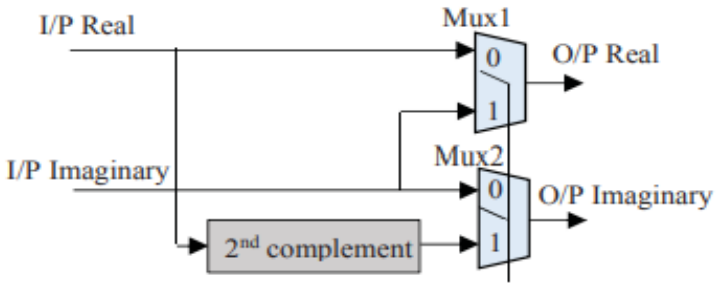

Fig 4.4 Trivial Multiplier $(-j)$

iii) Delay Buffer

Delay buffer is used to store the values. For 64-point FFT we require 6 no of delay buffer. the first delay buffer stores 32 values, the second delay buffer stores 16, the third delay buffer stores 8, the fourth delay buffer stores 4, the fifth delay buffer stores 2, and the sixth delay buffer stores one value.

iv) Complex Adder/Subtractor

Here we have used scaled parameterized, 2's complements fixed point arithmetic modules complex adder/subtractor to find the complex sum or difference of two complex values. So, using two adders or subtractors, a complex adder or subtractor can be implemented.

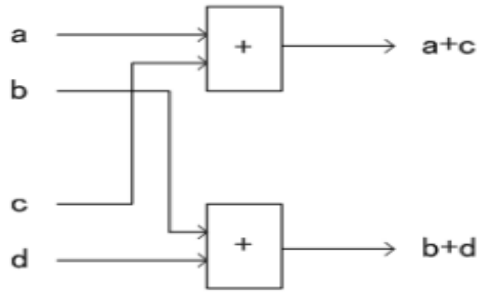

Fig 4.5 Block diagram of Complex Adder

v) Complex Multiplier:

Here we have used scaled parameterized, 2's complements fixed point arithmetic modules complex multiplier to find the complex product. So, using 4 real multipliers, 1 adder and one subtractor, a complex multiplier can be implemented.

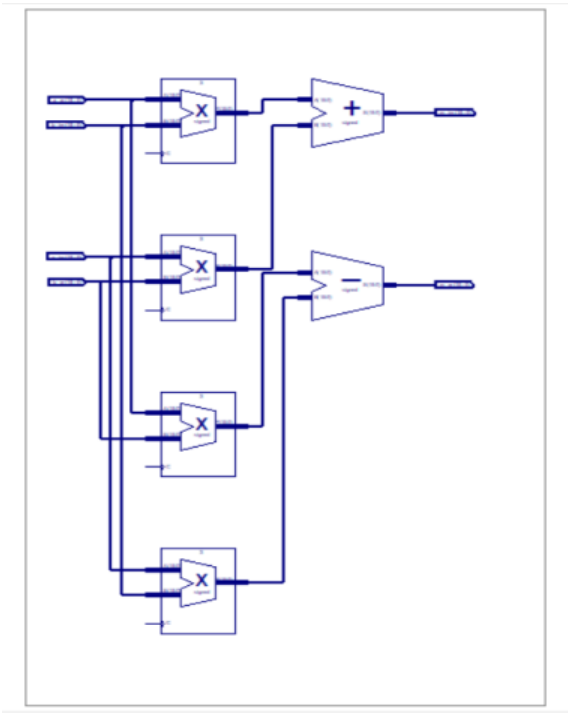

Fig 4.6 Complex Multiplier

vi) Twiddle Generator

It is used to generate twiddle factors. For 64-point FFT after 2nd stage it requires 45 no of twiddle factors and after 4th stage it requires 36 no of twiddle factors based on $W_N^{n_3(k_1+2k_2)}$.

vii) Single Path Delay Feedback Unit

SDF unit consists of delay buffer, multiplier, twiddle generator, and butterfly unit. It reduces the complexity of FFT processor.

