République Algérienne Démocratique et Pop

ulaire
Ministère de l'enseignement supérieur et de la recherche scientifique
Université Dr.Yahia Farès de MEDEA



Faculty of Technology
Department of Electrical engineering

Master's End of Study Project

Branch: Telecommunications

Option : Telecommunication Systems

Thème

Design, simulation and FPGA implementation of high throughtput FFT Processor for MIMO-OFDM

Etudié par :

-Abdelmalek Abderrahmane
-Roubehie Fissa Fares

Dirigé par :

- Dr : Dali Mohammed

promotion 2019-2020

Acknowledgments

First of all, we would like to thank ALLAH the Almighty and Merciful, who has given us the strength and patience to do this modest work.

Secondly, we thank our supervisor Dr. DALI Mohammed for his valuable advice and his help in carrying out this graduation project.

Our sincere thanks also go to the members of the jury, for accepting to examine our work and for enriching it with their propositions.

We would like to express our sincere thanks to all the teachers who contributed in our university education.

Finally, we would also like to thank our family and all people who supported us during the realization of this work.

DEDICATIONS

To mom, all member of my Family.

To all my friends with their help and support. And very special thanks to my best friend.

To all my colleagues they were a great team

To everyone who wants to make a positive change in the world.

Abderrahmane.

I dedicate this work

To my dear parents, my mother and my father

for their patience, their love, their support and their encouragement.
And to my sisters.
To my friends and my brothers from other mothers: Chiker Said Mustapha and Hocine Anouar.

Fares.

ABSTRACT

The goal of this project is to design, simulate, and implement on FPGA a DFT calculation processor for MIMO-ODFM high throughput systems. The circuit can calculate DFTs of two independent data sequences. Designs and simulations are carried out using Xilinx System Generator (XSG) software. Architectures are based on the FFT Multipath Delay Commutator (MDC) Mixed Radix 2^2-2^3 concept (MDC MR-2^2-2^3). In order to increase the clock frequency, we have inserted pipeline registers at the outputs of basic elements of the architecture. Increasing frequency allows us to obtain high throughput processing without a significant increase in resource usage. The designs are implemented on the powerful FPGA circuit Virtex-7 XC7VX485T_2FFG1761.

Keywords: FFT, FPGA, MDC MR-2^2-2 , MIMO-OFDM, System Generator

ملخص

الهدف من هذا المشروع هو إنجاز تصميم, محاكاة و تنفيذ على دارة من نوع FPGA لحاسب DFT من أجل استعماله في نظام اتصالات من نوع MIMO-ODFM ذات تدفق عالي. الدارة المستعملة بإمكانها حساب DFT لتسلسلين مستقلين من البيانات. التصاميم و المحاكاة المنجزة تم إجراؤها بإستعمال برمجية XSG. نماذج التصاميم تم بناؤها وفقا لمبدا تحويل فوريي السريع المبني على MDC MR-2^2-2^3, ومن ثم قمنا بإجراء بعض التعديلات على البنية بقصد رفع قيمة التردد و التي بدورها تقوم برفع سرعة البيانات المعالجة دون زيادة كبيرة في الموارد المستخدمة. التصاميم المنجزة تم تنفيذها على دارة FPGA جد فعالة من نوع Virtex-7 XC7VX485T_2FFG1761

كلمات مفتاحية: FFT ، FPGA، MDC MR-2^2-2^3، MIMO-OFDM، System Generator.

RÉSUMÉ

L’objectif de ce projet est de concevoir, simuler et l’implémenter sur FPGA un processeur de calcul de la DFT pour les systèmes MIMO-OFDM à haut débit. Le circuit peut calculer les DFTs de deux séquences de données indépendantes. Les conceptions et les simulations sont effectuées à l’aide du logiciel XSG. Les architectures sont développées selon le concept d’implémentation de l’algorithme FFT appelé MDC MR-2^2-2^3. Pour augmenter la fréquence de fonctionnement, nous avons inséré des registres de pipeline au niveau des sorties des éléments de base de l’architecture. Cela permet à augmenter le débit des données sans épuiser d’avantage les ressources utilisées. Les architectures sont implémentées sur un circuit FPGA puissant de type Virtex-7 XC7VX485T_2FFG1761.

Mots clés : FFT, FPGA, MDC MR-2^2-2^3, MIMO-OFDM, System Generator.

TABLE OF CONTENTS

ABSTRACT III
TABLE OF CONTENTS IV
LIST OF FIGURES VII
LIST OF TABLES IX
LIST OF ABBREVIATION IV
GENERAL INTRODUCTION 1
CHAPTER I: OVERVIEW OF DIGITAL COMMUNICATIONS AND MIMO-OFDM 3
1.1 Introduction 3
1.2 Digital Communications System 3
1.3 Orthogonal Frequency Division Multiplexing OFDM 5
1.4 Basic principle of OFDM 6
1.5 the advantages and drawbacks of OFDM 8
1.6 Multiple-Input–Multiple-Output System (MIMO) 8
1.7 MIMO-OFDM 11
1.8 Conclusion 12
CHAPTER II: FAST FOURIER TRANSFORM 13
2.1 Introduction 13
2.2 The concept of DFT: 13
2.3 Algorithm FFT 14
2.3.1. Radix 2 DIT-FFT 14
2.3.2. Radix 2 DIF-FFT 18
2.3.3. Radix 2^2 Algorithm 22
2.3.4. Radix 2^3 Algorithm 25
2.4 Bit-reversal permutation 27
2.5 Pipeline architecture for FFT implementation 27
2.5.1. Single Path Delay Feedback (SDF) Architecture 28
2.5.2. Multiple Path Delay Commutator (MDC) Architecture 29
2.5.3. Single Path Delay Commutator (SDC) Architecture 30
2.6 Comparison 31
2.7 Conclusion 32
CHAPTER III: OVERVIEW OF FPGAS CIRCUIT 33

3.1 Introduction 33
3.2 Definition of FPGA 33
3.3 Architecture and Component of FPGA 33
3.3.1. CLB 34
3.3.2. Input/output Blocks 35
3.3.3. Switching Matrix/Interconnection Wires 35
3.4 FPGA Programming Technologies 35
3.4.1. SRAM-Based Programming Technology 35
3.4.2. Flash Programming Technology 36
3.4.3. Anti-fuse Programming Technology 36
3.5 FPGA design flow 36
3.5.1. Design Entry 37
3.5.2. Logic Synthesis 37
3.5.3. Implementation 38
3.5.4. Device Programming 39
3.5.5. Design verification 40
3.6 Types of FPGA 40
3.7 FPGA Advantages and Disadvantages 41
3.7.1. Advantages 41
3.7.2. Disadvantages 42
3.8 Conclusion 43
CHAPTER IV: IMPLEMENTATION AND SIMULATION OF FFT IN MDC ARCHITECTURE 44
4.1 Introduction 44
4.2 Organizational Chart 44
4.3 Constituent blocks of MDC Radix-2^2 and Radix-2^3 architecture 46
4.3.1 Bloc diagram of architecture 46
4.3.2 Clock frequency increase technique 46
4.3.3 Inputs scheduling unit 46
4.3.4 Calculation modules 48
4.4 Simulation results 53
4.4.1 Architecture for *N*=8 53
4.4.2 Architecture for *N*=16 54
4.4.3 Architecture for *N*=32 54

4.4.4 Architecture for *N*=64 55
4.5 Implementation results 56
4.6 Verification by hardware-software Co-Simulation 60
4.7 Conclusion 61
GENERAL CONCLUSION 62

LIST OF FIGURES

Fig.1.1 Basic elements of a digital communication system 4

Fig.1.2 Concept of the OFDM signal: (a) conventional multicarrier technique, and (b) orthogonal multicarrier modulation technique. 6

Fig.1.3 Spectra of an OFDM subchannel 7

Fig.1.4 Spectra of an OFDM signal. (By MATLAB) 7

Fig.1.5 Basic structure of MIMO system 9

Fig.1.6 (a) Schematic of a MIMO-OFDM system. 11

Fig.2.1 division of XF to GF and HF 15

Fig.2.2 Butterfly to compute (II.4) 16

Fig.2.3 Division of the DIF-FFT algorithm with $N = 8$ 17

Fig.2.4 Radix 2 DIT FFT flow graph for N = 8 18

Fig.2.5 Butterfly structure for Radix-2 DIF algorithm. 20

Fig.2.6 Flow graph of decimation-in frequency decomposition of an N-point DFT into two N/2-point DFT computations ($N = 8$) ($W_N = W_8$) 21

Fig.2.7 Flow graph of decimation-in frequency decomposition of an eight-point DFT into two-point DFT computations. WN = W8 21

Fig.2.8 Flow graph of complete decimation-in frequency decomposition of an eight-point DFT computation. WN = W8 22

Fig.2.9 SFG of the basic butterfly structure of Radix-2^2 DIF algorithm 244

Fig.2.10 SFG of the basic butterfly structure of Radix-2^3 DIF algorithm 266

Fig.2.11 general structure of pipeline architecture 28

Fig.2.12 A 16- point radix-2 single path delay feedback architecture 29

Fig.2.13 A 16-point radix-4 multiple path delay commutator architecture 29

Fig.2.14 A 16-point radix-4 single path delay commutator architecture 300

Fig 3.1 FPGA architecture [14] 34

Fig 3.2 Configurable logic block 34

Fig 3.3 FPGA Architectural Design Flow 37

Fig 3.4 Xilinx Synthesis process 38

Fig 3.5 FPGA Translate process 38

Fig 3.6 FPGA Mapping with Static Timing Analysis 39

Fig 3.7 FPGA Place and Route 39

Fig 3.8 Types of FPGAs 40

Fig.4.1 Design flow graph 45

Fig.4.2 General Structure of the proposed MR22 - R23 MDC architecture 46
Fig.4.3 Schema of inputs scheduling unit 47
Fig.4.4 Design of the inputs scheduling unit using XSG 48
Fig.4.5 Schema of module Radix-2^2 for the architecture $MR2^2$ -2^3MDC with two sequences 49
Fig.4.6 Design of Radix 2^2 2 module using XSG 49
Fig.4.7 Structure of addition/subtraction unit using XSG 50
Fig.4.8 Architecture of shuffling unit using XSG..... 50
Fig.4.9 Structure of module Radix-2^3 for the architecture $MR2^2$ -2^3 MDC with two sequences 51
Fig.4.10 Design of Radix 2^3 module using XSG. 52
Fig.4.11 Architecture of CCMU1 unit using XSG 52
Fig.4.12 Global architecture for *N*=8 using XSG. 53
Fig.4.13 Global architecture for N=16 using XSG. 54
Fig.4.14 Global architecture for *N*=32 using XSG. 55
Fig.4.15 Global architecture for *N*=64 using XSG. 56
Fig.4.16 Generating HDL Netlist by System Generator. 57
Fig.4.17 Window of ISE for implementation step. 58
Fig.4.18 Picture of System Generator co-simulation block inserted in the design. 60
Fig.4.19 Hardware co-simulation process running on VC707 FPGA board 61

LIST OF TABLES

Table 2.1 An example of bit reversed order for *N* = 8 27

Table 2.2 Comparison hardware of different pipelined architecture used.[12] 31

Table 2.3 Resource utilization rate of the different architectures.[11] 311

Table 4.1 Simulation results for *N*=8. 53

Table 4.2 First eight simulation results for *N*=16.. 54

Table 4.3 First eight simulation results for *N*=32 55

Table 4.4 First eight simulation results for *N*=64 56

Table 4.5 Virtex-7 XC7VX485T_2FFG1761 resource usage of the low frequency architectures. 58

Table 4.6 Virtex-7 XC7VX485T_2FFG1761 resource usage of the high frequency architectures. 59

Table 4.7 Maximum frequency of high frequency architectures in comparison with low-frequency architectures 59

LIST OF ABBREVIATION

ASIC	Application Specific Integrated Circuit
AWGN	Additive White Gaussian Noise
CCMU	Constant Complex Multiplication Unit
CLB	Configurable Logic Blocks
CMOS	Complementary Metal Oxide Semiconductor
CP	Cyclic Prefix
CPLD	Complex Programmable Logic Device
CPU	Central Processing Unit
DFT	Discrete Fourier Transform
DIF	Decimation-In-Frequency
DIT	Decimation-In-Time
DRAM	Dynamic Random Access Memory
EPROM	Erasable Programmable Read Only Memory
FFT	Fast Fourier Transform
FPGA	Field Programmable Gate Array
HDL	Hardware Description Language
IDFT	Inverse Discrete Fourier Transform
IEEE	Institute of Electrical and Electronics Engineering
IFFT	Inverse Fast Fourier Transform
IOB	Input/Output Blocks
ISE	Integrated Synthesis Environment
ISI	InterSymbol Interference
JTAG	Joint Test Action Group
LAB	Logic Array Blocks
LUT	LookUp Table
MDC	Multiple path Delay Commutator
MIMO	Multi Input Multi Output

MUX	Multiplexer
NCD	Native Circuit Description
NGC	Native Generic Circuit
NGD	Native Generic Database
NRE	Non-Recurring Engineering
OFDM	Orthogonal Frequency Division Multiplexing
OTP	One Time Programmable
PAR	Placement And Route
PC	Personal Computer
PLD	Programmable Logic Device
QoS	Quality of Service
RAM	Random Access Memory
ROM	Read Only Memories
ROML	Read Only Memories-Lower
ROMU	Read Only Memories-Upper
RTL	Register-Transfer-Level
RTL	Register Transfer Level
SC	Signal carrier
SDC	Single path Delay Commutator
SDF	Single path Delay Feedback
SFG	Signal Flow Graph
SNR	Signal-to-Noise Ratio
SRAM	Static Random Access Memory
UCF	User Constraints File
VHDL	VHSIC-Hardware Description Language
VHSIC	Very High Speed Integrated Circuit
XSG	Xilinx System Generator
XST	Xilinx Synthesis Technology

General Introduction

Digital wireless communications is a rapidly growing industry in the last decades. Today's wireless communication systems can provide high-throughput and high-quality of service information exchange between users in the four corners of the world. An important factor that led to this tremendous development is the introduction of new signal processing methods and techniques such as Code Division Multiple Access (CDMA), Orthogonal Frequency Division Multiplexing (OFDM), and Multiple Input Multiple Output (MIMO). The combination of the MIMO and OFDM, i.e. MIMO-OFDM system has been widely recognized to significantly improves the reliability and throughput in wireless communication.

The crucial blocks in communication systems based on MIMO-OFDM is the bloc calculating the Discrete Fourier Transform (DFT). It is important to implement this bloc to insure efficient and fast calculation. One of the most efficient solutions is to use a Field Programmable Gate Arrays (FPGA) circuit to achieve this implementation. Our work is in the objective of the implementation on FPGA of architecture for two independent sequences DFT's computation.

This work is organized in four chapters as follow:

The first chapter give an overview of digital communication systems. We start by basic elements of communication systems. Then, we introduce the basic principle of OFDM and we give some advantages and disadvantages of OFDM. MIMO and MIMO_OFDM techniques are also discussed in the end of this chapter.

The second chapter is dedicated to the calculation algorithms and to the implementation architectures of the DFT. First, we give the concept of DFT, then we present several version of the FFT algorithm such as the decimation-in-time (DIT) and decimation-in-frequency (DIF). After that we talk about the pipeline architecture. Then we give a comparison of different pipelined architecture.

In Chapter three, we provide an overview of Field Programmable Gate Arrays (FPGAs), and we provide a definition of FPGAs as well as the architecture and components of FPGAs. Then we talk about the programming techniques responsible for such reconfigurable engineering, and we

also talk about the FPGA architectural design. Next, we introduce the types of FPGAs. Finally, we introduce some advantages and disadvantages of an FPGA circuit.

In the final chapter, we present the detailed architectures, simulations results and FPGA implementation results. We explain how the architectures are designed based on the MDC MR-2^2-2^3 concept and using the XSG tool. Obtained results during simulation step are compared with result of Matlab. A method to improve frequency is also presented in this chapter. Implementation results are discussed in the end of this chapter.

Chapter I

Overview of digital communications and MIMO-OFDM

1.1 Introduction

The domain of telecommunications has been developing quickly in recent decades. All this development aims to transfer the largest amount of data from one end to the other as quickly as possible without losing pieces of information. In light of this development, new technologies such as MIMO-OFDM appeared which allowed a better exploitation of spatial diversity.

In this chapter, we will give an overview of digital communication concepts and techniques such as digital systems blocs, Orthogonal Frequency-Division Multiplexing (OFDM) and Multiple-Input Multiple-Output (MIMO).

1.2 Digital Communications System

Digital communication has proliferated in a big way in previous years. Its goal is to transmit data over Point-To-Point or Point-To-Multipoint communication channel. In digital communication, the data transfer rate depends on several parameters like the modulation scheme, channel model... Digital communication can be done over large distances through the internet and other networks.[1]

In its simplest form a telecommunications system consists of a transmitter, a channel, a receiver.

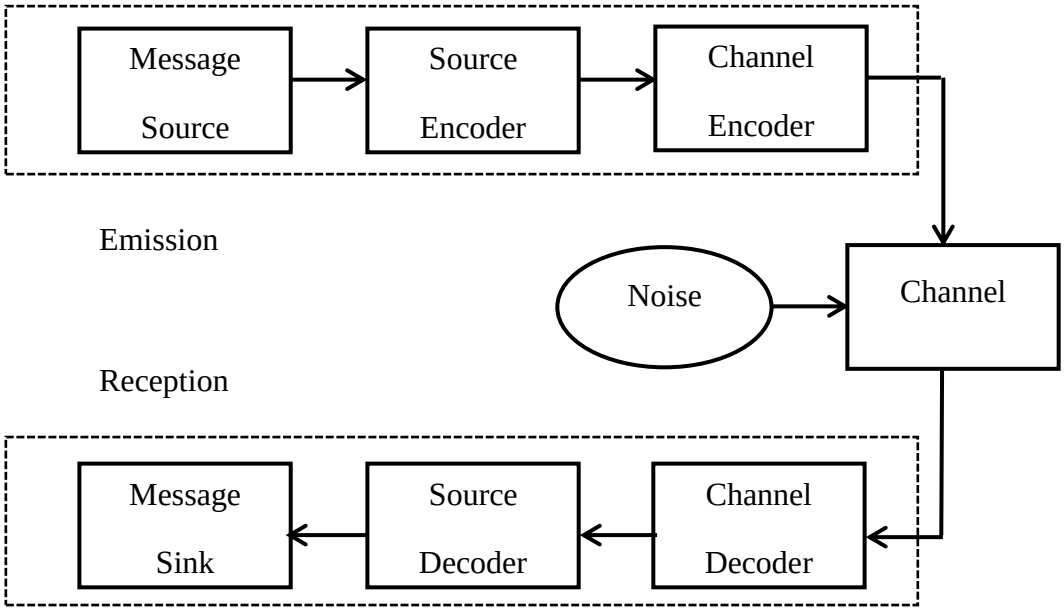


Fig.1.1 Basic elements of a digital communication system

Source Encoder: The purpose of the source encoder is to compress data. It consists of eliminating any unnecessary redundancy in the information in order to shorten the message to be transmitted and thus reduce the transmission time and the size of the storage volume.

Source Decoder: Having the binary sequence with the information, this decoder reconstitutes the transmitted information.

Channel Coding: One purpose of the source encoder is to eliminate redundant binary digits from the digitized signal. The strategy of the channel encoder, on the other hand, is to add redundancy to the transmitted signal in this case so that errors caused by noise during transmission can be corrected at the receiver. The process of encoding for protection against channel errors is called error-control coding.

Channel decoding: Knowing the coding code used, the receiver is able to detect the wrong binary data and then correct it, making it possible to find the primary binary information as much as possible. Generally, it cannot correct everything otherwise it will require a lot of redundant bits. They only correct potential errors.

Transmission Channel: A communication channel refers either to a physical transmission medium such as a wire or to a logical connection over a multiplexed medium such as a radio channel in telecommunications. A channel is used to convey an information signal, for example, a digital bit stream, from one or several senders (or transmitters) to one or several receivers. A channel has a certain capacity for transmitting information, often measured by its bandwidth in Hz or its data rate in bits per second. Transmission can be by electrical conductors, radio, or optical fiber. Although these appear to be completely different, they are in fact just different examples of electromagnetic energy traveling from one place to another. If the energy is made time-variant, information can be carried.

Noise: The most used channel model for simulating digital transmissions, which is also one of the easiest to generate and analyze, is the Additive White Gaussian Noise channel (AWGN). These represent internal noise (thermal noise due to equipment defects ...) and external noise (antenna noise ...). However, this model is related to wire transmission, as it represents an almost perfect transmission from transmitter to receiver. Then the received signal is written as follows[2]:

$$r(t) = e(t) + n(t) \qquad \textbf{I.1}$$

- $r(t)$ Represents the received signal.
- $e(t)$ Represents the transmitted signal.
- $n(t)$ Represents the AWGN.

1.3 Orthogonal Frequency Division Multiplexing OFDM

Over the past few years, there has been increasing emphasis on extending the services available on wired public telecommunications networks to mobile/movable non-wired users. In addition to voice services, only low-bit-rate data services were available to mobile users.

Wired networks cannot support extension to wireless mobile networks because mobile radio channels are more contaminated than wired data-transmission channels. We also cannot preserve the high QoS required in wired networks [3].