4.3 Radix-2^2 SDF Pipeline 128-Point FFT Architecture

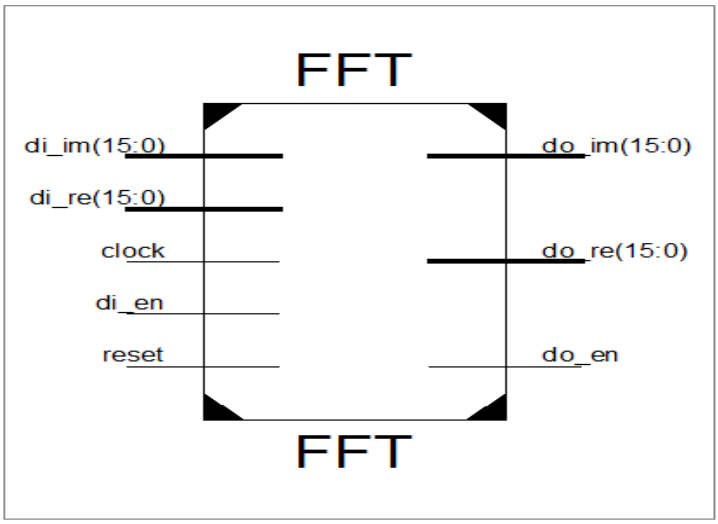

Fig 4.7 RTL diagram of 128 point FFT

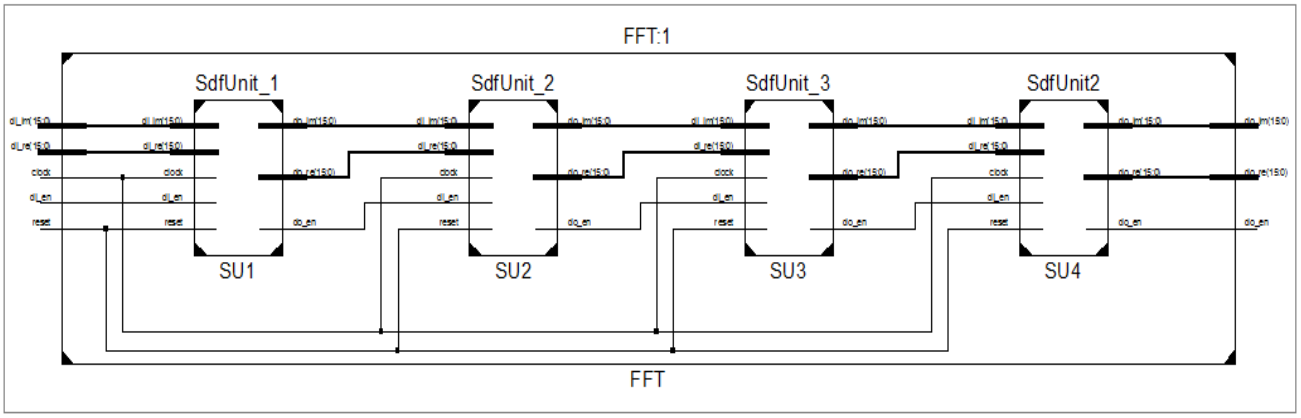

Fig 4.8 Architecture of 128 point FFT processor

A. Operation:

The operation of 128-point FFT is the same as 64-point FFT, which is described in the section no 4.2.

B. Formation:

The submodules which are required to form the 128-point FFT is the same as the 64-point FFT. The only difference is in the number of delay buffer and twiddle factor. For 128-point FFT, after 2nd stage it requires 93 number of twiddle factors, after 4th stage it requires 84 number of twiddle and after 6th stage it requires 48 number of twiddle factors based on $W_N^{n_3(k_1+2k_2)}$. The number of delay buffer it requires is 7.