To create broadband multimedia mobile communication systems, it is necessary to use high-bit-rate transmission of at least several megabits per second. However, if digital data is transmitted at the rate of several megabits per second, the delay time of the delayed waves is greater. Using adaptive equalization techniques at the receiver is one method for equalizing these signals. There are practical difficulties in operating this equalization at several megabits per second with compact, low-cost hardware [3].

1.4 Basic principle of OFDM

In a classical parallel-data system, the total signal frequency band is divided into N nonoverlapping frequency subchannels. Each subchannel is modulated with a separate symbol, and then the N subchannels are frequency multiplexed. It seems good to avoid spectral overlap of channels to eliminate inter-channel interference [3]. However, this leads to inefficient use of the available spectrum.

Fig (1.2) illustrates the difference between the conventional nonoverlapping multicarrier technique and the overlapping multicarrier modulation technique. By using the overlapping multicarrier modulation technique, we save almost 50% of bandwidth [3].

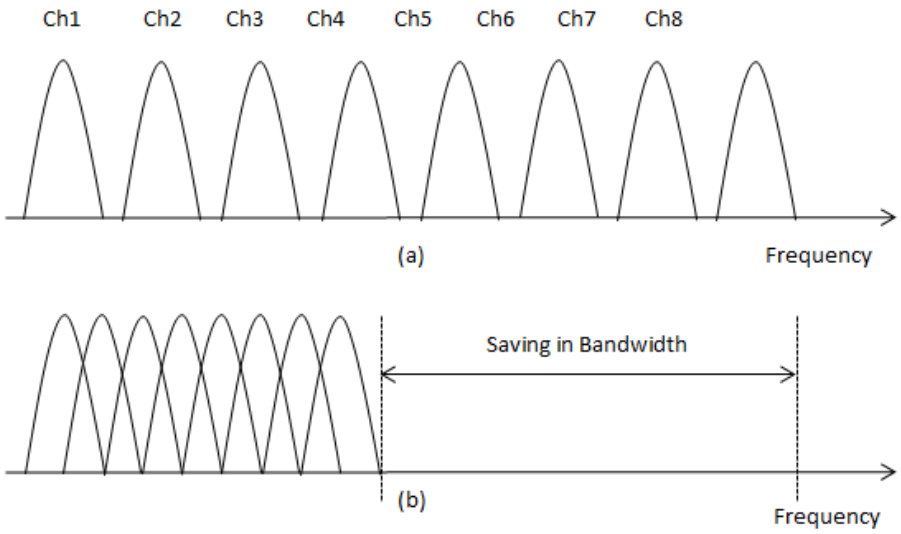


Fig.1.2 Concept of the OFDM signal: (a) conventional multicarrier technique, and (b) orthogonal multicarrier modulation technique.

To realize this technique, it is possible to arrange the carriers in an OFDM signal so that the sidebands of the individual carriers overlap and the signals are still received without adjacent carrier interference. To do this the carriers must be mathematically orthogonal.

Fig (1.3) shows the spectrum of the individual data of the subchannel. The OFDM signal, multiplexed in the individual spectra with a frequency spacing b equal to the transmission speed of each SC, is shown in Fig (1.4). Fig (1.3) and Fig (1.4) shows that at the center frequency of each SC, there is no cross talk from other channels. Therefore, if we use DFT at the receiver and calculate correlation values with the center of frequency of each SC, we recover the transmitted data with no cross talk. In addition, using the DFT-based multicarrier technique, FDM is achieved not by bandpass filtering but by baseband processing.[3]

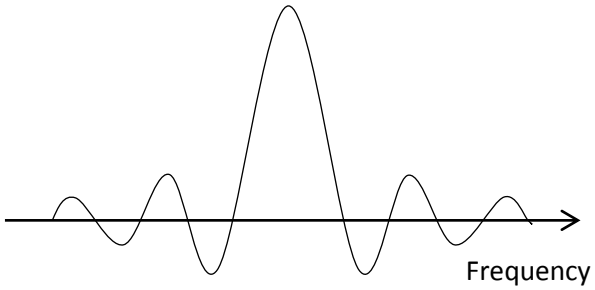


Fig.1.3 Spectra of an OFDM subchannel

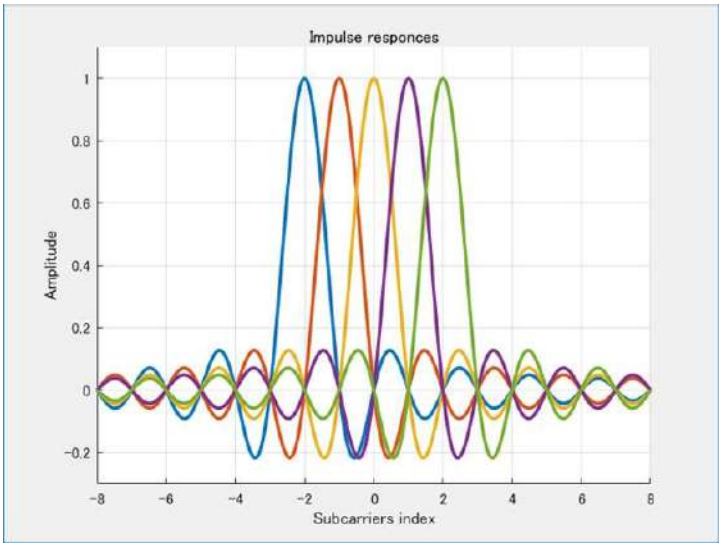


Fig.1.4 Spectra of an OFDM signal.

1.5 The advantages and drawbacks of OFDM

The OFDM transmission scheme has the following key advantages:

OFDM is an efficient way to deal with multipath; for a given delay spread, the implementation complexity is significantly lower than that of a single-carrier system with an equalizer.

- In relatively slow time-varying channels, it is possible to enhance capacity significantly by adapting the data rate per SC according to the signal-to-noise ratio (SNR) of that particular SC.
- OFDM is robust against narrowband interference because such interference affects only a small percentage of the SCs.
- OFDM makes single-frequency networks possible, which is especially attractive for broadcasting applications [4].

On the other hand, OFDM also has some drawbacks compared with single carrier modulation:

- OFDM is more sensitive to frequency offset and phase noise.
- OFDM has a relatively large peak-to-average-power ratio, which tends to reduce the power efficiency of the radio frequency (RF) amplifier.[4]

1.6 Multiple-Input–Multiple-Output System (MIMO)

MIMO wireless systems are those using multiple antenna elements at both the transmitter and receiver. They were first investigated by computer simulations in the 1980s, and later papers explored them analytically. Since that time, interest in MIMO systems has exploded. [5]

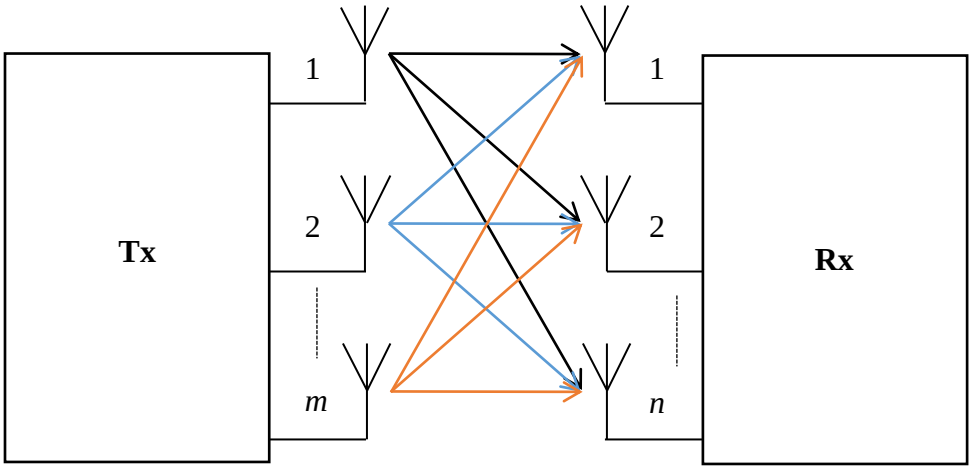


Fig.1.5 Basic structure of MIMO system

A channel may be affected by fading and this will impact the signal to noise ratio. In turn this will impact the error rate, assuming digital data is being transmitted. The principle of diversity is to provide the receiver with multiple versions of the same signal. If these can be made to be affected in different ways by the channel path, the probability that they will all be affected at the same time is considerably reduced. Accordingly, diversity helps to stabilize a link and improves performance by reducing error rate. [6]

Several different diversity modes are available and provide a number of advantages:

Time diversity: Using time diversity, a message may be transmitted at different times, e.g. using different timeslots and channel coding.

Frequency diversity: This form of diversity uses different frequencies. It may be in the form of using different channels, or technologies such as spread spectrum / OFDM.[6]

Space diversity: Space diversity used in the broadest sense of the definition is used as the basis for MIMO. It uses antennas located in different positions to take advantage of the different radio paths that exist in a typical terrestrial environment.[6]

MIMO is effectively a radio antenna technology as it uses multiple antennas at the transmitter and receiver to enable a variety of signal paths to carry the data, choosing separate paths for each antenna to enable multiple signal paths to be used.

When transmitting à signal from a transmitter to a receiver, it can take many paths. Additionally, by moving the antennas even a small distance the paths will change. The variety of paths available occurs as a result of the number of objects that appear between the transmitter and receiver. Previously these multiple paths only served to introduce interference. By using MIMO, these additional paths can be used in a positive way. They can be used to provide additional robustness to the radio link by improving the signal to noise ratio, or by increasing the link data capacity.[6]

The three main applications of MIMO are given below:

Spatial diversity: Spatial diversity used in this narrower sense often refers to transmit and receive diversity. These two methodologies are used to provide improvements in the signal to noise ratio and they are characterized by improving the reliability of the system with respect to the various forms of fading.[6]

Spatial multiplexing: This form of MIMO is used to provide additional data capacity by utilizing the different paths to carry additional traffic, i.e. increasing the data throughput capability.

Beamforming: These techniques can be used with any antenna system - not just on MIMO systems. They are used to create a certain required antenna directive pattern to give the required performance under the given conditions.

Smart antennas are normally used - these are antennas that can be controlled automatically according the required performance and the prevailing conditions.

As a result of the use multiple antennas, MIMO wireless technology is able to considerably increase the capacity of a given channel while still obeying Shannon's law. By increasing the number of receive and transmit antennas it is possible to linearly increase the throughput of the channel with every pair of antennas added to the system. This makes MIMO wireless technology one of the most important wireless techniques to be employed in recent years.

1.7 MIMO-OFDM

MIMO wireless technology in combination with orthogonal frequency division multiplexing (MIMO-OFDM) was an attractive solution for fourth-generation mobile cellular wireless systems.[7]

MIMO technology will predominantly be used in broadband systems that exhibit frequency-selective fading and, therefore, inter-symbol interference (ISI). OFDM modulation turns the frequency-selective channel into a set of parallel flat fading channels and is, hence, an attractive way of coping with ISI.[7]

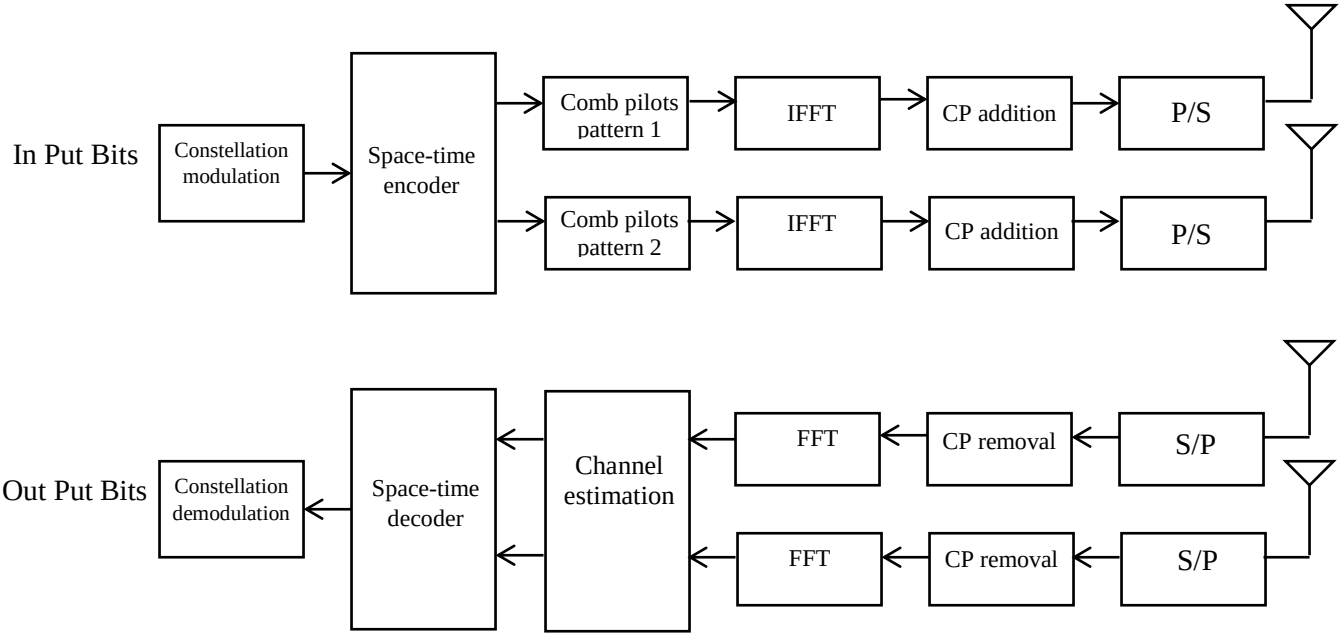


Fig.1.6 Schematic of a MIMO-OFDM system.

Fig (1.6) depicts the schematic of a MIMO-OFDM system. The basic principle that underlies OFDM is the insertion of a guard interval, called cyclic prefix (CP), which is a copy of the last part of the OFDM symbol Fig (1.6), and has to be long enough to accommodate the delay spread of the channel. The use of the CP turns the action of the channel on the transmitted signal from a linear convolution into a cyclic convolution, so that the resulting overall transfer function can be diagonalized through the use of an IFFT at the transmitter and an FFT at the receiver Fig (1.6). Consequently, the overall frequency-selective channel is converted into a set of parallel flat fading channels, which drastically simplifies the

equalization task. However, as the CP carries redundant information, it incurs a loss in spectral efficiency, which is usually kept at a maximum of 25%.[7]

1.8 Conclusion

In this chapter, we have explained the digital communications system and how the message is transmitted from the transmitter to the receiver through many wired and wireless channels. We also discussed one of the latest new technologies in the field of modulation, which is OFDM. As it contributed to utilizing the spectrum and creating more opportunities to send more data with high efficiency.

We also have presented the MIMO system, which is used to stabilize a link and improve its performances in term of error rate by using multiple antenna in both transmission and reception.

Furthermore, we have presented the combination of OFDM modulation with MIMO technics to obtain MIMO-OFDM. This combination is considered as one of the most important technologies in modern communication and it is included in many communication networks.

Chapter II

Fast Fourier Transform

2.1 Introduction

The discrete Fourier transform (DFT) is an extremely important concept that will eventually lead to the fast Fourier transform (FFT). The FFT algorithm is one of the most powerful and widely used algorithms in the digital signal processing field. This algorithm is used in a large spectrum of applications like OFDM implementation, image compression, audio compression, convolutional neural networks…etc.

Fourier transform is a mathematical transformation that can be written in terms of big matrix multiplication, the fast Fourier transform is a computationally efficient way of computing the DFT that scales to very large datasets. the basic ideas were popularized in 1965, but some algorithms had been derived as early as 1805 [8].

Several algorithms more commonly called FFT for Fast Fourier Transform, aim to accelerate the computation of the discrete Fourier transform.

Indeed, published work on the FFT algorithm as a means of calculating the DFT, by J. W. Cooley and J. W. Tukey in 1965, was a turning point in digital signal processing and in certain areas of numerical analysis. They showed that the DFT, which was previously thought to require $O(N^2)$ arithmetic operations, could be calculated by the new FFT algorithm using only $O(Nlog(N))$ operations. This algorithm had a revolutionary effect on many digital processing methods, and remains the most widely used method of computing Fourier transforms.[8]

2.2 The concept of DFT:

The N-point DFT of any signal *x(n)* is defined as follows[9]:

$$\mathbf{X}^{\boldsymbol{F}}(\boldsymbol{k}) = \sum_{\boldsymbol{n=0}}^{\boldsymbol{N-1}} \boldsymbol{x}(\boldsymbol{n}) \boldsymbol{W}_{\boldsymbol{N}}^{\boldsymbol{nk}} \qquad \mathbf{k = 0, 1, \dots, N - 1} \qquad \mathbf{II.1}$$

With *x (n)* are the input sequences and X^F *(k)* are the output sequences. The two sequences *x (n)* and X^F *(k)* are sequences of complex numbers. *N* is the length of the transform. The twiddle factors are given by:

$$\boldsymbol{W_N^{nk} = e^{-i(\frac{2\pi}{N})nk}, i^2 = -1}$$

The discrete Fourier transform is an invertible because it is linear transformation and it is called IDFT (inverse discrete Fourier transform), The inverse transform is given by[9]:

$$\boldsymbol{x(n) = \frac{1}{N}\sum_{k=0}^{N-1} X^F(k) W_N^{-nk}} \qquad \textbf{II. 2}$$

2.3 Algorithm FFT

Direct computation of an *N*-point DFT requires nearly $O(N^2)$ complex arithmetic operations. An arithmetic operation implies multiplication or addition. However, this complexity can be significantly reduced by developing efficient algorithms. These algorithms are denoted as FFT (fast Fourier transform) algorithms. Several techniques are developed for the FFT. We will initially develop the decimation-in-time (DIT) and decimation-in-frequency (DIF) FFT algorithms. The detailed development will be based on radix-2. We can then foresee that innumerable combinations of fast algorithms exist for the FFT, i.e., mixed-radix, split-radix, DIT, DIF, DIT/DIF, vector radix, vector-split-radix, etc.[9]

2.3.1. Radix 2 DIT-FFT

This algorithm is based on decomposing an N-point sequence (assume $N = 2^I$, *I* = integer) into two $N/2$-point sequences (one of even samples and another of odd samples) and obtaining the *N*-point DFT in terms of the DFTs of these two sequences. This operation by itself results in some savings of the arithmetic operations. Further savings can be achieved by decomposing each of the two $N/2$-point sequences into two $N/4$-point sequences (one of even samples and another of odd samples) and obtaining the *N/2*-point DFTs in terms of the corresponding two *N/4*-point DFTs. This process is repeated till two-point sequences are obtained [9].

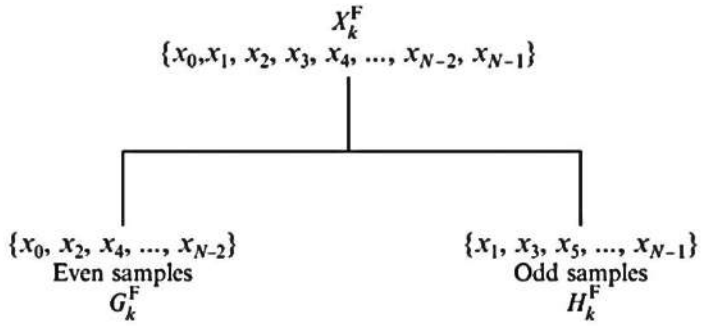


Fig.2.1 division of XF to GF and HF

Equation II.1 can be developed as follow:

$$= \sum_{\substack{n \text{ even} \\ \text{integer}}} x(n) W_N^{nk} + \sum_{\substack{n \text{ odd} \\ \text{integer}}} x(n) W_N^{nk}$$

$$= \sum_{r=0}^{\left(\frac{N}{2}\right)-1} x(2r) W_N^{2rk} + \sum_{r=0}^{\left(\frac{N}{2}\right)-1} x(2r+1) W_N^{(2r+1)k}$$

$$= \sum_{r=0}^{\left(\frac{N}{2}\right)-1} x(2r)(W_N^2)^{rk} + W_N^k \sum_{r=0}^{\left(\frac{N}{2}\right)-1} x(2r+1)(W_N^2)^{rk} \qquad \textbf{II.3a}$$

Note that

$$W_N^2 = \exp\left[\frac{-j2(2\pi)}{N}\right] = \exp\left(\frac{-j2\pi}{N/2}\right) = W_{N/2}$$

$$X^F(k) = \sum_{r=0}^{\left(\frac{N}{2}\right)-1} x(2r) W_{N/2}^{rk} + W_N^k \sum_{r=0}^{\left(\frac{N}{2}\right)-1} x(2r+1) W_{N/2}^{rk} \qquad \textbf{II.3b}$$

$$= G^F(k) + W_N^k H^F(k), \qquad k = 0,1,\ldots,\frac{N}{2}-1$$

Here $X^F(k)$ the N-point DFT of $x(n)$ is expressed in terms of N / 2-point DFTs, $G^F(k)$ and $H^F(k)$, which are DFTs of even samples and odd samples of $x(n)$ respectively.

$X^F(k)$: periodic with period $N, X^F(k) = X^F(k+N)$.

$X^F(k)$, $H^F(k)$: periodic with period $\frac{N}{2}$, $G^F(k) = G^F(k+\frac{N}{2})$ and $H^F(k) = H^F(k+\frac{N}{2})$.

$$X^{\mathrm{F}}(k) = G^{\mathrm{F}}(k) + W_N^k H^{\mathrm{F}}(k),\ k = 0,1,\dots,\frac{N}{2} - 1 \qquad \textbf{II.4a}$$

$$X^{\mathrm{F}}(k + \frac{N}{2}) = G^{\mathrm{F}}(k) + W_N^{k+\frac{N}{2}} H^{\mathrm{F}}(k)$$

$$W_N^{N/2} = \exp\left(\frac{-j2\pi}{N}\frac{N}{2}\right) = \exp(-j\pi) = -1$$

Since $W_N^{k+N/2} = \exp\left[\frac{-j2\pi}{N}\left(k + \frac{N}{2}\right)\right] = W_N^k W_N^{N/2} = -W_N^k$, , it follows that

$$X^{\mathrm{F}}\left(k + \frac{N}{2}\right) = G^{\mathrm{F}}(k) - W_N^k H^{\mathrm{F}}(k), \qquad k = 0,1,\dots,\frac{N}{2} - 1 \qquad \textbf{II.4b}$$

Equation (II.4) is shown below as a butterfly Fig (2.2)

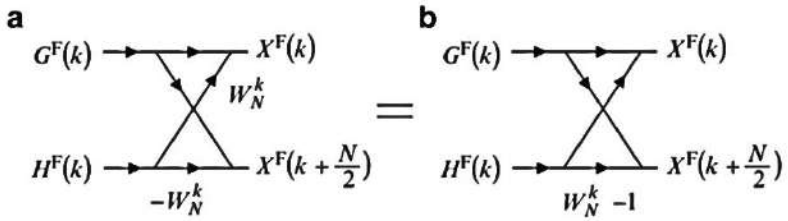


Fig.2.2 Butterfly to compute (II.4)

For each k, Fig (2.2) a requires two multiplies and two adds whereas Fig (2.2b) requires one multiply and two adds. Repeat this process iteratively till two-point DFTs are obtained.

Let $N = 16$. Direct computation of a 16-point DFT requires $N^2 = 256$ adds and multiplies. Through $G^{\mathrm{F}}(k)$ and $H^{\mathrm{F}}(k)$, it requires only 128 + 16 = 144 adds and multiplies, resulting in saving of 112 adds and multiplies. Additional savings can be achieved by decomposing the eight-point DFTs into two four-point DFTs and finally into four two-point DFTs. $G^{\mathrm{F}}(k)$ and $H^{\mathrm{F}}(k)$ are eight-point DFTs. Each requires 64 adds and multiplies. The algorithm based on this technique is called radix-2 DIT-FFT [9].

The radix-2 DIT-FFT algorithm is illustrated below for N = 8.

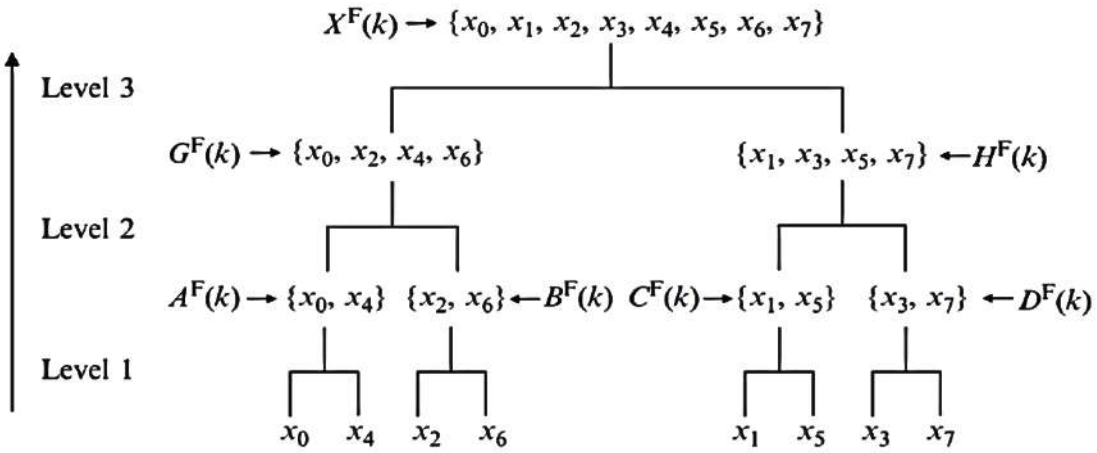


Fig.2.3 Division of the DIF-FFT algorithm with N = 8

$G^F(k)$ *is DFT of* $\{x_0, x_2, x_4, x_6\}$

$A^F(k)$ is DFT of $\{x_0, x_4\}$

$B^F(k)$ is DFT of $\{x_2, x_6\}$

$H^F(k)$is DFT of $\{x_1, x_3, x_5, x_7\}$

$C^F(k)$is DFT of $\{x_1, x_5\}$

$D^F(k)$is DFT of $\{x_3, x_7\}$

Fig (2.4) shows Radix 2 FFT DIT decomposition concept for $N = 8$ [9].

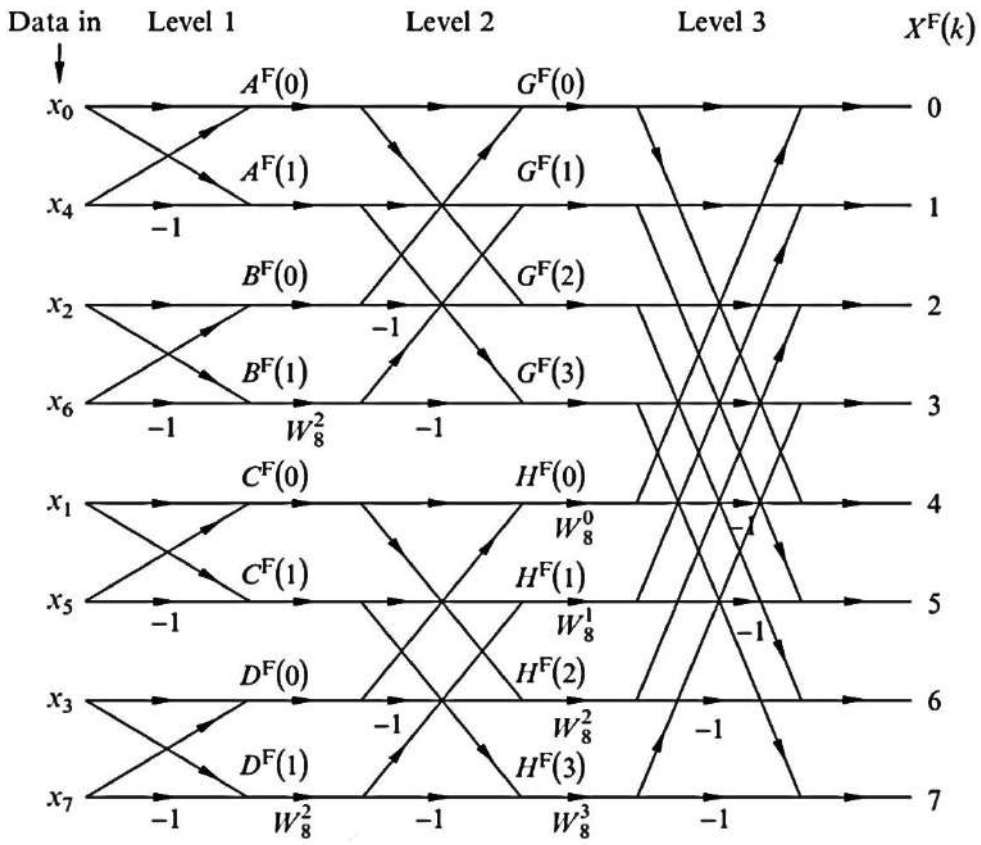


Fig.2.4 Radix 2 DIT FFT flow graph for N = 8

2.3.2. Radix 2 DIF-FFT

Similar to radix-2 DIT-FFT radix-2 DIF-FFT can be developed as follows:

$$X^F(k) = \sum_{n=0}^{\left(\frac{N}{2}\right)-1} x(n) W_N^{nk} + \sum_{n=\frac{N}{2}}^{N-1} x(n) W_N^{nk} \qquad \textbf{II.5}$$

let $n = m + N/2$, the second summation changes to:

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

Hence (**II.5**) becomes

$$X^{\mathrm{F}}(k) = \sum_{n=0}^{(N/2)-1} x\,(n)W_N^{nk} + W_N^{(N/2)k} \sum_{n=0}^{(N/2)-1} x\left(n+\frac{N}{2}\right) W_N^{nk}, \quad k = 0,1,\dots,N-1$$

$$X^{\mathrm{F}}(k) = \sum_{n=0}^{\left(\frac{N}{2}\right)-1} \left[x(n) + (-1)^k x\left(n+\frac{N}{2}\right)\right] W_N^{nk},\ k = 0,1,\dots,N-1 \qquad \textbf{II.6}$$

For k = even integer = $2r$, we can write:

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

And for k = odd integer = $2r + 1$:

$$X^{\mathrm{F}}(2r+1) = \sum_{n=0}^{\left(\frac{N}{2}\right)-1} \left[x(n) - x\left(n+\frac{N}{2}\right)\right] W_N^{n} W_N^{2nr} \qquad \textbf{II.8}$$

Since $W_N^{2nr} = \exp\left(\frac{-j2\pi 2nr}{N}\right) = \exp\left(\frac{-j2\pi nr}{N/2}\right) = W_{N/2}^{nr}$ then

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

$X^{F}(2r)$ is N / 2-point DFT of $\left[x(n) + x\left(n+\frac{N}{2}\right)\right],\ r,n = 0,1,\dots,\frac{N}{2}-1$ **II.9a**

Similarly

$X^{F}(2r+1)$ is N / 2-point DFT of $\left[x(n) - x\left(n+\frac{N}{2}\right)\right] W_N^{n},\ r,n = 0,1,\dots,\frac{N}{2}-1$ **II.9b**

Fig (2.5) shows the butterfly structure to compute (II.9) for $n = 0,1,\dots,\frac{N}{2}-1$

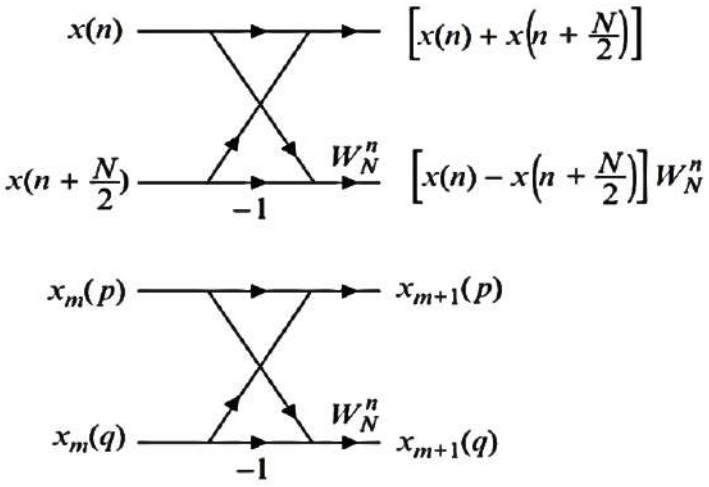


Fig.2.5 Butterfly structure for Radix-2 DIF algorithm.

The N-point DFT can be calculated from the two *N*/2-point DFTs as described in equation II.9. This results in reduction of computational complexity as in the case of DIT-FFT algorithm. Further savings can be achieved by successively repeating this decomposition (i.e. first half of the samples and second half of the samples). The DIF-FFT algorithm for for N = 8 is described as follow.

For N = 8, (II.9) becomes

$$X^{\mathrm{F}}(2r)=\sum_{n=0}^{3}\left[x(n)+x\left(n+\frac{N}{2}\right)\right]W_4^{nr},\qquad r=0,1,2,3 \qquad \textbf{II.10a}$$

$$X^{F}(2r+1)=\sum_{n=0}^{3}\left[x(n)-x\left(n+\frac{N}{2}\right)\right]W_8^{n}W_4^{nr},\qquad r=0,1,2,3 \qquad \textbf{II.10b}$$

Fig (2.6) shows the SFG of this first step of decomposition.

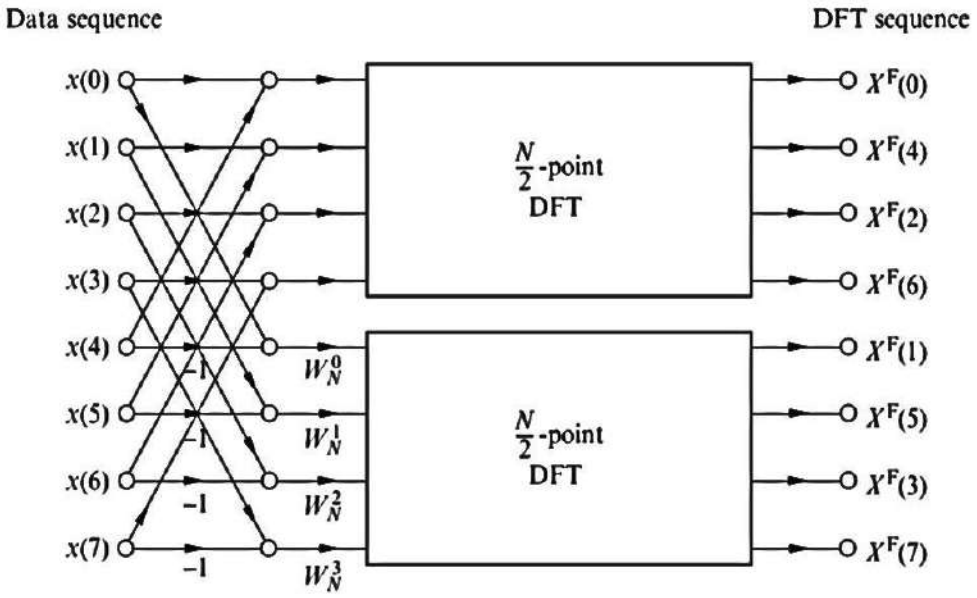


Fig.2.6 Flow graph of decimation-in frequency decomposition of an *N*-point DFT into two N/2-point DFT computations ($N = 8$) ($W_N = W_8$)

Fig (2.7) shows the SFG of the second step of decomposition where every 4-points DFT is decomposed into two 2-points DFT.

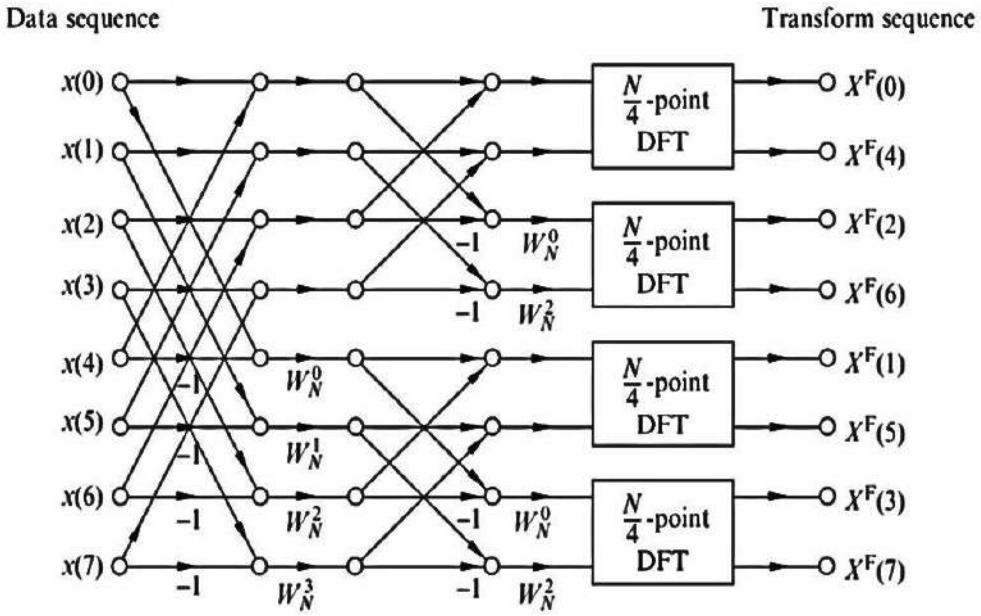


Fig.2.7 Flow graph of decimation-in frequency decomposition of an eight-point DFT into two-point DFT computations. WN = W8

Hence the eight-point DIF-FFT is obtained in three stages as illustrated in Fig (2.8) [9].

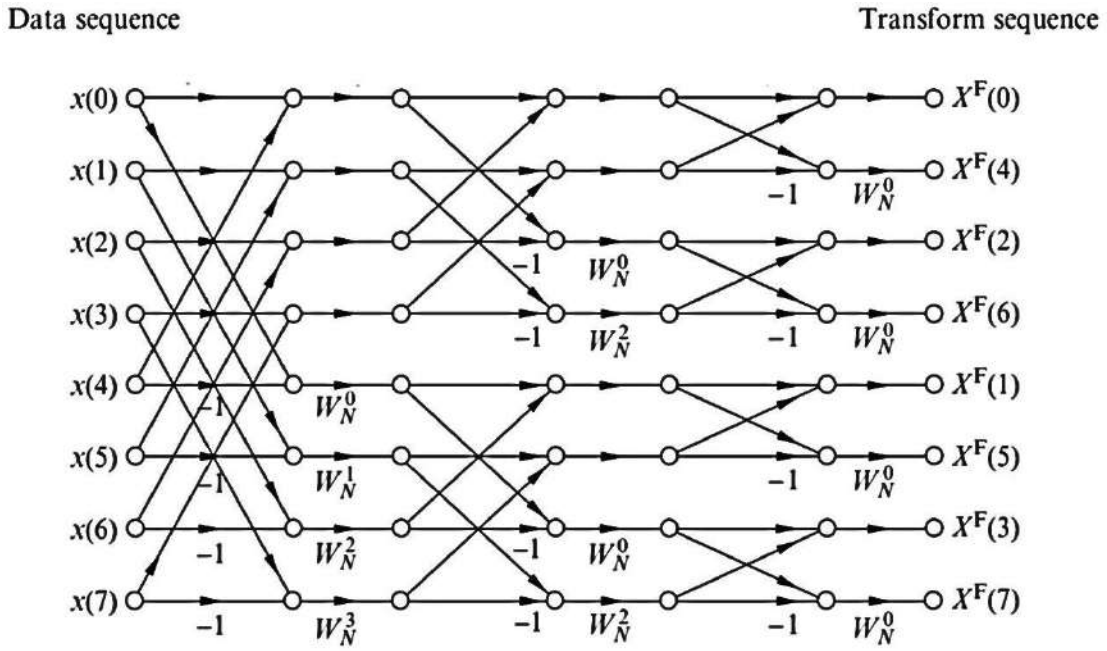


Fig.2.8 Flow graph of complete decimation-in frequency decomposition of an eight-point DFT computation. WN = W8

The Radix-2 is the simplest FFT algorithm to implement. Several works are carried out on this algorithm in order to find more elaborate methods to reduce the number of complex multiplications and additions.

The radix-4 FFT algorithm is a decomposition of the discrete Fourier transform algorithm into 4 independent transforms of size N / 4. It aims to reduce the multiplicative complexity of DFT by 25% compared to Radix-2. Provided that the size of the FFT is to a power of 4, We can take advantage of trivial multiplications by 1, -1, j, and -j which do not have to be performed by a real multiplication.[9]

2.3.3. Radix 2^2 Algorithm

For radix 2^2 algorithm we consider consider the following three-dimensional mapping of *n* and *k:*

$$n = < \frac{N}{2} n_1 + \frac{N}{4} n_2 + n_3 >_N \qquad \text{II.11}$$

$$k = < k_1 + 2k_2 + 4k_3 >_N \qquad \text{II. 12}$$

Where the notation $< >_N$ represents the modulo-N operator. Substituting equations (II.11) and (II.12) into equation (II.1) obtains:

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

$$\sum_{n_3=0}^{\frac{N}{4}-1} \sum_{n_2=0}^{1} \sum_{n_1=0}^{1} x\left(\frac{N}{2}n_1 + \frac{N}{4}n_2 + n_3\right) W_N^{\left(\frac{N}{2}n_1 + \frac{N}{4}n_2 + n_3\right)(k_1 + 2k_2 + 4k_3)} \quad \text{II. 13}$$

Summing among n_1 and considering:

$$B_{\frac{N}{2}}\left(\frac{N}{4}n_2 + n_3, k_1\right) = x\left(\frac{N}{4}n_2 + n_3\right) + (-1)^{k_1} x\left(\frac{N}{4}n_2 + n_3 + \frac{N}{2}\right) \quad \text{II. 14}$$

Equation (II.13) can be re-formulated as follows:

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

$$\sum_{n_3=0}^{\frac{N}{4}-1} \sum_{n_2=0}^{1} \left[B_{\frac{N}{2}}\left(\frac{N}{4}n_2 + n_3, k_1\right) W_N^{\left(\frac{N}{4}n_2 + n_3\right)k_1}\right] W_N^{\left(\frac{N}{4}n_2 + n_3\right)(2k_2 + 4k_3)} \quad \text{II. 15}$$

The critical concept of Radix-2^2 is to cascade the twiddle factor $W_N^{\left(\frac{N}{4}n_2 + n_3\right)k_1}$ into the next step decomposition instead of multiplying with $B_{\frac{N}{2}}\left(\frac{N}{4}n_2 + n_3, k_1\right)$

$$W_N^{\left(\frac{N}{4}n_2 + n_3\right)k_1} W_N^{\left(\frac{N}{4}n_2 + n_3\right)(2k_2 + 4k_3)} = (-j)^{n_2(k_1 + 2k_2)} W_N^{n_3(k_1 + 2k_2)} W_N^{4n_3k_3} \quad \text{II. 16}$$

Substituting (II.16) into (II.15) and summing among n_2 determine:

$$X(k_1 + 2k_2 + 4k_3) = \sum_{n_3=0}^{\frac{N}{4}-1} \left[H_{\frac{N}{4}}(n_3, k_1, k_2) W_N^{n_3(k_1 + 2k_2)}\right] W_{\frac{N}{4}}^{n_3k_3} \quad \text{II. 17}$$

Where $H_{\frac{N}{4}}(n_3, k_1, k_2)$ can be expressed as follows:

$$H_{\frac{N}{4}}(n_3, k_1, k_2) = B_{\frac{N}{2}}(n_3, k_1) + (-j)^{(k_1 + 2k_2)} B_{\frac{N}{2}}\left(n_3 + \frac{N}{4}, k_1\right) \quad \text{II. 18}$$

Equation (II.14) represents the first butterfly structure containing only addition and subtraction operations. Equation (II.18) demonstrates the second butterfly structure containing addition, subtraction and multiplication with $-j$.