4.4 Radix-2^2 SDF Pipeline 256-Point FFT Architecture

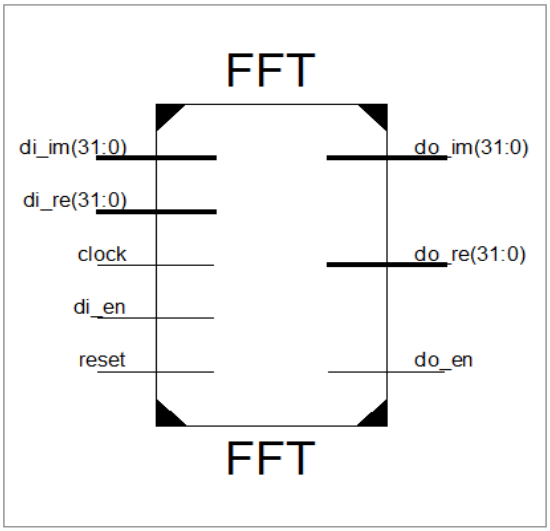

Fig 4.9 RTL diagram of 256-point FFT processor

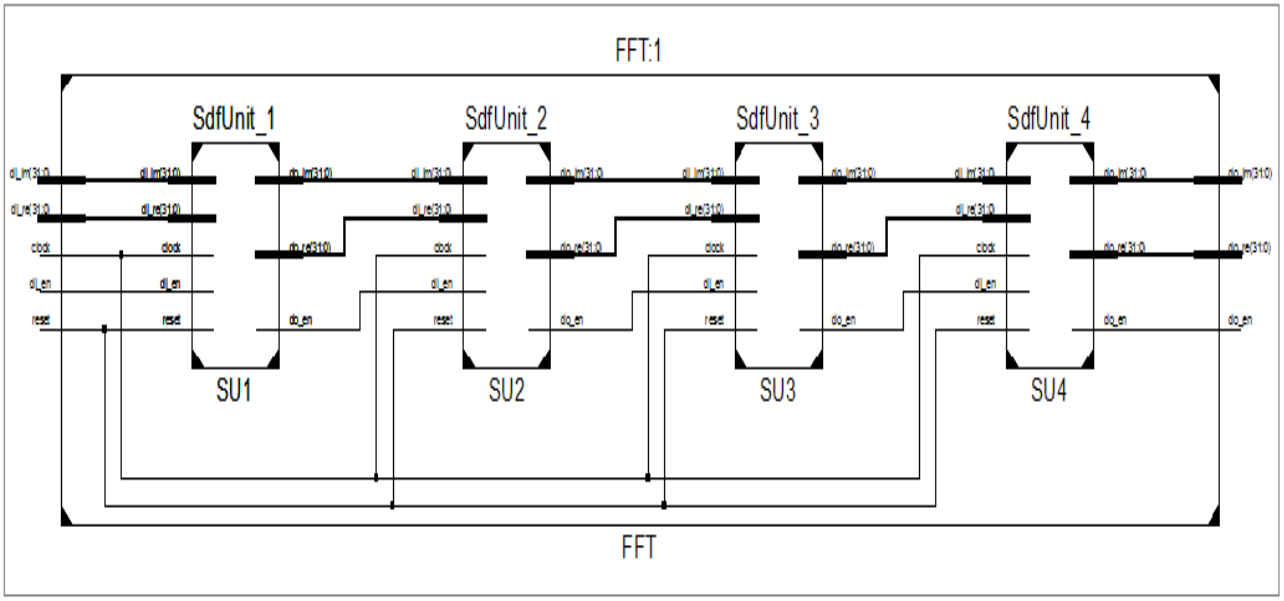

Fig 4.10 Architecture of 256-point FFT processor

A. Operation:

The operation of 256-point FFT is the same as 64-point and 128-point FFT, which is described in the section no 4.2.

B. Formation:

The submodules which are required to form the 256-point FFT is the same as the 64,128-point FFT. The only difference is in the no of delay buffer and twiddle factor. For 256-point FFT, after 2^{nd} stage it requires 189 number of twiddle factors, after 4^{th} stage it requires 180 no of twiddle and after 6^{th} stage it requires 144 number of twiddle factors based on $W_N^{n_3(k_1+2k_2)}$. The number of delay buffer it requires is 8.

4.5 Radix-2^2 SDF Pipeline 512-Point FFT Architecture

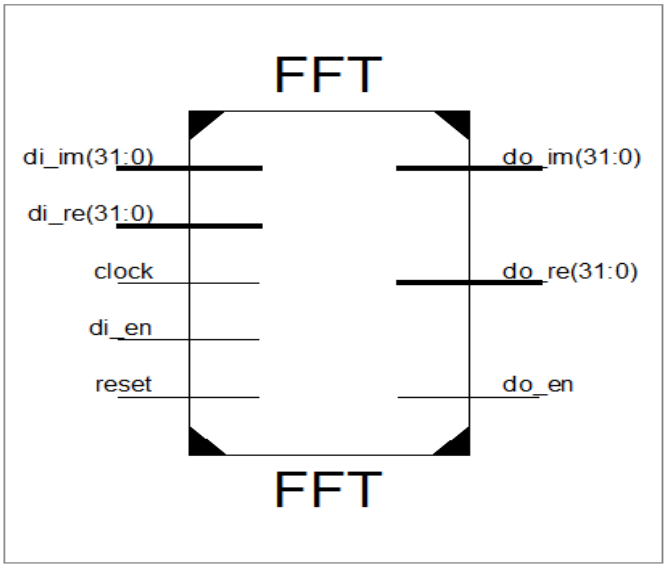

Fig 4.11 RTL diagram of 512-point FFT processor

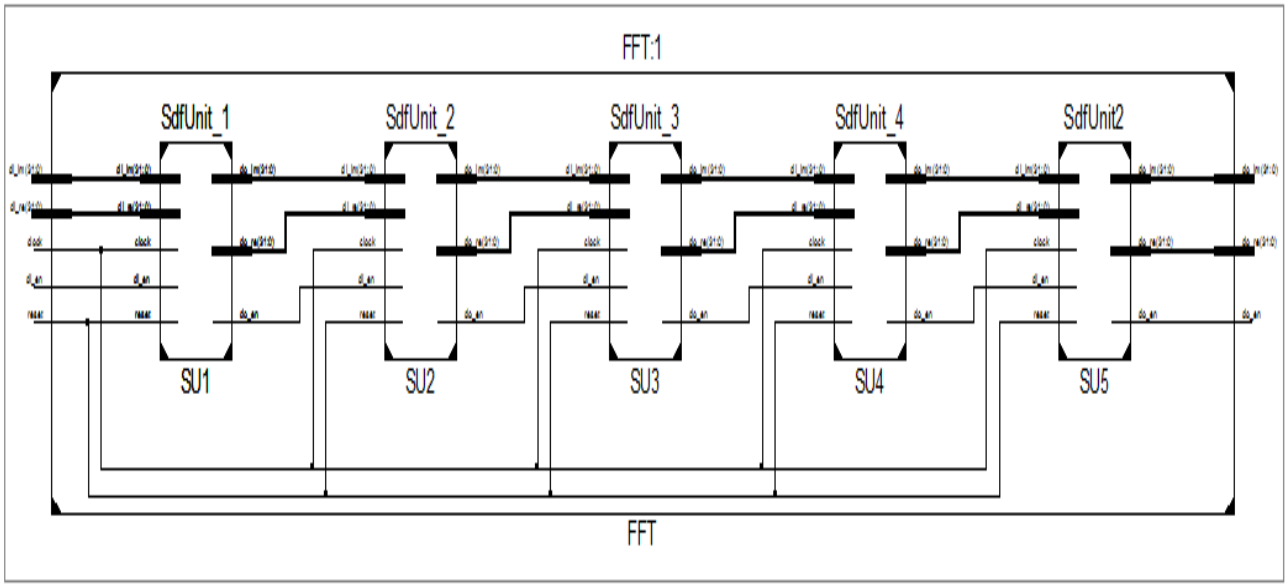

Fig 4.12 Architecture of 512-point FFT processor

A. Operation

The operation of 512-point FFT is the same as 64, 128, and 256-point FFT, which is described in the section no 4.2.