The Radix-2^2 algorithm requires full multiplication every two stages as demonstrated in equation (II.17), opposed to multiplication occurring during every stage within Radix-2 [10].

Hence, Radix-2^2 demonstrates a 50% reduction of multiplication operations required in comparison to Radix-2. The complete Radix-2^2 algorithm can be obtained through decomposing equation (II.17) recursively to the remaining *N/4* points DFTS.

The fundamental structure to obtain two consecutive stages in the Signal Flow Graph (SFG) for Radix-2^2 Decimation In Frequency (DIF) algorithm is illustrated in Fig (2.9), where $x_i(.)$ denotes the inputs of stage i and $X_{i+1}(.)$ denotes the outputs of stage $i+1$. When n is varied from 0 to $Ni/4-1$ in the basic structure of Fig (2.9) a group of N_i inputs is formed, where N_i is given by:

$$N_i = \frac{N}{2^{(i-1)}} = 2^{n_s - i + 1} \qquad \text{II. 19}$$

The group is repeated N/N_i times to obtain two consecutive stages in the complete SFG.

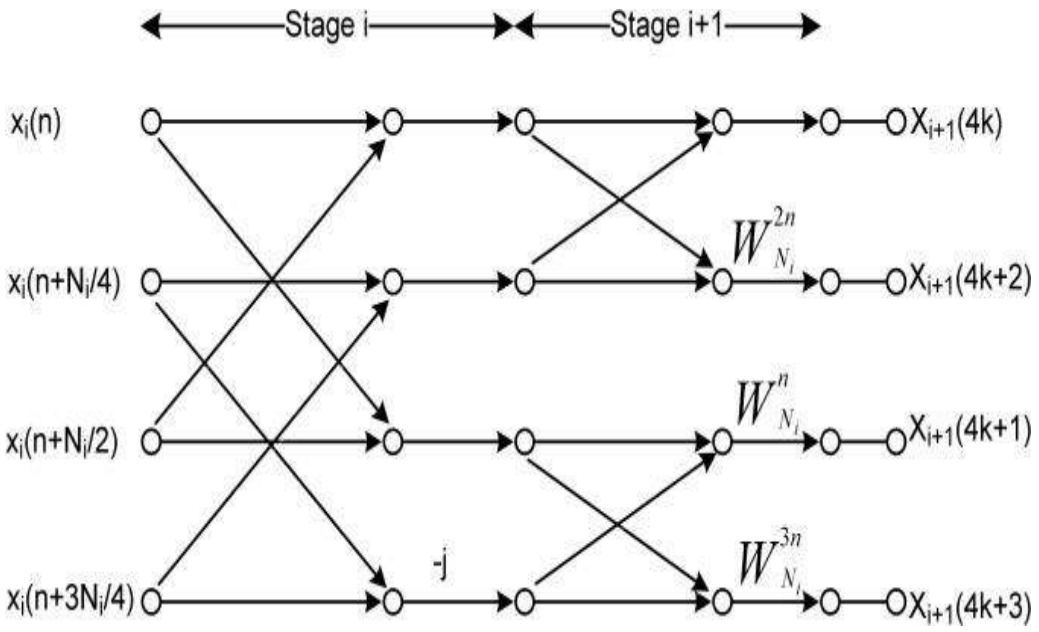


Fig.2.9 SFG of the basic butterfly structure of Radix-2^2 DIF algorithm

2.3.4. Radix 2^3 Algorithm

Applying four-dimensional linear index mapping to *n* and *k* can be expressed as follows:

$$n =< \frac{N}{2}n_1 + \frac{N}{4}n_2 + \frac{N}{8}n_3 + n_4 >_N \qquad \text{II. 20}$$

$$k =< k_1 + 2k_2 + 4k_3 + 8k_4 >_N \qquad \text{II. 21}$$

The DFT function in equation (II.1) can be expressed in terms of four-dimensional linear index mapping, as follows:

$$X(k_1 + 2k_2 + 4k_3 + 8k_4) =$$

$$\sum_{n_4=0}^{\frac{N}{8}-1}\sum_{n_3=0}^{1}\sum_{n_2=0}^{1}\sum_{n_1=0}^{1} x\left(\frac{N}{2}n_1 + \frac{N}{4}n_2 + \frac{N}{8}n_3 + n_4\right)W_N^{nk} \qquad \text{II. 22}$$

Additionally, the twiddle factor $W_N^{nk}$with cascade decomposition can be expressed as:

$$W_N^{nk} = W_N^{\frac{N}{2}n_1k_1}W_N^{\frac{N}{4}n_2(k_1+2k_2)}W_N^{\frac{N}{8}n_3(k_1+2k_2+4k_3)}W_N^{n_4(k_1+2k_2+4k_3+8k_4)} \qquad \text{II. 23}$$

$$= (-1)^{n_1k_1}(-j)^{n_2(k_1+2k_2)}W_N^{\frac{N}{8}n_3(k_1+2k_2+4k_3)}W_N^{n_4(k_1+2k_2+4k_3)}W_{\frac{N}{8}}^{n_4k_4}\Big)$$

Substitution of equation (II.23) into equation (II.24) and expending summation among n_1, n_2 and n_3 determine:

$$X(k_1 + 2k_2 + 4k_3 + 8k_4) =$$

$$\sum_{n_4=0}^{\frac{N}{8}-1}\left[T_{\frac{N}{8}}(n_4, k_1, k_2, k_3)W_N^{n_4(k_1+2k_2+4k_3)}\right]W_{\frac{N}{8}^{n_4}}{}_{k_4} \qquad \text{II. 24}$$

Eight DFT functions of length $\frac{N}{2}$ were expressed in equation (II.24), where a third butterfly structure in $T_{\frac{N}{8}}(n_4, k_1, k_2, k_3)$ is expressed as follows:

$$T_{\frac{N}{8}}(n_4, k_1, k_2, k_3) =$$

$$H_{\frac{N}{4}}(n_4,k_1,k_2)+W_N^{\frac{N}{8}(k_1+2k_2+4k_3)}H_{\frac{N}{4}}\left(n_4+\frac{N}{8},k_1,k_2\right)=$$

$$H_{\frac{N}{4}}(n_4,k_1,k_2)+\left(\frac{\sqrt{2}}{2}(1-j)\right)^{k_1}(-j)^{(k_2+2k_3)}H_{\frac{N}{4}}\left(n_4+\frac{N}{8},k_1,k_2\right) \qquad \text{II. 25}$$

The third butterfly structure involves the twiddle factor $\left(\frac{\sqrt{2}}{2}(1-j)\right)^{k_1}(-j)^{(k_2+2k_3)}$ which can be realized with constant scalar operations. The initial radix-2^3 SFG stage contains only trivial multiplication with $-j$, the second stage multiplies with $-j$ and constants W_8^1 W_8^3 while the third stage contains multiplications with the general term [10].

$$W_N^{n_4(k_1+2k_2+4k_3)}$$

In comparison, Radix-2 required multiplication during every stage, while Radix-2^2 only uses multiplication operations every second stage.

The fundamental structure to obtain three consecutive stages in the SFG of Radix-2^3 DIF algorithm is presented in Fig (2.10) .A group of *Ni* inputs is obtained when *n* varies from *0* to N_i */8-1*. Three consecutive stages in the complete SFG are formed by N/N_i groups.

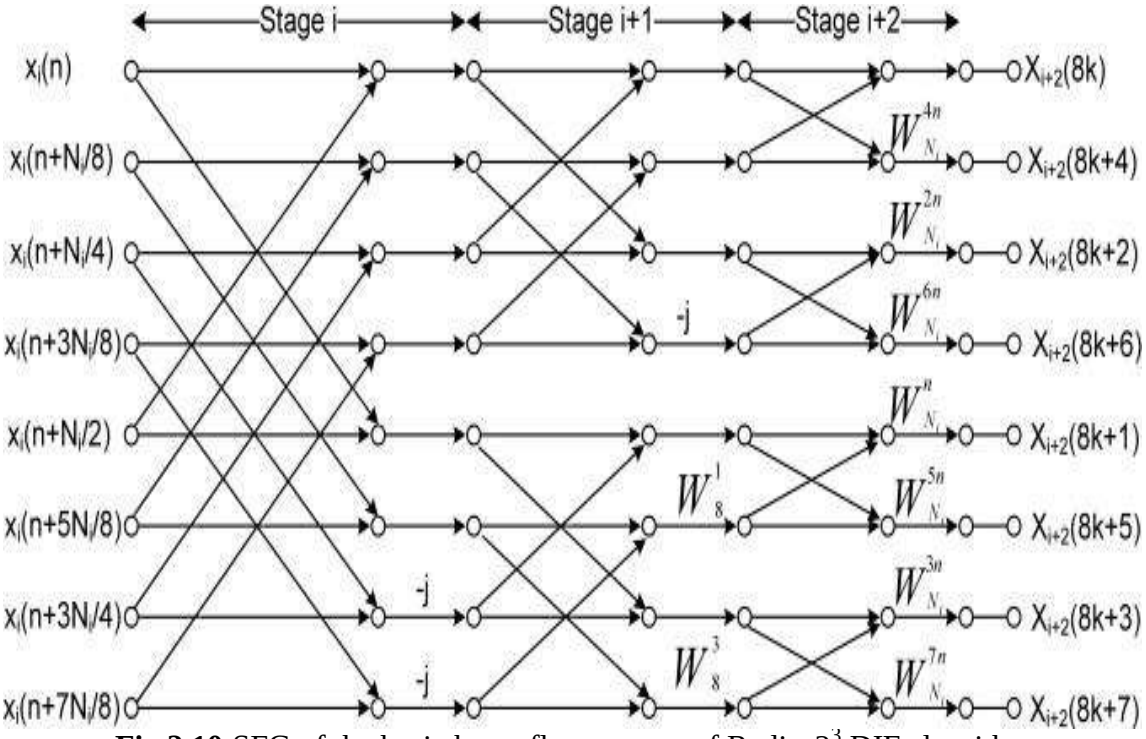


Fig.2.10 SFG of the basic butterfly structure of Radix-2^3 DIF algorithm

2.4 Bit-reversal permutation

In applied mathematics, a bit-reversal permutation is a permutation of a sequence of n items, where $N = 2^n$ is a power of two. It is defined by indexing the elements of the sequence by the numbers from 0 to $N - 1$ and then reversing the binary representations of each of these numbers (padded so that each of these binary numbers has length exactly n). Each item is then mapped to the new position given by this reversed value. The bit reversal permutation is an involution, so repeating the same permutation twice returns to the original ordering on the items.

Table 2.1 An example of bit reversed order for $N = 8$

Natural order	Binary	Bit reverse	Bit reversed order
0	000	000	0
1	001	100	4
2	010	010	2
3	011	110	6
4	100	001	1
5	101	101	5
6	110	011	3
7	111	111	7

2.5 Pipeline architecture for FFT implementation

The pipeline architecture is one of the most popular and most suitable architectures for signal processing applications where high data throughput is the dominant condition.

Since then, the pipeline architecture offers a good compromise between hardware complexity/data processing rate for high-speed communications systems. Fig (2.11) represents a basic pipeline architecture.[11]

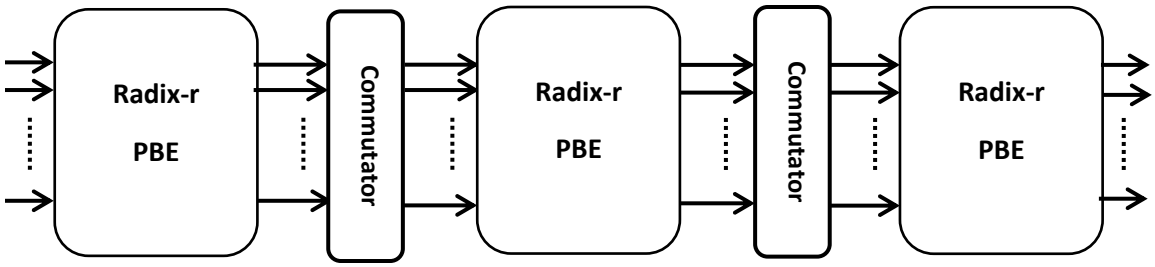


Fig.2.11 General structure of pipeline architecture

Pipeline architectures utilize parallel processing among the stages. These structures consist of one butterfly element between commutators at each stage. The function of butterfly element is to compute the data addition, subtraction and multiplication. The commutator is like a switch to rearrange the data from the butterfly element in order to perform subsequent calculations more conveniently.[11]

Pipeline architectures are widely used to attain high-speed operation, they can be roughly classified into three types:

- Single path Delay Feedback (SDF),
- Multipath Delay Feedback (MDF),
- Multiple path Delay Commutator (MDC),
- Single path Delay Commutator (SDC).

2.5.1. Single Path Delay Feedback (SDF) Architecture

Fig (2.12) is 16-point radix-2 single path delay feedback architecture. The procedural flow of this architecture is as follows.

a) In the beginning, the input data with indices 0 to 7 are stored in the shift register (D8 in Fig (2.12). The radix-2 butterfly elements operate on these data and on the remaining input data with indices 8 to 15.

b) The data resulting from the butterfly addition operations are passed to the second stage and the subtraction results are fed back to the shift register (D8).

c) After all the 8-point data addition results are output, the subtraction data from the registers are passed to the second stage with a multiply twiddle factor coefficient. The symbol A in Fig (2.12) represents the last output data of the first stage.

d) The procedure flow of the latter three stages for the 16-point R2SDF are similar to the first stage.[11]

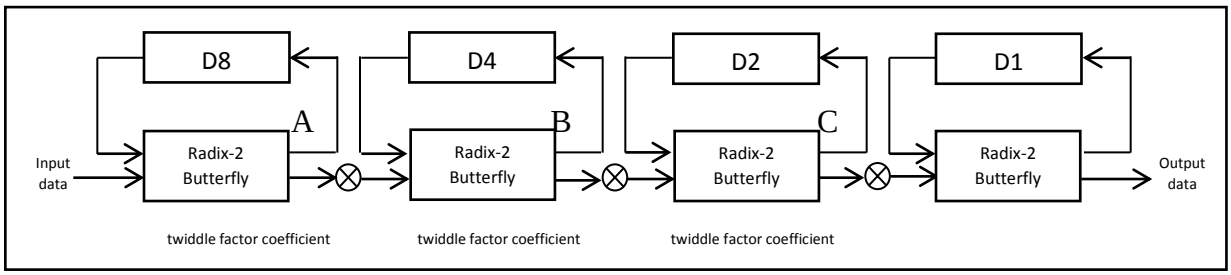


Fig.2.12 A 16- point radix-2 single path delay feedback architecture

2.5.2. Multiple Path Delay Commutator (MDC) Architecture

In the general multiple path delay commutator architecture, the input data sequences are organized into a parallel data path entering the commutator by suitable delays and data summation is performed by the butterfly element at each stage. The intermediate data is not fed back to register. It has a higher throughput data rate than the single path delay feedback architecture.[11]

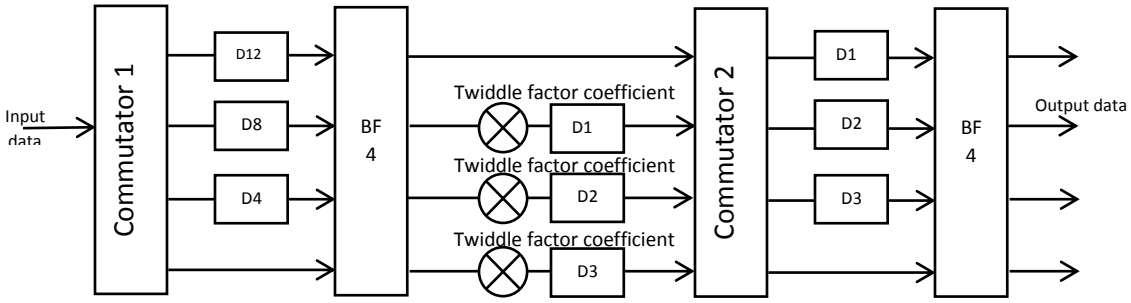


Fig.2.13 A 16-point radix-4 multiple path delay commutator architecture

Here, we give an example to illustrate the principles of the multiple path delay commutator architecture. The architecture of a 16-point Radix-4 multiple path delay commutator architecture (R4MDC) is as shown in Fig (2.13). The workflow for this architecture is

a) In the beginning, the input data from index 0 to 15 is decomposed into four parallel data streams of four points each after the first commutator.

b) These data are passed through different delay registers.

c) After the delay, the resulting data are operated by radix-4 butterfly processing elements (BF4) and then multiplied by a twiddle factor.

d) The result data are stored in the delay register.

e) The function of the second commutator is to reorder the desired adjacent data.

f) Before the second butterfly operation, the ordered data are stored in an appropriate delay register. The first parallel data stream must be shifted by 3 delays. The second parallel data stream must be shifted by 2 delays. The third parallel data stream must be shifted by 1 delay and the last data stream is directly passed to the butterfly input in the second stage.[11]

2.5.3. Single Path Delay Commutator (SDC) Architecture

The single path delay commutator (SDC) architecture is based on the modified multiple path delay commutator. Each stage produces a single output rather than 4 outputs as in the multiple path delay commutator. Each stage requires one complex multiplier, a delay commutator to correct the order of the data, and a butterfly element. Here, we give an example to illustrate the principle of single path delay commutator architecture. The architecture of a 16-point radix-4 single path delay commutator (R4SDC) is as shown in Fig (2.14). The multiplexer control parameters C_1,C_2 and C_3 select the required data.[11]

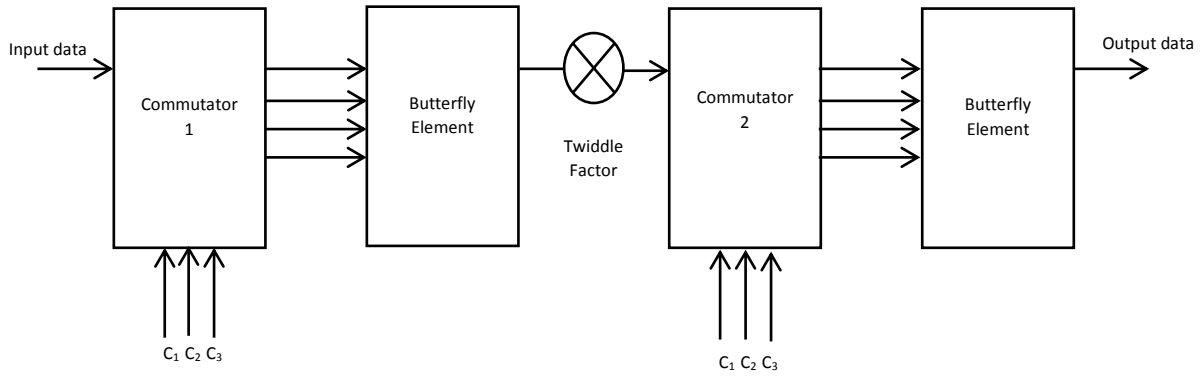


Fig.2.14 A 16-point radix-4 single path delay commutator architecture

2.6 Comparison

In this part, we summarize different architectures with different hardware requirements. The radix-r MDC architecture is faster *r* times than the radix-r SDF architecture. However in terms of memories, the SDF architecture is more economical than the MDC architecture. In terms of resources, the Radix-MDC and Radix-SDF architectures perform better than any other architecture. They always require the least amount of arithmetic resources and memories. This is due among other things to the fact that they use several multipliers with constant values optimized and characterized by a lower complexity. In terms of control complexity, the architectures for radix-4-SDC and split-radix are the most complex to control.[12]

Table 2.2 Comparison hardware of different pipelined architecture used.[12]

Architecture	Multipliers	Adders	Memory	Control
R2 MDC	$2 \log_4 N - 1$	$4 \log_4 N$	$3N/2 - 2$	Simple
R2 SDF	$2 \log_4 N - 1$	$4 \log_4 N$	$N - 1$	Simple
R4 MDC	$3 \log_4 N - 1$	$8 \log_4 N$	$5N/2 - 2$	Simple
R4 SDF	$\log_4 N - 1$	$8 \log_4 N$	$N - 1$	Average
R4 SDC	$\log_4 N - 1$	$3 \log_4 N$	$2N - 2$	Complex

Table 2.3 Resource utilization rate of the different architectures.[11]

Architecture	**Utilization of Multipliers**	**Utilization of Adders**	**Utilization of Registers**
R2SDF	50 %	50 %	100 %
R4SDF	25 %	25 %	100 %
R4MDC	25 %	25 %	25 %
R4SDC	75 %	100 %	100 %

2.7 Conclusion

In this chapter we discussed the FFT algorithm and its use to compute the DFT. The FFT has certainly optimized the solution process method by transform methods, therefor it reduces considerably the number of arithmetic operations compared to the DFT. Hence, the FFT becomes indispensable in the modern communications systems.

Pipeline architectures are the most effective way to implement FFT algorithm for digital communications with high data rates. Among the various piping algorithms, the (SDF) architecture is better at implementing of FFT in terms of resource utilization rate where MDC architecture is more suitable for high throughput applications.

Chapter III

Overview of FPGAs Circuit

3.1 Introduction

In this chapter, we will provide an overview of Field Programmable Gate Arrays (FPGA) circuit, its types, and how they work. And also we will give a look at how to program it and that to see its advantages and disadvantages.

3.2 Definition of FPGA

Field Programmable Gate Arrays (FPGAs) are semiconductor devices based on a matrix of configurable logic blocks (CLBs) connected via programmable interconnects. FPGAs can be reprogrammed to the desired application or functionality requirements after manufacturing. This feature distinguishes FPGAs from Application Specific Integrated Circuits (ASICs), which are custom manufactured for specific design tasks. Although one-time programmable (OTP) FPGAs are available, the dominant types are SRAM based which can be reprogrammed as the design evolves. [13]

The FPGA configuration is generally specified using a hardware description language (HDL).

3.3 Architecture and Component of FPGA

A basic FPGA architecture Fig (3.1) consists of thousands of fundamental elements called configurable logic blocks (CLBs) surrounded by a system of programmable interconnects, called a fabric, that routes signals between CLBs, Input/output (I/O) blocks. [14]

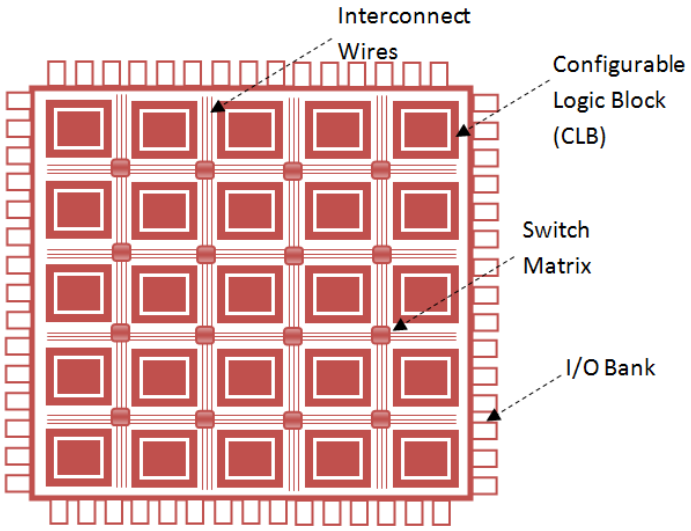


Fig 3.1 FPGA architecture [14]

The general FPGA architecture consists of three types of modules, the function of each of them are discussed below:

3.3.1. CLB

The Logic Block in Xilinx based FPGAs are called Configurable Logic Blocks or CLB while the similar structures in Altera based FPGAs are called Logic Array Blocks or LAB. [15]

Logic Block contains MUX (Multiplexer), D flip flop, and LUT as shown in the Fig (3.2). Both Xilinx and Altera use Look-up Table LUT.

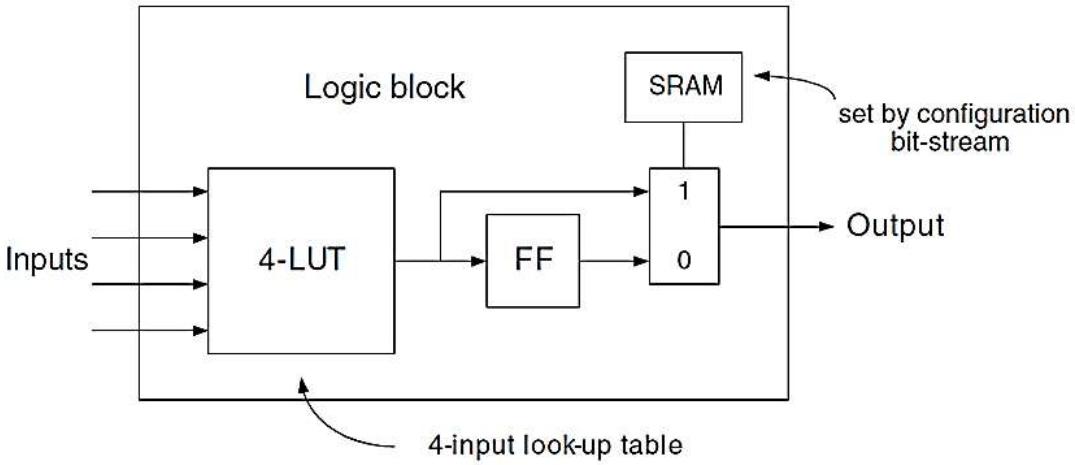


Fig 3.2 Configurable logic block

- Lookup table LUT: to implement the combinational logic functions. The output from the lookup table is given as inputs to both multiplexer and D flip flop
- Register D flip flop: to store and delay the output from the Lookup Table.
- Multiplexer: selects LUT output or delayed output.

A Logic Block can be made up of a single Basic Logic Element or a set of interconnected Basic Logic Elements, where a Basic Logic Element is a combination of a Look-up table and a Flip-flop.

3.3.2. Input/output Blocks

I/O Blocks consists of interfacing pins which helps in connecting Logic blocks to external components. They are located at the periphery of the grid and also connected to the Interconnection Matrix.

3.3.3. Switching Matrix/Interconnection Wires

The function of routing the signals and providing connections between the Logic Blocks and I/O Blocks is the task of the Interconnection Matrix. It consists of wires and switches to establish the connection within FPGA.

3.4 FPGA Programming Technologies

There are three of well-known programming technologies used in FPGAs: SRAM, Flash, and Anti-Fuse.

3.4.1. SRAM-Based Programming Technology

We know that there are two types of semiconductor RAM called the SRAM and DRAM.

SRAM is designed using transistors and the term static means that the value loaded on a basic SRAM Memory Cell will remain the same until deliberately changed or when the power is removed. SRAM cells are mainly used for following purposes:

- To program the routing interconnect of FPGAs which are generally steered by small multiplexors.
- To program CLBs those are used to implement logic functions.

This is in contrast to the DRAM, which consists of a combination of a transistor and a capacitor. The term Dynamic refers to the fact that the value loaded in the basic DRAM Memory Cell is valid until there is charge in the capacitor.

Many FPGA vendors implement Static Memory Cells in SRAM based FPGAs for programming. SRAM based FPGAs are used to program both the logic cells and the interconnects and they have become quite predominant due to their re-programmability and use of complementary metal-oxide-semiconductor (CMOS) technology, which is known for its low dynamic power consumption, high speed and tighter integration. [11]

3.4.2. Flash Programming Technology

A close alternative to SRAM based programming technology is based on Flash programming technologies. The main advantage of flash-based programming is its non-volatile nature. Even though flash supports re-programmability, the number of times this can be done is very small when compared to an SRAM technology.

3.4.3. Anti-fuse Programming Technology

The anti-fuse programming technology is an old technique of producing one-time programmable devices. They are implemented using a link called the anti-fuse, which in its unprogrammed state has a very high resistance and can be considered an open circuit (an insulator with very high resistance) linking.

3.5 FPGA design flow

FPGA architectural design flow comprises design entry, logic synthesis, design implementation, device programming, and design verification. However, the exact steps vary with manufacturers.[16]

A simplified version of the FPGA design flow is given in the flowing diagram. Fig (3.3)

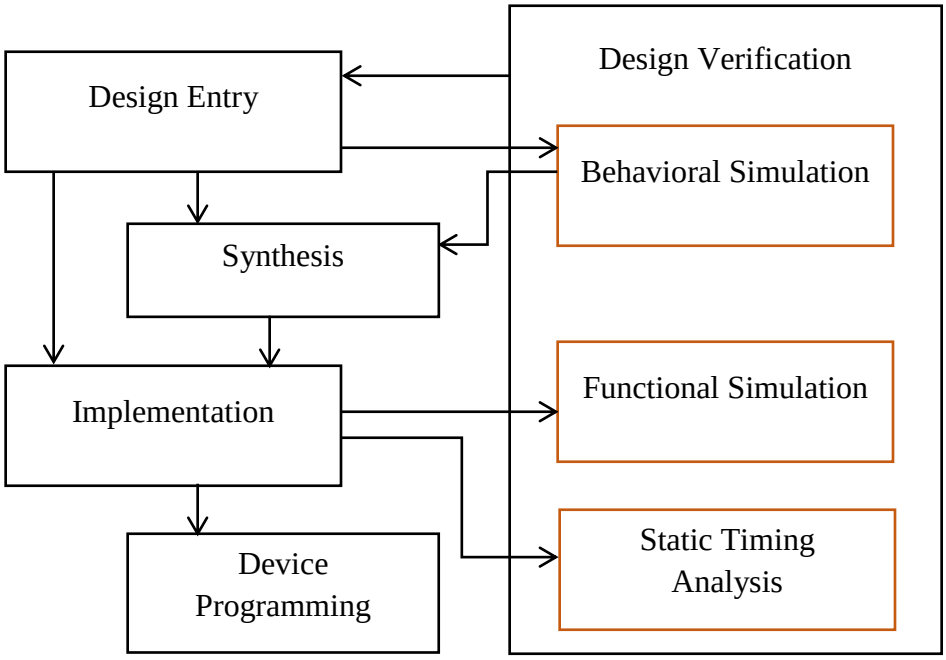


Fig 3.3 FPGA Architectural Design Flow

3.5.1. Design Entry

There are different techniques for design entry. Schematic based, Hardware Description Language or combination of both, Selection of a method depends on the design and designer

When designs with a very large amount of function become difficult to manage graphically, HDL may be used to capture the design either in a structural or in a behavioral way. Besides VHDL and Verilog, which are the most established HDLs, another method but rarely used is state-machines. It is the finite state machine FSM editor, it is the better choice for the designers who think the design as a series of states. But the tools for state machine entry are limited.[16]

3.5.2. Logic Synthesis

Synthesis is the process that translates VHDL or Verilog code into a device netlist format for depicting a complete circuit with logical elements (gates, flip flop, etc...). If the design contains more than one sub designs, to implement a processor, we need a CPU as one design element and RAM as another and so on, Synthesis involves checking the code syntax and analyzing the hierarchy of the design architecture. Next, the code is compiled along with optimization, and the

generated netlist is saved as an NGC(Native Generic Circuit) file for Xilinx Synthesis Technology XST.[16]

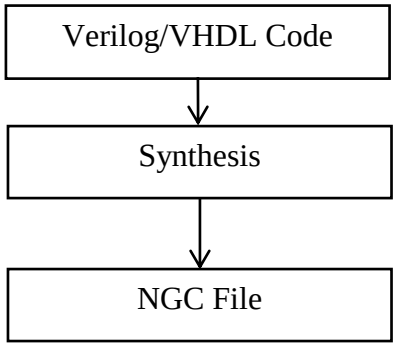


Fig 3.4 Xilinx Synthesis process

3.5.3. Implementation

The design implementation process consists of the following steps:

Synthesis: This process combines all the input netlists into the logic design file which is saved as an NGD (Native Generic Database) file, this can be done using the NGD Build program. Here user constraints file assigns the ports to the physical elements (ex. pins, switches, buttons …) of the targeted device and specifying time requirements of the design. This information is stored in a file named UCF (User Constraints File).

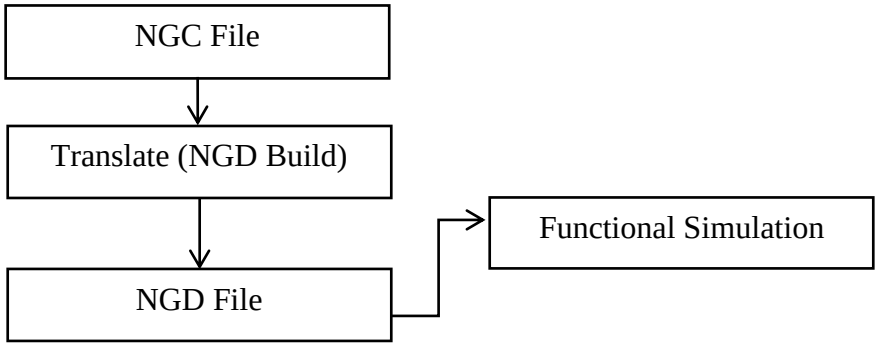


Fig 3.5 FPGA Translate process

Map: This involves mapping the logic defined by the NGD file into the components of FPGA (CLB, IOB) and then generating an NCD (Native Circuit Description) file which physically represents the design mapped to the components of FPGA.

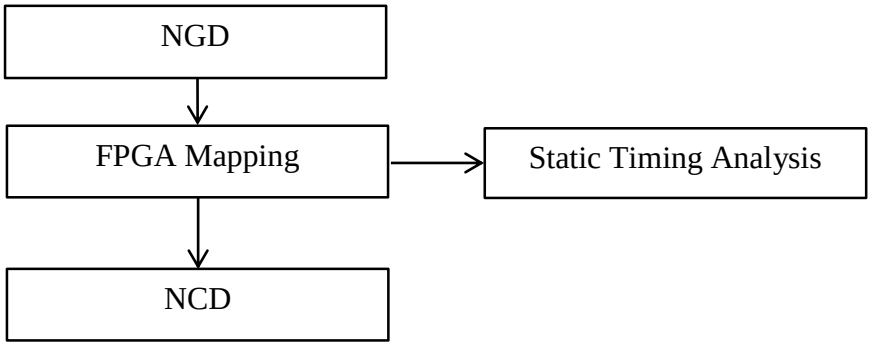


Fig 3.6 FPGA Mapping with Static Timing Analysis

Place and Route: Here routing places the sub-blocks from the above process into the logic blocks according to the constraints and then connect those blocks. The PAR tool takes the mapped NCD file as input and produces a completely routed NCD file as output. Output NCD file consists of the routing information.[16]

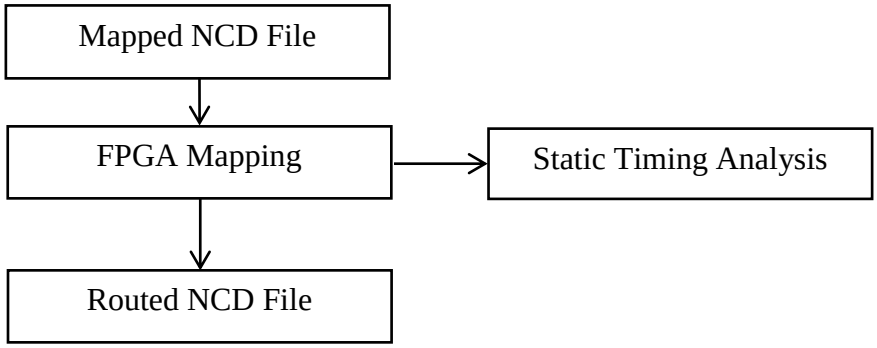


Fig 3.7 FPGA Place and Route

3.5.4. Device Programming

The above-mentioned routed design must be loaded and converted into a format supported by the FPGA. Hence, the routed NCD file is given to the Bit-Gen program, which generates a bitstream file that contains all the programming information for an FPGA. This can be done using a cable. Selection of cable depends on the design.[16]

3.5.5. Design verification

Verification can be done at different stages of the process steps.

Behavioral Simulation (RTL Simulation): This is first of all simulation steps those are encountered throughout the hierarchy of the design flow. This simulation is performed before the synthesis process to verify the register-transfer-level (RTL) (behavioral) code and to confirm that the design is functioning as intended. Behavioral simulation can be performed on either VHDL or Verilog designs. In this process, signals and variables are observed, procedures and functions are traced and breakpoints are set. This is a very fast simulation and so allows the designer to change the HDL code if the required functionality is not met within a short time period. Since the design is not yet synthesized to gate level, timing and resource usage properties are still unknown.

Functional simulation (Post Synthesis Simulation): Functional simulation gives information about the logic operation of the circuit. The designer can verify the functionality of the design using this process after the Translate process. If the functionality is not as expected, then the designer has to make changes in the code and again follow the design flow steps.

Static Timing simulation: This can be done after MAP or PAR processes Post MAP timing report lists signal path delays of the design derived from the design logic. Post Place and Route timing report incorporates timing delay information to provide a comprehensive timing summary of the design.[16]

3.6 Types of FPGA

Field Programmable Gate Arrays are classified into three types based on applications:

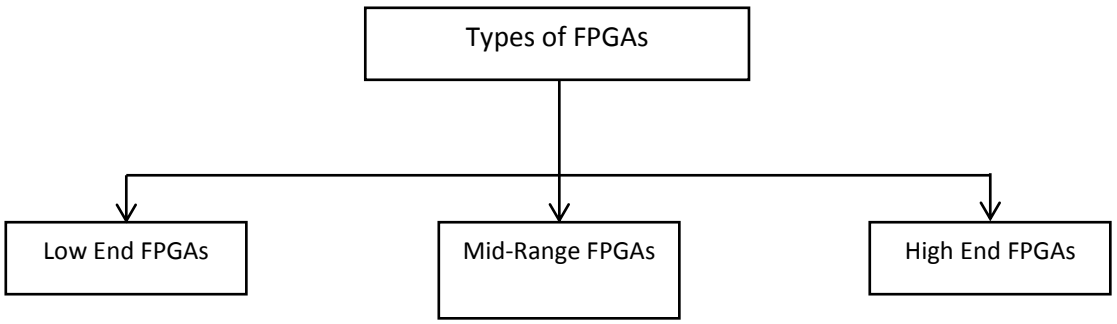


Fig 3.8 Types of FPGAs

Low-End FPGAs: designed for low power consumption, low logic density and low complexity per chip. for example (Cyclone family from Altera).

Mid-Range FPGAs: the optimum solution between the low-end and high-end FPGAs and these are developed as a balance between the performance and the cost. for example (Arria from Altera).

High End FPGAs: developed for logic density and high performance. for examples (Stratix family from Altera).[16]

3.7 FPGA Advantages and Disadvantages

FPGAs are customized by electrical modification of a packaged part. By eliminating the customization during manufacturing, FPGAs eliminate each design's custom mask-making, test pattern generation, wafer fabrication, packaging and testing. The electrical modification takes milliseconds or minutes, depending on the programming technology and size of the part and this some advantages and disadvantages of FPGA[17]:

3.7.1. Advantages

- **Low Tooling Costs**

There is no custom tooling required for an FPGA, so there are no associated tooling costs, making FPGAs cost effective for most logic designs.

- **Rapid Turnaround**

The FPGA can be programmed in minutes by the user. On an FPGA, a modification to correct a design flaw or to address a late specification change can be made quickly and cheaply. Faster design turnaround leads to faster product development and shorter time-to-market for new FPGA products.

- **Low Risk**

The benefits of low initial Non-Recurring Engineering (NRE) charges and rapid turnaround means that a design iteration due to an error incurs neither a large expense nor a long delay. Low costs encourage early system integration and prototyping. The low cost of error also encourages

more aggressive logic design, which may yield better performance and more cost-effective designs.

- **Effective Design Verification**

FPGAs avoid the problems of NRE. Instead of simulating large amounts of time, FPGA users may choose to use in-circuit verification. Designers can implement the design and use a functioning part as a prototype. The prototype operates at full speed and with excellent timing accuracy. A prototype can be inserted into the system to verify functionality of the system as a whole, eliminating a class of system errors early.

- **Low Testing Costs**

All ICs must be tested to verify proper manufacturing and packaging. This test is different for each design. The test program for FPGAs is the same for all designs and tests the FPGA for all users. Because there is only one test program, it is reasonable to invest a considerable amount of effort in it, and it can be continually improved over the lifetime of the FPGA. The resulting test program achieves excellent test coverage, leading to high-quality.

- **Standard-Product Advantages**

New, denser integrated circuit technologies drive microelectronic advances. New processing technologies have finer geometries with smaller transistors and wires. The speed and cost of a chip are related to these dimensions, so a smaller chip is both cheaper and faster.[17]

3.7.2. Disadvantages

- **Chip Size and Cost**

The area penalty for field-programmability is significant. Current FPGAs are about ten times larger for the same gate capacity as the equivalent Mask Programmable Gate Array (MPGA), and are correspondingly more expensive on a per-chip basis.