B. Formation:

The submodules which are required to form the 512-point FFT is the same as the 64, 128, 256-point FFT. The only difference is in the number of delay buffer and twiddle factor. For 512-point FFT, after 2nd stage it requires 381 number of twiddle factors, after 4th stage it requires 372 number of twiddles, after 6th stage it requires 336 number of twiddle factors and after 8th stage it needs 192 number of twiddle factors based on $W_N^{n_3(k_1+2k_2)}$. The number of delay buffer it requires is 9.

4.6 Radix-2^2 SDF Pipeline 1024-Point FFT Architecture

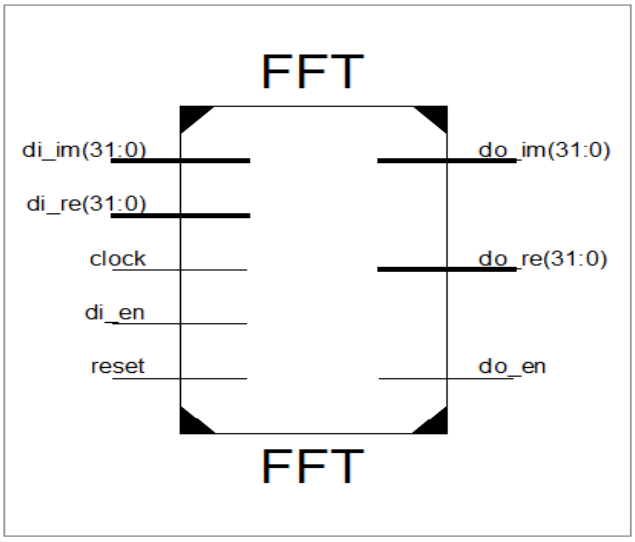

Fig 4.13 RTL diagram of 1024-point FFT processor

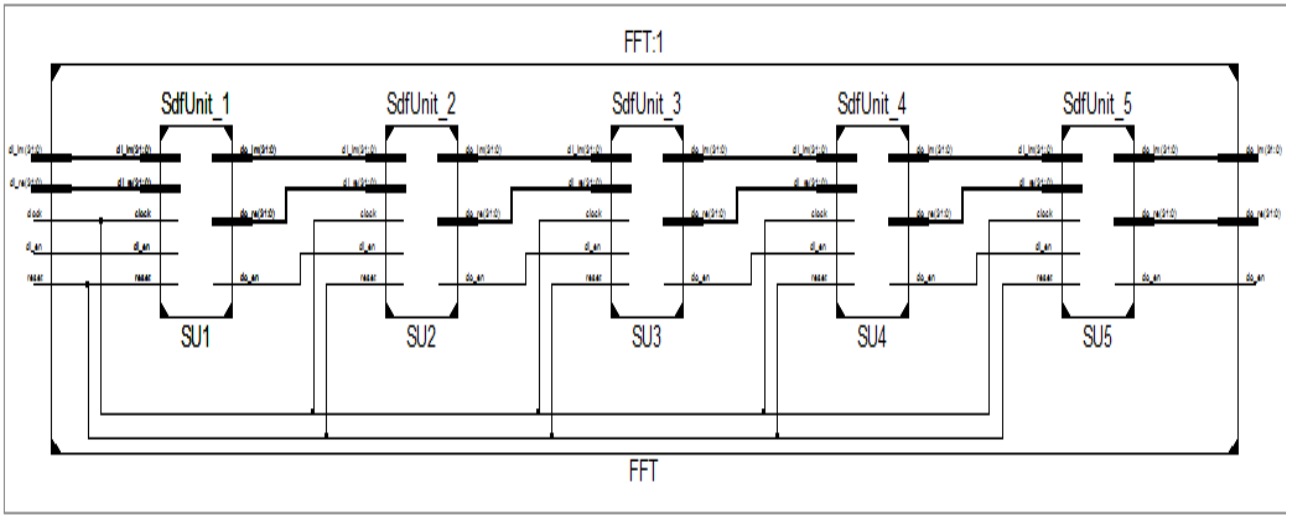

Fig 4.14 Architecture of 1024 point FFT processor

A. Operation

The operation of 1024-point FFT is the same as 64, 128, 256, and 512-point FFT, which is described in the section no 4.2.