- **Speed of Circuitry**

The connection paths in an FPGA are slowed by the programming circuitry. Programmable interconnection points along a wiring path add resistance to the path. All programming points in

the interconnect add capacitance to the internal paths. Finally, since more area is required for the same amount of logic, interconnect lines between logic are longer. Longer lines have greater resistance and capacitance, further slowing the resulting circuitry. Current FPGAs are two to three times slower than MPGAs and it is unlikely that they will ever equal MPGA performance on the same manufacturing process.

- **Design Methodology**

FPGAs have been criticized because they are too easy to use, thereby encouraging a "try-it-and-see-what-happens" methodology for logic design. If these "sloppy" design practices result in poor-quality designs, the resulting products will be inferior.[17]

3.8 Conclusion

In this chapter, we presented the FPGA circuit and its basic architecture and how it consists of thousands of fundamental elements CLBs. Also, we explained the FPGA architectural design flow with a diagram.

The FPGA circuit has many types that we already explained one by one (Low-End FPGAs, Mid-Range FPGAs, and High-End FPGAs). As we discussed the most important advantages and disadvantages of the FPGA circuit.

Chapter IV

Implementation and Simulation of FFT in MDC Architecture

4.1 Introduction

In In this chapter, we will present the detailed architecture of the FFT algorithm using the MDC mixed-radix 2^2-radix 2^3, simulation results, and their implemented results on FPGA. the proposed design allows calculating DFTs of two independents data sequences.

First, we give the design flow that we followed to carry out different design tasks. Then, we present detailed architectures of blocs used to build MDC mixed radix 2^2-radix 2^3 DFT calculation processor. After that, we introduce global schemas and simulation results of FFT processors for values of N ranging from 8 to 64. Finally, we present the implementation results and hardware co-simulation verification of the architectures.

4.2 Organizational Chart

Fig (4.1) illustrate the flow graph of the design process.

Input generating: The two input sequences are generated using Matlab. Each input is a complex vector where its real and imaginary part are both random sequences.

Designing the architecture: different architectures are designed using Xilinx System Generator (XSG) tool. This tool is very powerful in both designing and verification of applications based on FPGA.

DFT calculation and comparison: the DFTs of the two inputs sequences are calculated using Matlab and using the designed architectures. Matlab uses floating-point representation, where the designed architectures use fixed-point representation. Because of this difference in representation, results of the designed architectures contain a quantization error.

If the difference between the two calculation ways is small, we go to the implementation step otherwise we go back to the design step.

Implementation step: the implementation step allows us to calculate the FPGA resources utilized by our designs and the maximum achieved frequency. In addition, it allows implementing the design on the FPGA circuit.

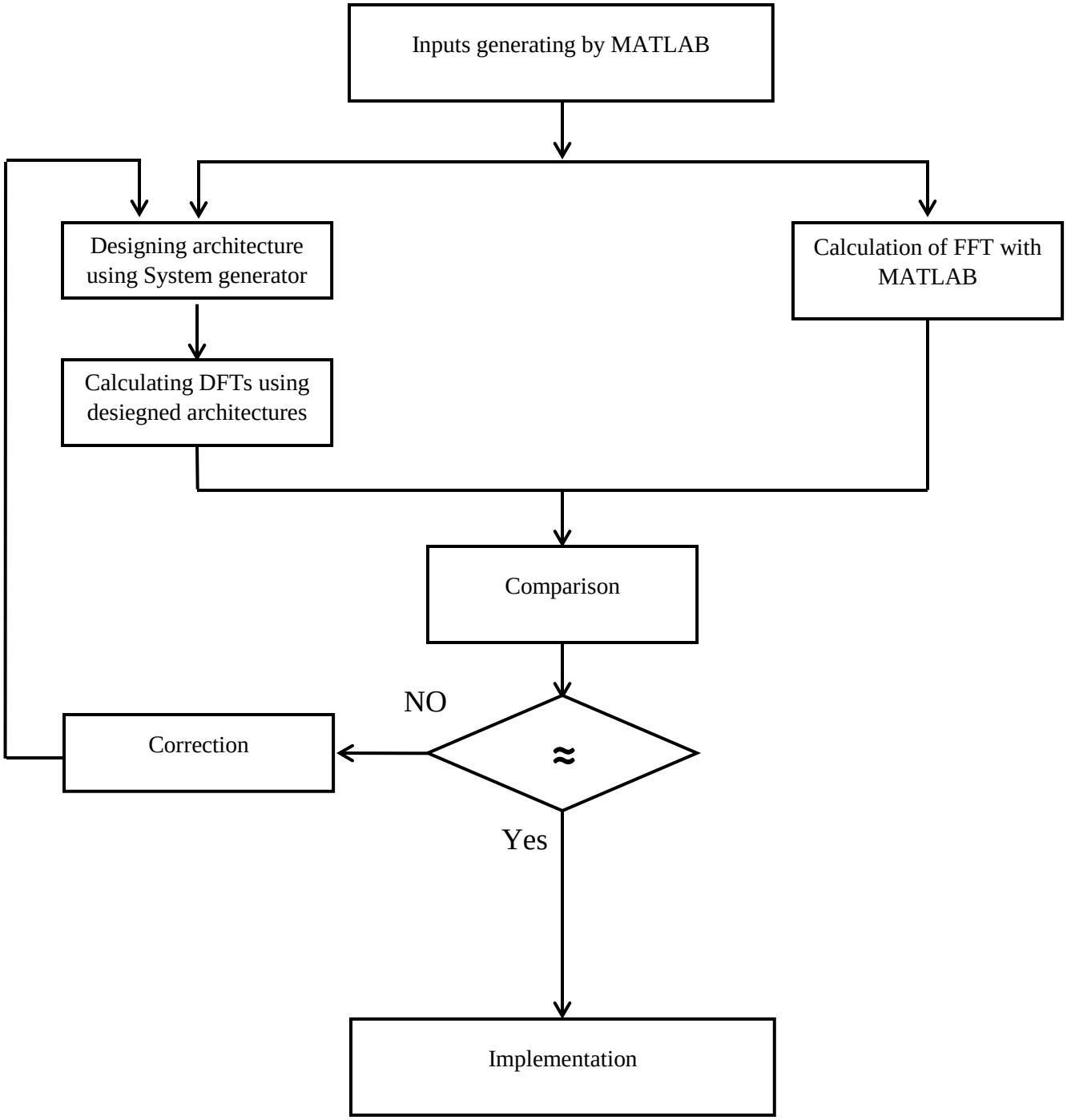


Fig.4.1 Design flow graph

4.3 Constituent blocks of MDC Radix-2^2 and Radix-2^3 architecture

4.3.1 Bloc diagram of architecture

Fig (4.2) represents the general structure of MDC Radix-2^2 and Radix-2^3 (MR-2^2-2^3) architecture. This architecture is divided into three main parts, which are: input scheduling unit, *m* modules and the control unit.

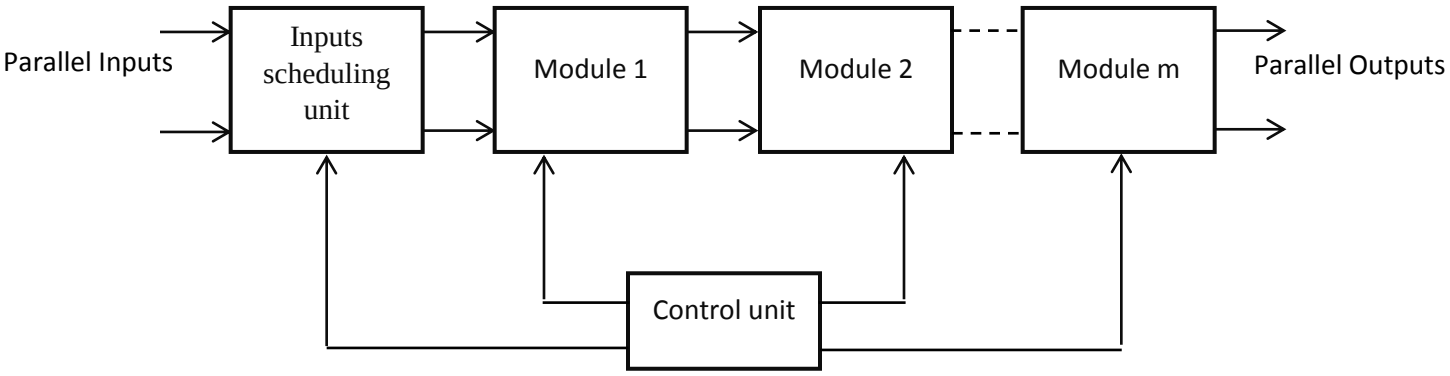


Fig.4.2 General Structure of the proposed MR22 - R23 MDC architecture

4.3.2 Clock frequency increase technique

The clock frequency is a very important parameter in the performance evaluation of FPGA based applications and systems. Indeed, the increase in the clock frequency makes it possible to increase the processing speed and data throughput of the processor without a significant increase in resources usage.

The operating frequency of the FFT processor remains limited if an appropriate method is not adopted for its improvement. The method of improving the operating frequency of an architecture implemented on an FPGA circuit consists of inserting an appropriate number of registers Flip-Flop called pipeline registers which makes it possible to reduce the length of the critical paths.

Literature shows that the number of delays to reach the maximum frequency is equal to one for the addition/subtraction and the multiplexers, two for the RAM and ROM blocks, and four for the multipliers [10].

4.3.3 Inputs scheduling unit

The proposed FFT system determines the DFT responses for two parallel data streams; *A* and *B*. The input memory scheduling system arranges the input sequences of data streams *A* and *B* as

shown in Fig. (4.3) Data stream $A = x\ (0...N - 1)$ is separated into two sub-streams, obtaining upper and lower sub-streams $A_1 = x\ (0... N/2 - 1)$ and $A_2 = x\ (N/2... N - 1)$ where sub-stream length is $N/2$. Similarly, data stream B is separated into upper sub-stream B_1 and lower sub-stream $B2$. The input data stream sequences are reshuffled to allow sub-streams $A1$ and $A2$ to be processed during the initial time index with Module 1 through m, followed by processing sub steams $B1$ and $B2$. Modules 1 through m can consist of Radix-2^2 or Radix-2^3 structures. Pipeline register latencies associated for operation functions have been presented within block diagrams. Delay elements are operated to maintain data synchronization with upper and lower sub-streams. Operations are applied to the upper Uin and lower Lin input data sub-streams to achieve the upper Uout and lower Lout output data sub-streams [10].

Initial Input order

Sequence A	0	1	…	N/2-1	N/2	N/2+1	…	N-1
Sequence B	0	1	…	N/2-1	N/2	N/2+1	…	N-1

Input order after scheduling

0	1	…	N/2-1	0	1	…	N/2-1
N/2	N/2+1	…	N-1	N/2	N/2+1	…	N-1

Sequence A Sequence B

Time Index

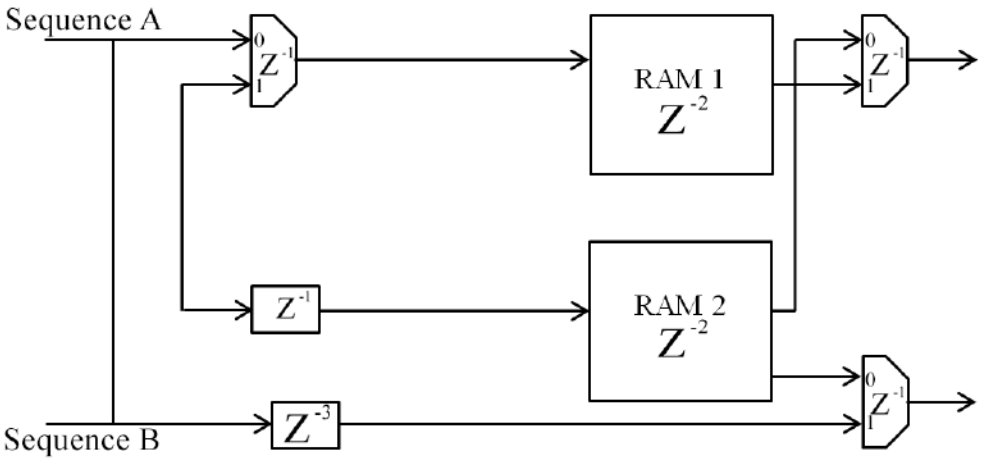


Fig.4.3 Schema of inputs scheduling unit

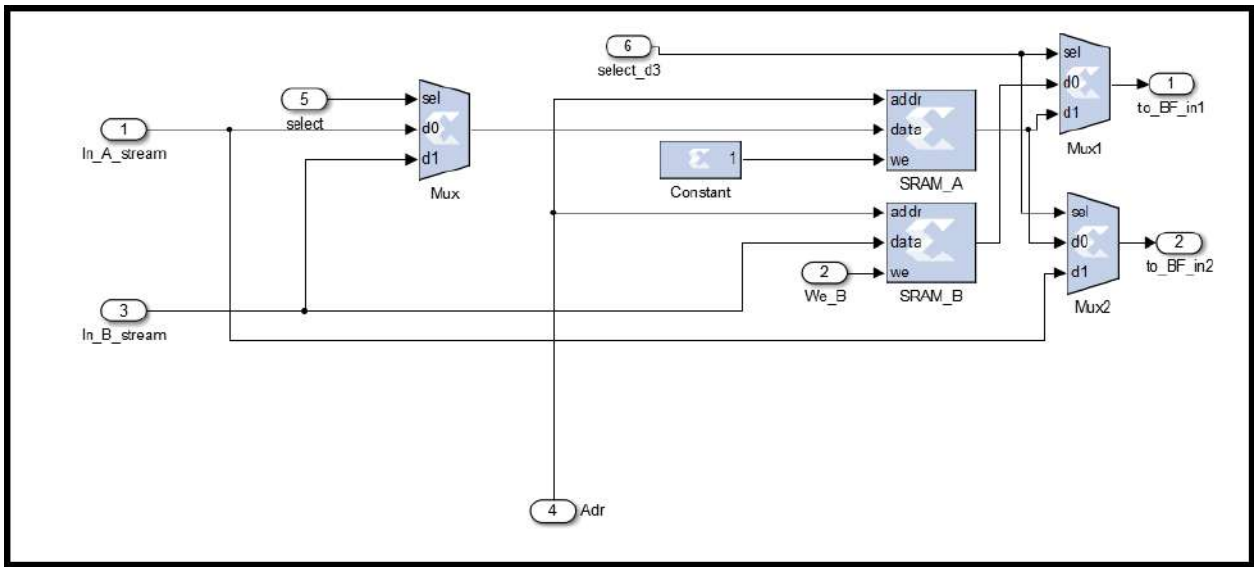


Fig.4.4 Design of the inputs scheduling unit using XSG

4.3.4 Calculation modules

In this Architecture of MR2^2 - R2^3, we will use m calculation modules. Where each module consists of either a Radix-2^2 or Radix-2^3 butterfly structures. For example to calculate DFT for *N=16* we need to use two modules of Radix 2^2 and for *N=8* we need to use only one module of Radix 2^3. Also we can use mixed Radix 2^2 and Radix 2^3 modules for *N=32*.

Let m_2 denote the number of Radix-2^2 modules, m_3 the number of Radix-2^3 modules and n_s the number of stages in the Signal Flow Graph (SFG) as we mentioned in chapter 2, $(n_s = log_2 (N))$, then *ns*, m_2, and m_3 are related as follows:

$$\boldsymbol{n_s = 2m_s + 3m_s} \qquad \textbf{IV.1}$$

4.3.4.1 Structure of Radix 2^2 module

As shown in Fig (4.5), the Radix-2^2 MDC module includes two addition/subtraction functions, one trivial multiplier, two standard multipliers, two Read Only Memories (*ROM*), two shuffling units, two multiplexers and three delay elements. The trivial multiplier performs real-imaginary swap with sign inversion operations. Fig (4.6) shows the Radix 2^2 module designed using System Generator, the detailed design of addition/subtraction unit and shuffling unit are illustrated in fig (4.7) and fig (4.8) respectively.

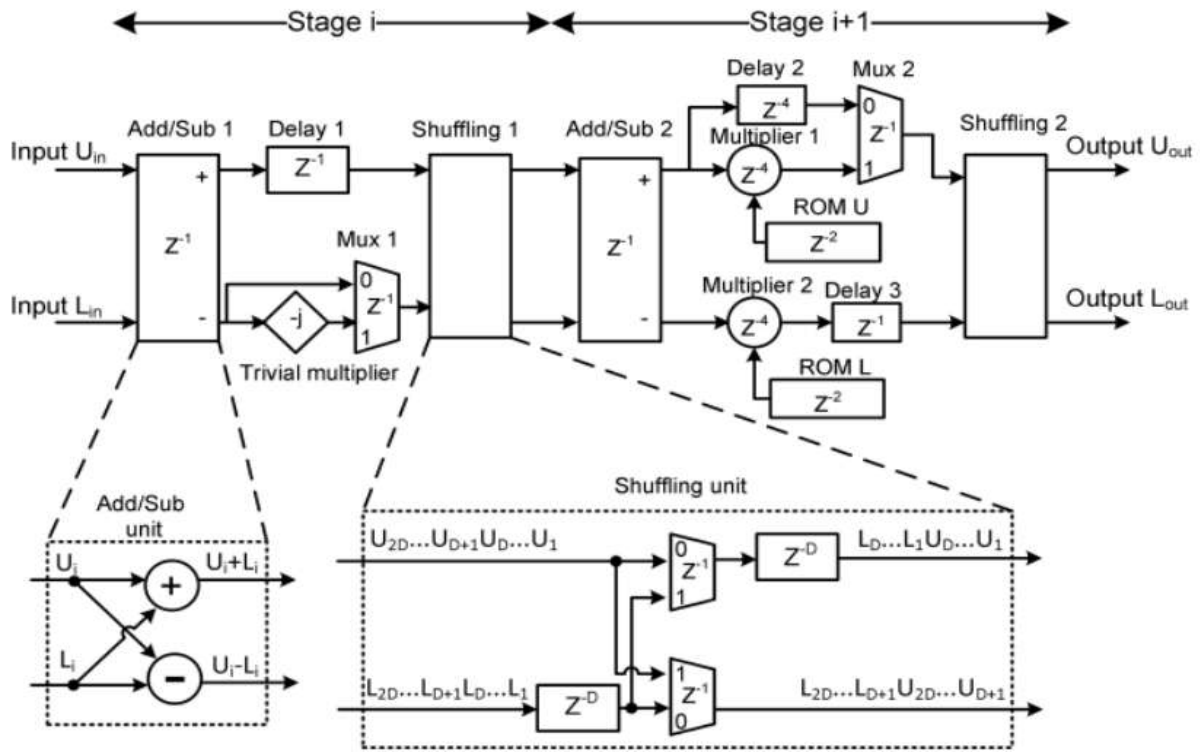


Fig.4.5 Schema of module Radix-2^2 for the architecture MR2^2 -2^3MDC with two sequences

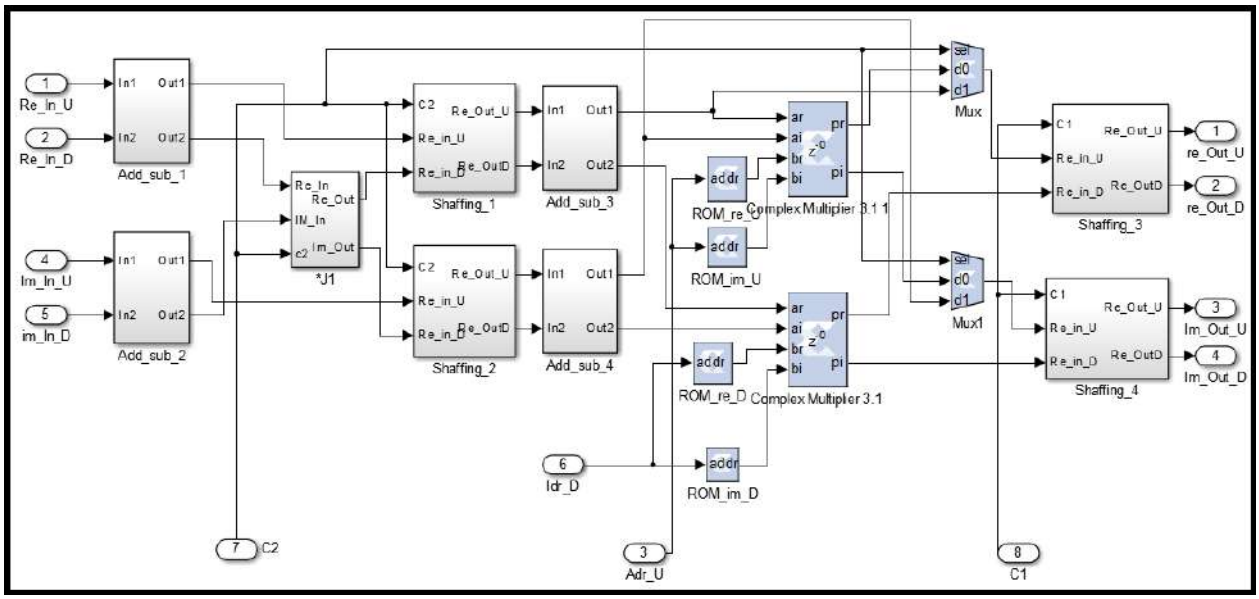


Fig.4.6 Design of Radix 2^2 2 module using XSG

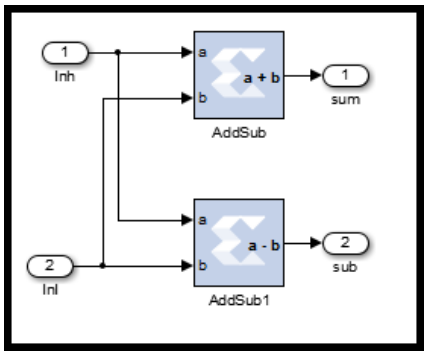


Fig.4.7 Structure of addition/subtraction unit using XSG

Shuffling units are inserted between each consecutive stages, i.e. a shuffling unit is inserted between stage *i* and stage *i+1*, they are built using *2D* delay elements, where *D = N/2(i+1)* and requires two multiplexers.

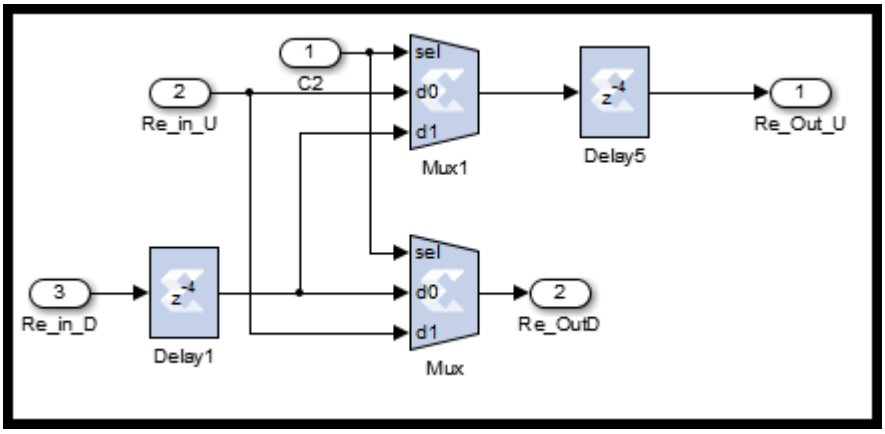


Fig.4.8 Architecture of shuffling unit using XSG

The delay elements are used for synchronizing the previous stage data outputs, i.e. stage *i*, where multiplexers allow routing data to the corresponding input of the Add/Sub unit of the stage *i*+1. The multiplexer control signal C_{n_s-i-1} is directly obtained from a binary counter. It takes the value zero for *D* clock cycles followed by the value one for *D* others clock cycles, i.e. $C_{n_s-i-1} = 00..0\ 11...1$ For *D* clock cycles, the upper data outputs of stage *i* U_1 to U_D are routed to the upper input of stage *i+1*, while simultaneously data outputs U_{D+1} to U_{2D} are routed to the lower input of stage *i+1*. The process is similarly repeated with the lower data outputs of stage *i* during the next *D* clock cycles, i.e. lower data L_1 to L_D are routed to the next stage upper input, while data L_{D+1} to L_{2D} are routed to the next stage lower input. Twiddle factors $W_{N_i}^{2n}$ with

n=0..N_i/4-1 are stored in the upper path's ROM (*ROM U*) of size $N_i/4$ while twiddle factors $W_{N_i}^{n}$ and $W_{N_i}^{3n}$ are stored in the lower path's ROM (*ROM L*) of size *Ni/2 ROM U* is addressed using bits C_0 to C_{n_s-i-2} of a binary counter while *ROM L* is addressed using bits C_0 to C_{n_s-i-1} of the same counter [10].

4.3.4.2 Structure of Radix 2^3 module

Fig (4.9) shows the Radix-2^3 module structure, it utilizes three addition/subtraction functions, one trivial multiplier, two standard multipliers, one Constant Complex Multiplication Unit (CCMU1), two ROMs, three shuffling units, one multiplexer and two delay elements. The CCMU1 operation applies rotation with $-j$, W_8^1 and W_8^3 , where the associated structure is presented in Fig (4.9). The CCMU1 contains one complex scaling operator, two trivial multipliers, and three multiplexers in addition to delay elements. The same complex scaling operator used to rotate with W_8^1, is cascaded with a trivial multiplier $-j$ to accomplish rotation with W_8^3. Twiddle factors $W_{N_i}^{0}$, $W_{N_i}^{4n}$ $W_{N_i}^{2n}$ and $W_{N_i}^{6n}$ with *n=0...N_i /8-1* are stored in *ROM U* while twiddle factors n $W_{N_i}^{n}$, $W_{N_i}^{5n}$ $W_{N_i}^{3n}$and $W_{N_i}^{7n}$ are stored in *ROM L*. Both *ROM U* and *ROM L* are of size $N_i/2$ and are addressed using bits C_0 to C_{n_s-i-1} of a binary counter. It is important to note that all operations after the last addition unit are removed in both Radix-2^2 and Radix-2^3 functions, when they are implemented within the architecture end stage of module *m* [10].

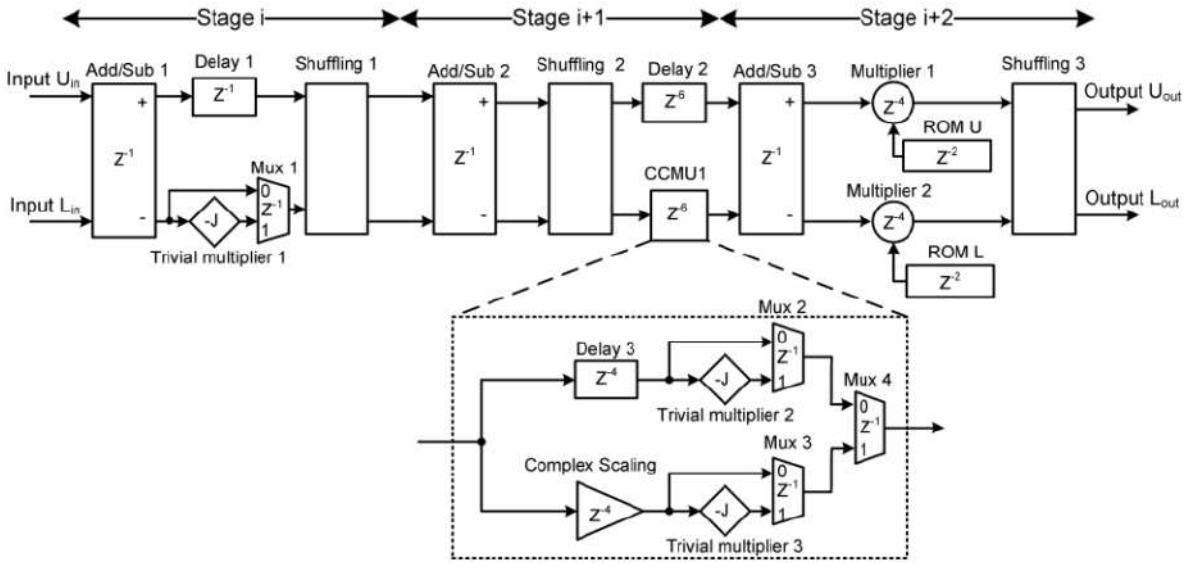


Fig.4.9 Structure of module Radix-2^3 for the architecture MR2^2 -2^3 MDC with two sequences

Based on the structure shown in fig (4.9), we have designed the corresponding architecture of fig (4.10) using XSG. Details of CCMU1 unit are illustrated in fig (4.11).

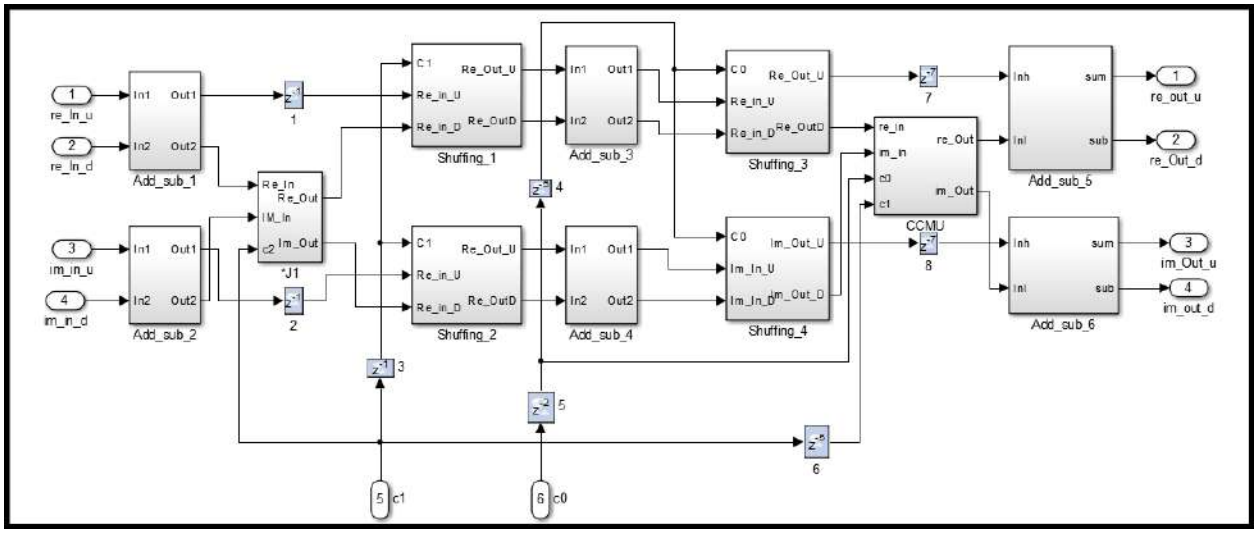


Fig.4.10 Design of Radix 2^3 module using XSG.

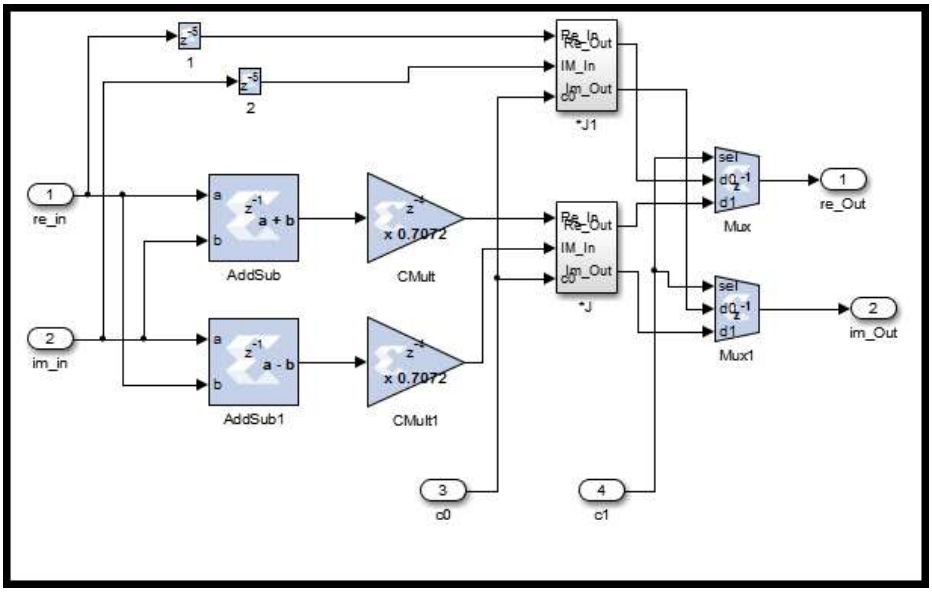


Fig.4.11 Architecture of CCMU1 unit using XSG

4.4 Simulation results

4.4.1 Architecture for *N*=8

Fig (4.12) shows the global architecture of design for *N*=8, the structure contains one Radix-2^3 module.

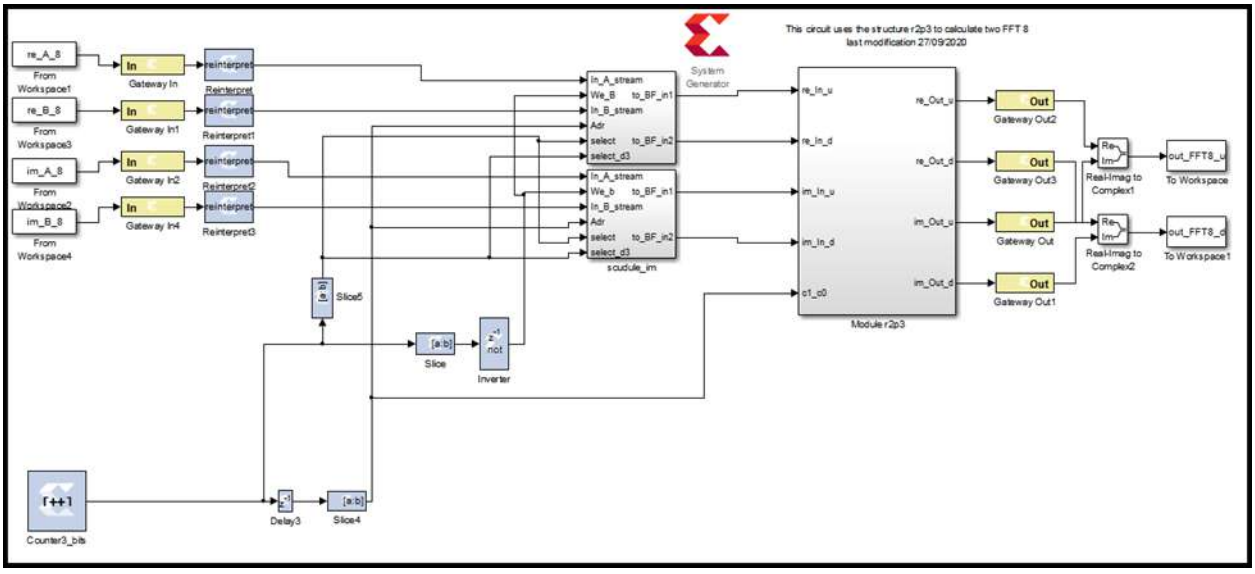


Fig.4.12 Global architecture for *N*=8 using XSG.

Simulation results are presented in table 4.1, where the first column is for Matlab results, the second column is for XSG results and third column is for the difference between column one and column two (quantization error). From the table 4.1 we can see that results from XSG are very close to those from Matlab.

Table 4.1 Simulation results for *N*=8.

FFT From Our Architecture	FFT From MATLAB	Difference
-1.5542 + 0.1826i	-1.5537 + 0.1831i	0.0005 + 0.0005i
-1.4990 + 0.4678i	-1.4964 + 0.4703i	0.0002 + 0.0071i
2.2109 - 0.2754i	2.2112 - 0.2683i	0.0027 + 0.0025i
0.6396 - 0.2432i	0.6378 - 0.2384i	-0.0019 + 0.0047i
-0.4790 - 0.5610i	-0.4788 - 0.5609i	0.0002 + 0.0001i
0.8867 + 2.1958i	0.8855 + 2.1977i	0 + 0.0001i
1.6719 - 3.5488i	1.6719 - 3.5487i	-0.0012 + 0.0019i
0.4893 - 0.3311i	0.4899 - 0.3419i	0.0007 - 0.0109i

4.4.2 Architecture for *N*=16

For *N*=16, the architecture contains two Radix-2^2. The global architecture is shown in Fig (4.13).

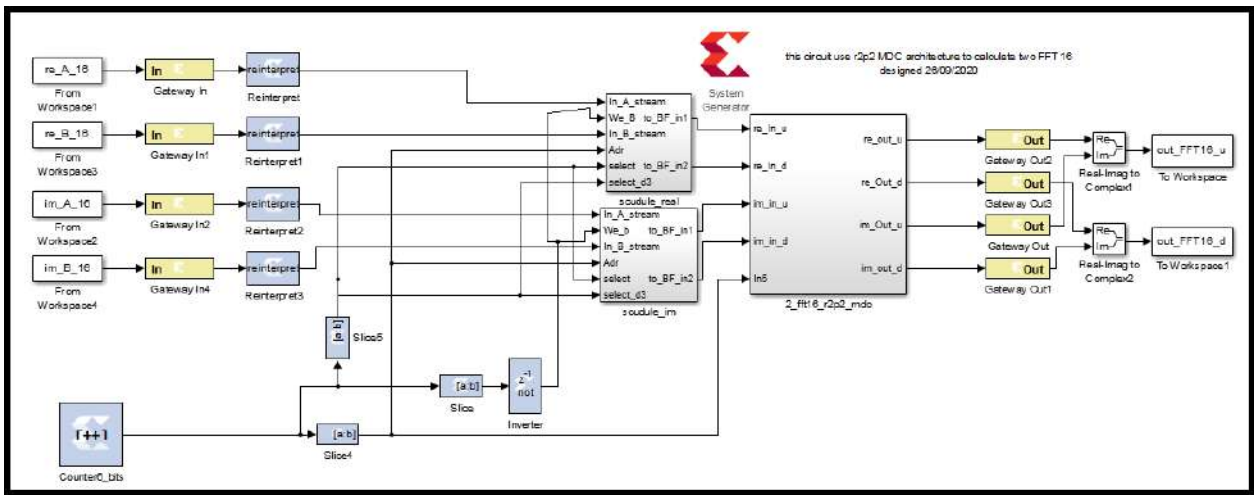


Fig.4.13 Global architecture for N=16 using XSG.

Table 4.2 present simulation results of our design for *N*=16. To avoid the presentation of long list of values, we choose only the first eight results.

Table 4.2 First eight simulation results for *N*=16.

FFT From Our Architecture	FFT From MATLAB	Difference
2.7471 - 3.2422i	2.7478 - 3.2407i	0.0007 + 0.0015i
0.3223 - 0.3691i	0.3283 - 0.3621i	0.0004 + 0.0018i
0.3291 - 1.1953i	0.3303 - 1.1951i	0.0012 + 0.0002i
-1.1611 + 4.8086i	-1.1635 + 4.8079i	0.0004 + 0.0028i
-0.0742 + 1.0195i	-0.0739 + 1.0214i	0.0060 + 0.0071i
-2.0693 - 2.3828i	-2.0676 - 2.3678i	0.0018 + 0.0150i
-1.1338 - 3.2061i	-1.1334 - 3.2032i	-0.0023 - 0.0007i
-1.5723 - 0.2305i	-1.5733 - 0.2277i	-0.0011 + 0.0028i

4.4.3 Architecture for *N*=32

To design the DFT calculation circuit for *N*=32, we can start by a Radix-2^2 module followed by a Radix-2^3 module. Another alternative, consist of using Radix-2^3 followed by Radix-2^2 module. Fig (4.14) present the global architecture for *N*=32.

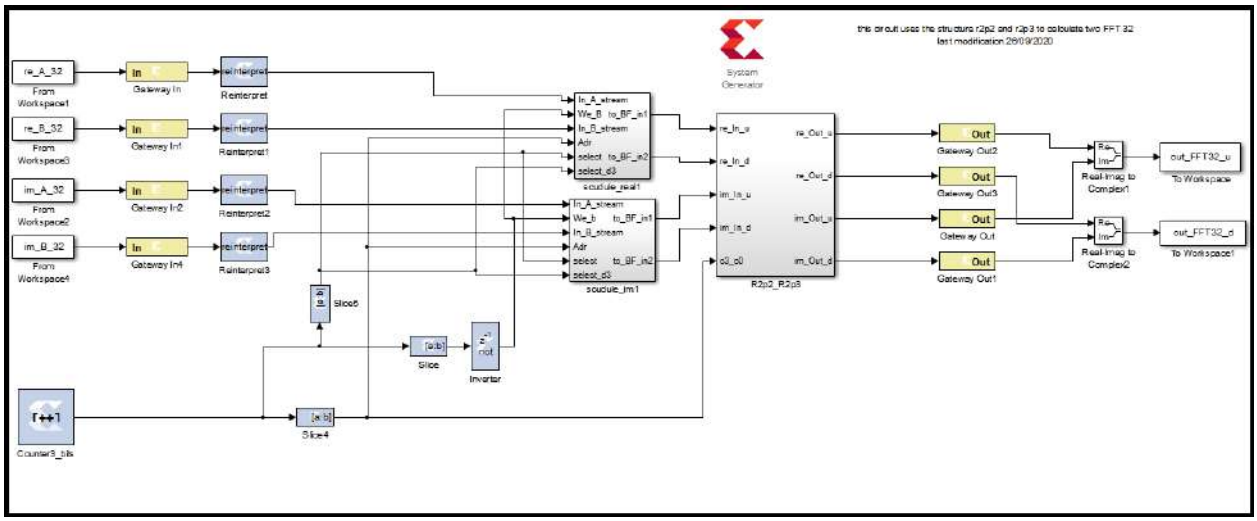


Fig.4.14 Global architecture for *N*=32 using XSG.

The eight first values of DFT outputs for *N=32* are given in table 4.3.