B. Formation:

The submodules which are required to form the 1024-point FFT is the same as the 64, 128, 256, 512-point FFT. The only difference is in the no of delay buffer and twiddle factor. For 1024-point FFT, after 2nd stage it requires 765 number of twiddle factors, after 4th stage it requires 756 number of twiddle factor, after 6th stage it requires 720 number of twiddle factors and after 8th stage it needs 576 number of twiddle factors based on $W_N^{n_3(k_1+2k_2)}$. The number of delay buffer it requires is 10.

CHAPTER 5:

FPGA-Based Synthesis Results

In this section, different kind of FFT processor with different size have been represented in Verilog hardware description language (HDL) in FPGA platform. The FPGA synthesis results of all functional blocks for radix-2^2SDF pipeline FFT processor for 64,128,256,512 and 1024-point have been synthesized using Virtex-7 (xc7vx330t-2ffg-1157) FPGA device family of Xilinx Vivado 2019.1 synthesis tool. We have analyzed the performance of our FFT architecture which is shown in Table 5.1. The synthesis results of these architecture in terms of slice LUTs, slice registers, input output buffers (IOBs), delay, and power in three different FPGA device families such as Artix-7 (xc7a100t-3csg324), virtex-6 (6vcx75tff484-2) and virtex-7 (xc7vx330t-2ffg-1157).

Table 5.1: FPGA synthesis result of implemented FFT architectures of different input size:

FFT Size	No. of Slice Registers	No. of Slice LUTs	No. of Bonded IOBs	Delay (ns)	Logic Power(W)	Target Device
64	681	1351	68	4.874	2.01	Artix-7
128	823	1582	68	7.202	2.12	
256	1786	3662	132	8.798	2.78	
512	2059	4585	132	8.812	3.08	
1024	2295	6027	132	9.123	3.33	
64	683	1357	68	4.985	2.24	Virtex-6
128	827	1586	68	7.293	2.38	
256	1790	3667	132	8.820	3.06	
512	2062	4589	132	8.854	3.18	
1024	2299	6028	132	8.204	3.40	
64	681	1351	68	4.850	2.04	Virtex-7
128	823	1582	68	7.199	2.11	
256	1786	3662	132	8.795	2.78	
512	2059	4585	132	8.810	3.10	
1024	2295	6027	132	9.121	3.36	

Table 2 and Table 3 show the comparison of the implemented architectures and the existing architectures in terms of number of slice LUTs, number of slice registers, IOBs and delay in two FPGA device families such as virtex-7 (xc7vx330t-2ffg-1157) and Artix-7 (xc7a100t-3csg324).

Table 5.2: Comparison of the implemented and the existing FFT architectures on Virtex-7:

FFT Size	FFT Architecture	No. of Slice Registers	No. of Slice LUTs	No. of Bonded IOBs	Delay (ns)	Target Device
64	Nakhate et al. [11]	725	1345	-	5.278	Virtex-7
	Shirbhate et al. [24]	2068	2991	87	5.988	
	Wang et al. [13]	-	1110	-	6.120	
	Naidu et al. [8]	-	3118	87	-	
	Our	681	1351	68	4.850	
128	Nakhate et al. [11]	1042	1937	-	8.240	
	Bansal et al. [10]	989	2170	102	8.524	
	Our	823	1582	68	7.199	
256	Nakhate et al. [11]	1323	2497	-	9.923	
	Joseph et al. [21]	-	2233	-	-	
	Wang et al. [13]	-	1733	-	9.548	
	Naidu et al. [8]	-	7057	67	-	
	Our	1786	3662	132	8.795	
512	Nakhate et al. [11]	1795	3335	-	9.948	
	Our	2059	4585	132	8.810	
1024	Wang et al. [13]	-	2804	-	-	
	Our	2295	6027	132	9.121	

Table 5.3: Comparison of the implemented and the existing FFT architectures on Artix-7:

FFT Size	FFT Architecture	No. of Slice Registers	No. of Slice LUTs	No. of Bonded IOBs	Delay (ns)	Target Device
64	Nakhate et al. [11]	725	1345	-	5.278	Artix-7
	Shirbhate et al. [24]	2068	2991	87	5.988	
	Wang et al. [13]	-	1110	-	6.120	
	Naidu et al. [8]	-	3118	87	-	
	Our	681	1351	68	4.874	
128	Nakhate et al. [11]	1042	1937	-	8.240	
	Bansal et al. [10]	989	2170	102	8.524	
	Our	823	1582	68	7.202	
256	Nakhate et al. [11]	1323	2497	-	9.923	
	Joseph et al. [21]	-	2233	-	-	
	Wang et al. [13]	-	1733	-	9.548	
	Naidu et al. [8]	-	7057	67	-	
	Our	1786	3662	132	8.798	
512	Nakhate et al. [11]	1795	3335	-	9.948	
	Our	2059	4585	132	8.812	
1024	Wang et al. [13]	-	2804	-	-	
	Our	2295	6027	132	9.123	

CHAPTER 6:

Summary

&

Future Scope

Conclusion:

We have successfully designed radix-2^2 SDF pipeline architecture for 64, 128, 256, 512 and 1024-point FFT based on common factor algorithm. By using radix-2^2-algorithm we can able to decrease the area because in this algorithm the number of twiddle factors is less. To reduce the complexity, the radix-2 butterfly structure is employed. This overall effect increases speed as the number of multiplications decreases, sending input signal rather than multiplying the twiddle factor W_N^0 since the value of $W_N^0 = 1$, and reusing twiddle factor value for other point FFT. In comparison to conventional FFT processors, this method raises the maximum frequency of the implemented FFT processor. By using this architecture we can also decrease the logic power.

Future Scopes:

Several researches are still trying to improve the FFT architectural parameters such as delay, area and power. In future we will be exploring the performances of our implemented FFT architectures using different combination of multiplier, adder/subtractor and studied their merits and demerits.

References:

1. T. Y. Sung, H. C. Hsin, and L.T. Ko, “Reconfigurable VLSI architecture for FFT processor”, In WSEAS International Conference. Proceedings. Mathematics and Computers in Science and Engineering, No. 9, World Scientific and Engineering Academy and Society, June 2009.
2. P. Choudhary and A. Karmakar, “CORDIC based implementation of Fast Fourier Transform”, In 2011 2nd International Conference on Computer and Communication Technology (ICCCT-2011), IEEE, pp. 550-555, September 2011.
3. A. Sapper, G. Paim, E. A. C. da Costa and S. Bampi, “Exploring the CORDIC Algorithm and Clock-Gating for Power-Efficient Fast Fourier Transform Hardware Architectures”, Journal of Integrated Circuits and Systems, 16(2), pp.1-11, 2021.
4. C. Yu, M. H. Yen, P. A. Hsiung and S. J. Chen, “A low-power 64-point pipeline FFT/IFFT processor for OFDM applications”, IEEE transactions on consumer electronics, 57(1), pp.40-40, 2011.
5. B. Zhou, Y. Peng, and D. Hwang, “Pipeline FFT architectures optimized for FPGAs”, International Journal of Reconfigurable Computing, 2009.
6. S. Chandra, “Sign/logarithm arithmetic for FFT implementation”, IEEE Transactions on Computers 100, No. 6, pp-526-534, 1983.
7. T. Ahmed, M. Garrido and O. Gustafsson, “A 512-point 8-parallel pipelined feedforward FFT for WPAN”, In 2011 Conference Record of the Forty Fifth Asilomar Conference on Signals, Systems and Computers (ASILOMAR), IEEE, pp. 981-984, November 2011.
8. P. S. Naidu, “Fourier transform of large-scale aeromagnetic field using a modified version of fast Fourier transform”, pure and applied geophysics, 81(1), pp.17-25, 1970.
9. H. Magsi and A. H. Sodhro, “Fast Fourier Transform by Verilog, November 2020.
10. M. Bansal and S. Nakhate, “Implementation of fast FFT design for 128-point using Radix-22 CFA”, International Journal of Engineering and Technology, 7(4), pp.2646-2650, 2018.
11. M. Bansal and S. Nakhate, “Fast performance pipeline re-configurable FFT processor based on radix-2 2 for variable length N”, International Journal of Electrical and Electronic Engineering & Telecommunications”, 8(3), pp.163-170, 2019.
12. M. Sridhanya, and Mrs. G. Annapurna, “Efficient design of FFT/IFFT processor using Verilog HDL”, International Journal of Professional Engineering Studies (IJPRES), July 2015.
13. W. Liu, F. Lombardi and M. Shulte, “A retrospective and prospective view of approximate computing,” Proceedings of the IEEE, Vol. 108, No. 3, pp. 394-399, 2020.
14. R. N. Bracewell and R. N. Bracewell, “The Fourier transform and its applications” Vol. 31999, pp. 267-272, New York: McGraw-Hill, 1986.
15. A. V. Oppenheim, J. R. Buck and R. W. Schafer, “Discrete-time signal processing”, Vol. 2, Upper Saddle River, NJ: Prentice Hall, 2001.
16. P. Kleewein, C. Rosolen, A. Lecacheux, “New digital spectrometers for ground-based decametre radio astronomy”, Austrian Academy of Sciences Press, Vienna, 349, 1997.