Table 4.3 First eight simulation results for *N*=32

FFT From Our Architecture	FFT From MATLAB	Difference
-1.1680 + 1.0801i	-1.1637 + 1.0828i	0.0043 + 0.0027i
-1.7695 - 1.8086i	-1.7544 - 1.7956i	0.0007 - 0.0006i
-2.2344 - 3.7148i	-2.2286 - 3.7108i	0.0065 + 0.0079i
3.2461 + 0.7676i	3.2446 + 0.7715i	-0.0015 + 0.0107i
3.4355 - 9.3281i	3.4420 - 9.3202i	0.0058 + 0.0041i
1.9375 - 1.0430i	1.9468 - 1.0329i	0.0004 - 0.0001i
1.7559 - 5.2637i	1.7682 - 5.2554i	0.0123 + 0.0083i
1.6914 + 3.8828i	1.6966 + 3.9040i	-0.0028 + 0.0315i

4.4.4 Architecture for *N*=64

For *N*=64, we can design the architecture using three Radix-2^2 modules or using two Radix-2^3 modules. Fig (4.15) illustrate the global architecture of our design for *N*=64 using three Radix-2^2 modules.

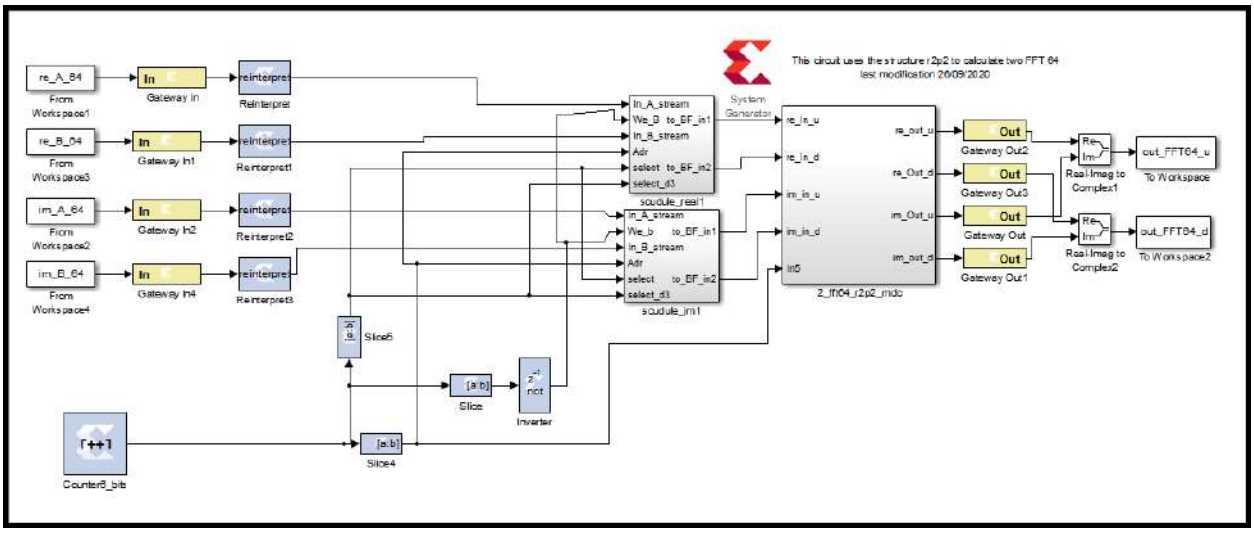


Fig.4.15 Global architecture for *N*=64 using XSG.

Table 4.4 First eight simulation results for *N*=64

FFT From Our Architecture	FFT From MATLAB	Difference
-1.1328 - 2.1719i	-1.1233 - 2.1602i	0.0095 + 0.0117i
-2.1445 - 2.7695i	-2.1170 - 2.7381i	0.0059 + 0.0466i
-6.4844 - 5.1367i	-6.4781 - 5.1269i	0.0072 + 0.0082i
3.3711 - 1.5195i	3.3742 - 1.5079i	0.0002 + 0.0237i
-1.0195 + 5.3008i	-0.9873 + 5.3442i	0.0323 + 0.0434i
-8.9453 - 1.2266i	-8.9213 - 1.1952i	-0.0029 + 0.0147i
-8.8594 + 4.8711i	-8.8386 + 4.8883i	-0.0078 + 0.0111i
1.5273 + 1.5977i	1.5451 + 1.6373i	-0.0033 + 0.0168i

4.5 Implementation results

After obtaining the correct result from the simulation step, we generate the HDL netlist of all the developed architectures fig (4.16). The FPGA target board is the Virtex-7 XC7VX485T_2FFG1761.



Fig.4.16 Generating HDL Netlist by System Generator.

After generating HDL Netlist, we start the implementation process using the ISE tool as shown in fig (4.17). In order to estimate the maximum frequency of our designs, we choose the period duration before launching the implementation process. If there are no errors after implementation finished, we try to decrease the period until obtaining the maximum achievable frequency.

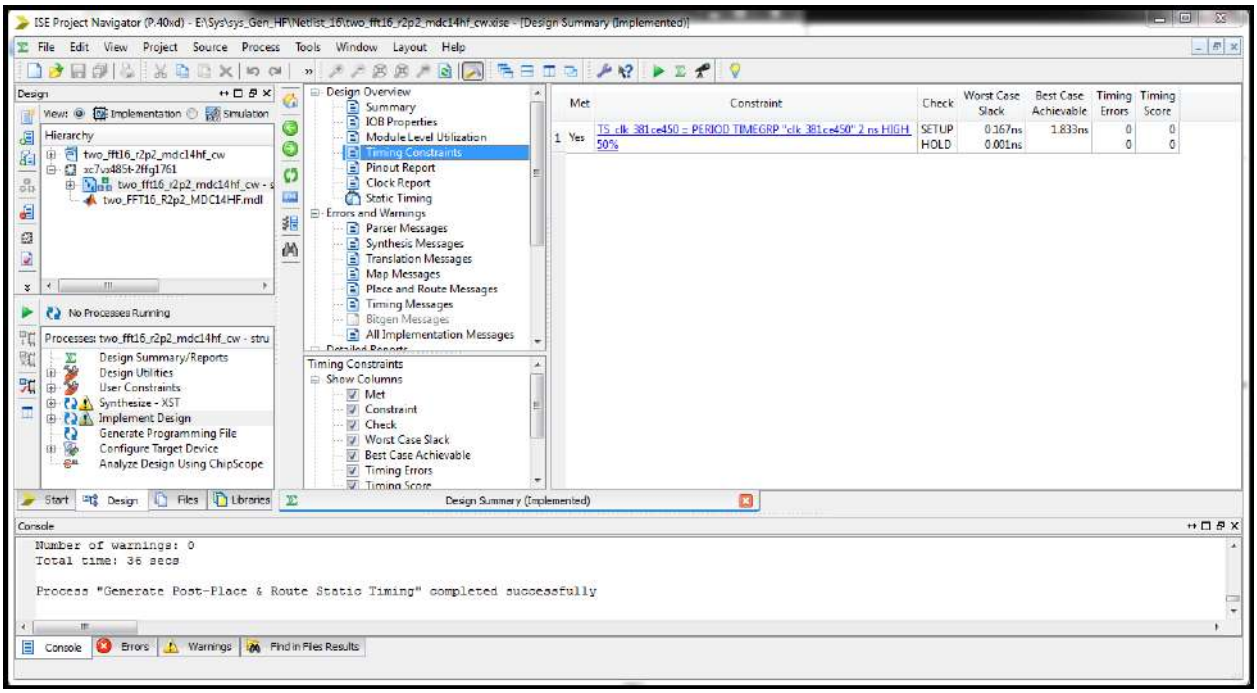


Fig.4.17 Window of ISE for implementation step.

Table (4.5) summarizes the FPGA resources utilized by different low frequency designs for various values of N. Low frequency designs refer to designs that do not use any pipeline registers to improve frequency. We can see that, in general, the number of resources increase with the increasing value of N. In addition, we can see that the designs use only a small portion of the available resources.

Table 4.5 Virtex-7 XC7VX485T_2FFG1761 resource usage of the low frequency architectures.

N	Structures	Resources			
		IOB	Number of slice registers	Number of slice LUTs	Number of slice DSP48E1s
8	2^3	113	124	638	0
16	$2^2_2^2$	113	182	589	8
32	$2^2_2^3$	113	234	967	8
	$2^3_2^2$	113	235	943	8
64	$2^2_2^2_2^2$	113	289	951	16
	$2^3_2^3$	113	291	1279	8

FPGA resources consumed by high frequency architectures are given in table 4.6. With the exception of the slice registers, it can be noted that high frequency architectures do not consume much more resources in comparison with low frequency architectures.

Table 4.6 Virtex-7 XC7VX485T_2FFG1761 resource usage of the high frequency architectures.

N	Structures	Resources			
		IOB	Number of slice registers	Number of slice LUTs	Number of slice DSP48E1s
8	2^3	113	1018	887	0
16	$2^2_2^2$	113	1104	847	8
32	$2^2_2^3$	113	1606	1291	8
	$2^3_2^2$	113	1521	1225	8
64	$2^2_2^2_2^2$	113	1629	1211	16
	$2^3_2^3$	113	2057	1679	8

Maximum clock frequency of the high frequency architectures in comparison with low frequency architectures are depicted in table 4.7. Results show clearly the important gain in frequency and throughput that we can achieve by using high frequency architecture. This gain becomes more and more significant for the large values of *N*. The most important thing is that the great improvement in working frequency is not penalized by an excessive increase in resources.

Table 4.7 Maximum frequency of high frequency architectures in comparison with low-frequency architectures.

N	Structures	Low-frequency (MHZ)	High-frequency (MHZ)
8	2^3	226.296	507.099
16	$2^2_2^2$	88.905	545.554
32	$2^2_2^3$	84.246	503.271
	$2^3_2^2$	58.882	502.765
64	$2^2_2^2_2^2$	54.121	503.778
	$2^3_2^3$	54.080	505.051

4.6 Verification by hardware-software Co-Simulation

Xilinx System Generator for DSP allows designers to perform a powerful simulation mode called hardware co-simulation. During the co-simulation process, input data consists of a Simulink signal source. Simulink runs on the PC, sending input data over the Ethernet interface to the model running in the FPGA. Processed results output from the model are sent back for display in Simulink.

We compile our design for hardware co-simulation. This automatically creates a bitstream and associates it to a System Generator co-simulation block. The picture of fig (4.18) shows the System Generator co-simulation block inserted in the design.

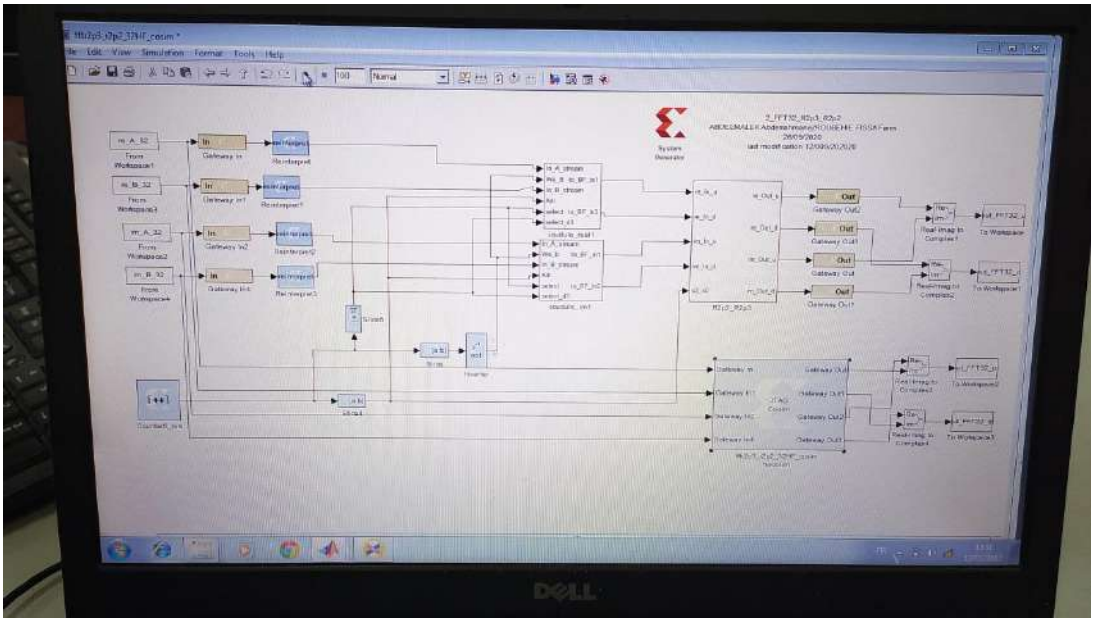


Fig.4.18 Screenshot of System Generator co-simulation block inserted in the design.

We connect the FPGA card with a JTAG cable and we start our verification. Inputs are sent to the compiled hardware model in the FPGA and results sent back to Simulink. This allows us to compare results obtained by FPGA with those obtained by system generator designs.

Fig (4.19) displays a picture of the running hardware co-simulation process using VC707 FPGA board.

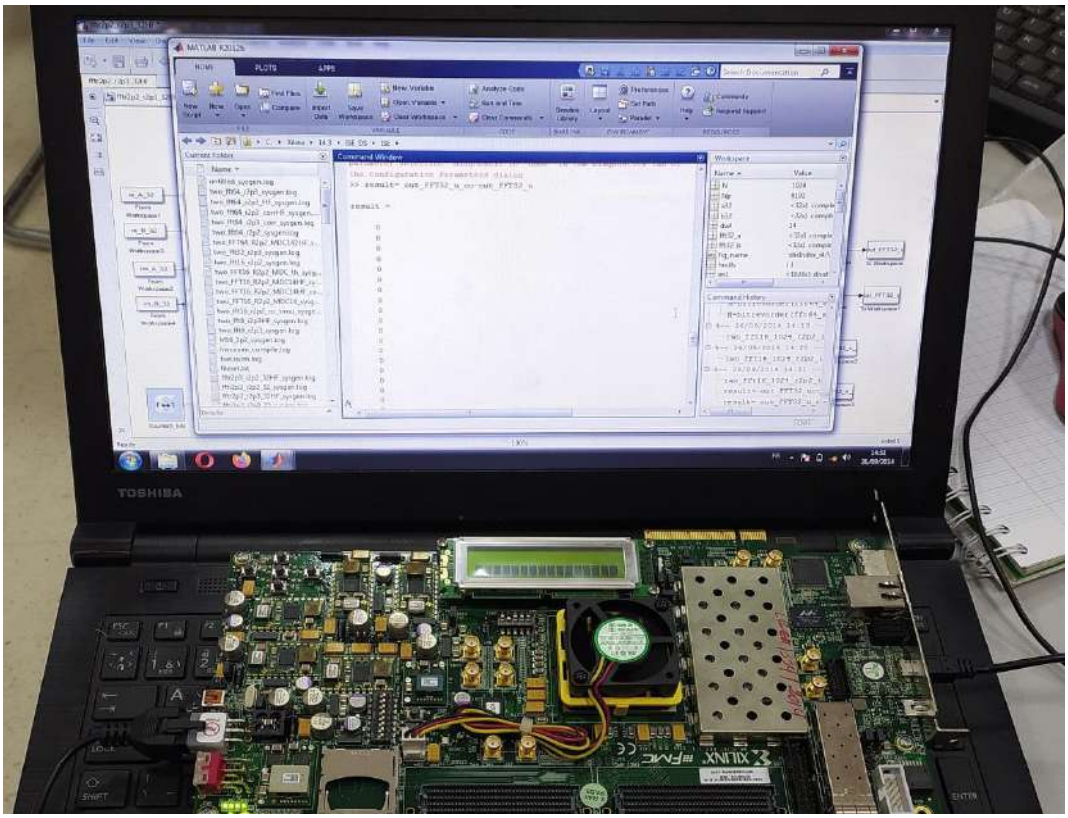


Fig.4.19 Hardware co-simulation process running on VC707 FPGA board

After we calculate the different between the outputs of FPGA cart and the outputs of architecture from XSG all the results was null, this means that the results are correct.

4.7 Conclusion

In this chapter, we presented architectures, simulation results and implementation results of the Multipath Delay Commutator Mixed Radix 2^2-2^3 (MDC *MR*-2^2-2^3) DFT calculation circuits. Simulation results shows that output using system generator are very close to those obtained using Matlab. Implementation results shows that we can achieve a high frequency by using pipeline register without an excessive increase in resources usage. The designs are also verified using the on board hardware co-simulation technique.

General Conclusion

In this work, we dealt with an explanation of four main sections to present our project.

In the first chapter, we explained the development of the communication field, and how the appearance of a new technology, called MIMO_OFDM, allowed better exploitation of spatial diversity. The use of OFDM allows big saving of bandwidth and improve performance of communication over frequency selective channel. MIMO Systems provide spatial diversity and spatial multiplexing that can be used to improve throughput and bit error rate of communication systems. The use of MIMO-OFDM can significantly improves the reliability and throughput in wireless communication.

The second chapter talked about The Discrete Fourier Transform or the DFT. To calculate efficiently the DFT, several versions of the Fast Fourier Transform FFT algorithms were proposed in the literature. In order to implement those algorithms, designers have imagined several architectures. Pipeline architectures are more suitable for high throughput applications. Pipelined architectures are classified into four types: Single path Delay Feedback (SDF) architecture, Multipath Delay Feedback (MDF) architecture, Multipath Delay Commutator (MDC) architecture and Single path Delay Commutator (SDC). These architectures differ in resources usage, control complexity and throughput.

In the third chapter, we provided an overview about Field Programmable Gate Arrays (FPGA) circuits, we gave a definition of FPGA then we presented its architecture and components, which consist of fundamental elements called configurable logic blocks (CLBs), Input/output Blocks and Switching Matrix or Interconnection Wires. Then we talked about the most commonly used programming techniques that are responsible for such reconfigurable architecture. There are three of the well-known programming technologies used in FPGAs (SRAM, Flash, and Anti-Fuse). We also talked about the FPGA architectural design flow comprises design entry, logic synthesis, design implementation, device programming, and design verification and represent all this designed in a general structure. After that, we gave the three

types of FPGA: Low-End FPGAs, Mid-Range FPGAs, and High-End FPGAs. Finally, we gave some of the advantages and drawbacks of the FPGA circuit.

In the last chapter, we presented design simulation an FPGA implementation of high frequency FFT architecture based on mixed Radix-2^2 Radix-2^3 MDC structures. The architectures can process two independent data streams with high-throughput and efficient resource usage. The system is very simple to control with binary counter signals. In addition, the architectures are based on mixed Radix concept, which allows to realize scalable architectures to any power of two FFT size ($N = 2^{n_s}$). The designs were implemented on Virtex-7 FPGA device. Implementation results demonstrate that the architectures can achieve a high clock frequency . (Around 500 Mhz) which is a significant improvement in comparison to equivalent low-frequency designs.

As perspective of the present work, we can suggest the extension of the architecture to other values of *N* (128, 256, 512, 1024…). Another perspective is to use more parallel paths in the architectures (4 paths, 8 paths …) to improve the throughput.

Bibliography

[1] A. Baraskar, “Digital Communication system,” *researchgate*, 2009. .

[2] B. B.-M. L Akram, “Implementation sur FPGA d’une Architecture FFT Parallélé pour les Systemes de Telecommunication MIMO-OFDM,” University Dr Yahia Fares de Medea, 2017.

[3] P. Ramjee, *OFDM for Wireless Communications Systems*. Artech House, 2004.

[4] G. Wiranto, *Innovative ASEAN: Creating ASEAN Competitiveness Through Innovation, Sciences, and Technology : the 7th ASEAN Science and Technology Week, Sub Commitee Conference on Microelectronics and Information Technology*. Menteng, Jakarta : LIPI Press, 2006.

[5] M.Z. Win , A.F. Molisch, “MIMO systems with antenna selection,” *IEEE Microwave Magazine*, pp. 46–56, 2004.

[6] electronics notes, “What is MIMO Wireless Technology,” *electronics-notes*, 2012. .

[7] H. Bolcskei, “MIMO-OFDM Wireless Systems: Basics, Perspectives, And Challenges,” *IEEE Wireless Communications*, pp. 31–37, 2006.

[8] C. B. M Heideman, D Johnson, “Gauss and the History of the Fast Fourier Transform,” *IEEE Assp Magazine*, vol. 369, no. 1, pp. 1689–1699, 1984.

[9] J. H. KR Rao, DN Kim, *Fast Fourier Transform: Algorithms and Applications*, vol. (5)2, no. 2. 2011.

[10] M. Dali, A. Guessoum, R. M. Gibson, A. Amira, and N. Ramzan, “Efficient FPGA implementation of high-throughput mixed radix multipath delay commutator FFT processor for mimo-ofdm,” *Adv. Electr. Comput. Eng.*, vol. 17, no. 1, pp. 27–38, 2017, doi: 10.4316/AECE.2017.01005.

[11] D. T.-H. L. Lin Liyun, Li Yun Lin Li, Chenghui, “Implementation of a variable length FFT processor for VDSL system,” National Chiao Tung University Institutional Repository, 2003.

[12] N Chokri, “Conception et Implémentation sur FPGA de Structures Multiplexes de la FFT en Pipeline,” Université Du Québec, 2017.

[13] xilinx, “xilinx.com.” .

[14] S. Akthar, “FPGA Architecture,” 2014. .

[15] H. Mehrez , Z. Marrakchi , U. Farooq, *Tree-Based Heterogeneous FPGA Architectures*. Springer Science & Business Media, 2012.

[16] S. Devi, “FPGA Design Flow,” *academia*. .

[17] S. M. Trimberger, *Field-Programmable Gate Array Technology*. Springer Science & Business Media, 2012.