17. R. D. Strum, D. E. Kirk, “Linear systems: be discrete-then continuous. IEEE Transactions on Education”, 32(3), pp.335-342, 1989.
18. C. T. Mullis and R. A. Roberts, “On the design of orthogonal digital filters”, In Twenty-Second Asilomar Conference on Signals, Systems and Computers, IEEE, Vol. 1, pp. 320-322, October 1988.
19. B. M. Baas, “An approach to low-power, high-performance, fast Fourier transform processor design”, Stanford University, 1999.
20. B. Wang, Q. Zhang, T. Ao and M. Huang, “Design of pipelined FFT processor based on FPGA”. In 2010 Second International Conference on computer modeling and simulation, IEEE, Vol. 4, pp. 432-43, January 2010.
21. E. Joseph, A. Rajagopal and K. Karibasappa, “FPGA implementation of Radix-2 FFT processor based on Radix-4 CORDIC”, In 2012 Nirma University International Conference on Engineering (NUiCONE), IEEE, pp. 1-6, December 2012.
22. J. Du, K. Chen, P. Yin, C. Yan and W. Liu, “Design of An Approximate FFT Processor Based on Approximate Complex Multipliers”, In 2021 IEEE Computer Society Annual Symposium on VLSI (ISVLSI), IEEE, pp. 308-313, July 2021.
23. Z. Wang, X. Liu, B. He and F. Yu, “A combined SDC-SDF architecture for normal I/O pipelined radix-2 FFT”, IEEE Transactions on very large-scale integration (VLSI) systems”, 23(5), pp.973-977, 2014.
24. R. Shirbhate, T. Panse and C. Ralekar, “Design of parallel FFT architecture using Cooley Tukey algorithm”, In 2015 International Conference on Communications and Signal Processing (ICCSP), IEEE, pp. 0574-0578, April 2015.
25. X. Liu, F. Yu, Z. K. Wang, “A pipelined architecture for normal I/O order FFT,” J Zhejiang Univ-Sci C (Comput & Electron), 12(1), pp.76-82, 2011.
26. B. Liu, X. Ding, H. Cai, W. Zhu, Z. Wang, W. Liu and J. Yang, “Precision adaptive MFCC based on R2SDF-FFT and approximate computing for low-power speech keywords recognition”, IEEE Circuits and Systems Magazine, 21(4), pp.24-39, 2021.
27. S. J. Huang and S. G. Chen, “A green FFT processor with 2.5-GS/s for IEEE 802.15. 3c (WPANs)”, In The 2010 International Conference on Green Circuits and Systems, IEEE, pp. 9-13, 2010.
28. W. H. Chang and T. Q. Nguyen, “On the fixed-point accuracy analysis of FFT algorithms”, IEEE Transactions on Signal Processing, 56(10), pp.4673-4682, 2008.