Analysis and Design of FFT Processor Architecture for OFDM Applications

THESIS

Submitted in partial fulfilment
of the requirements for the degree of
DOCTOR OF PHILOSOPHY

by

Ganjikunta Ganesh Kumar
ID. No. 2013PHXF0007H

Under the Supervision of
Prof. Subhendu K Sahoo

BITS Pilani
Pilani | Dubai | Goa | Hyderabad

BIRLA INSTITUTE OF TECHNOLOGY AND SCIENCE - PILANI
2018

BIRLA INSTITUTE OF TECHNOLOGY AND SCIENCE - PILANI

CERTIFICATE

This is to certify that the thesis entitled, Analysis and Design of FFT Processor Architecture for OFDM Applications and submitted by Ganjikunta Ganesh Kumar ID No. 2013PHXF0007H for award of Ph.D. of the Institute embodies original work done by him under my supervision.

Supervisor

Prof. Subhendu K Sahoo

Associate Professor,

BITS-Pilani, Hyderabad Campus

Date:

Acknowledgements

I wish to express my sincere gratitude to my supervisor Prof. Subhendu K Sahoo for his continuous help and valuable guidance, without which this work would not have been accomplished. His unwavering support through-out the journey has been a constant source of encouragement. I am grateful to Prof. G. Sundar, Director, Birla Institute of Technology and Science - Pilani, Hyderabad Campus, for providing all the necessary facilities required to carried out my research. I also take this opportunity to thank the previous and current Heads of the Department of Electrical Engineering, Prof. Y. Yoganandam and Prof. Sanket Goel, respectively for the support and help extended to me. I would also like to thank my Doctoral Advisory Committee (DAC): Prof. Prabhakara Rao and Dr. S. K. Chatterjee, for their timely feedback and insightful comments. I heartily thank Prof. Pramod Kumar Meher, School of Computer Engineering, Nanyang Technological University, Singapore, for his support throughout the research work. To all my friends in the department, thank you for your cooperation and camaraderie - especially to Mr. Avinash S Vaidya, Mr. Sai Phaneendra P, Mr. Goutham Makkena, and Mr. Chetan Kumar V for helping me to overcome difficult phases and not letting me give up. Lastly, my warmest acknowledgments to my parents, Mr. Venkata Ramudu and Mrs. Lakshmi Devi, my sister Mrs. Bhargavi and my brother-in-law Mr. P Manohara for their unconditional love, and support throughout my Ph.D.

Abstract

The aim of this thesis is to analyze fast Fourier transform (FFT) at algorithmic, architecture, and arithmetic level and the possibilities to reduce complexity and power consumption of the twiddle factor multiplication in FFT for IEEE 802.11a and 802.15.4$-g$ OFDM systems. In algorithmic level, we have analyzed various algorithms in terms of arithmetic complexity (number of real multiplications and real additions), that effect accuracy, area, and power consumption of the architecture. At architecture level, different FFT architectures are analyzed and compared with respect to number of complex multipliers, complex adders and memory. Further, the design choice of FFT algorithm and architecture for OFDM systems is discussed. At arithmetic level, the hardware implementation of the twiddle factor multiplication is presented. Initially, the implementation of twiddle factor multiplication using general complex multiplication is discussed and then, the twiddle factor multiplication architecture based on constant multiplications are investigated.

Based on the FFT analysis, two architectures are proposed: i) fixed length (64-point) FFT for IEEE 802.11a WLAN application and ii) variable-length (16/32/64/128-point) FFT for IEEE 802.15.4 $-$ g WPAN application. For fixed-length FFT, novel pipelined single-path delay feedback (SDF) architectures are proposed for radix-2^2 and radix-2^3 algorithms using modified complex constant multipliers. These architectures

result in low hardware complexity and consumes low power compared to earlier FFT architectures.

For variable-length (16/32/64/128-point) FFT, a SDF architecture is proposed based on mixed-radix algorithm to reduce the number of complex multipliers. It employs a configurable complex constant multiplier (CCCM) structure instead of fixed constant multiplier to perform variable-length twiddle factor multiplication. A hardware-sharing mechanism is introduced to reduce the memory space requirements of the proposed FFT computation scheme. Both Field Programmable Gate Array (FPGA) and Application Specific Integrated Circuit (ASIC) targeted synthesis results of the proposed variable-length processor are presented.

Contents

List of Figures

List of Tables

Abbreviations

DFT	**D**iscrete **F**ourier **T**ransform
IDFT	**Inverse D**iscrete **F**ourier **T**ransform
FFT	**F**ast **F**ourier **T**ransform
IFFT	**Inverse F**ast **F**ourier **T**ransform
OFDM	**O**rthogonal **F**requency **D**ivision **M**ultiplexing
FPGA	**F**ield-**P**rogrammable **G**ate **A**rray
ASIC	**A**pplication-**S**pecific **I**ntegrated **C**ircuit
CORDIC	**CO**ordinate **R**otation **DI**gital **C**omputer
CFFT	**C**omplex-valued **F**ast **F**ourier **T**ransform
DIT	**D**ecimation **I**n **T**ime
DIF	**D**ecimation **I**n **F**requency
RFFT	**R**eal-valued **F**ast **F**ourier **T**ransform
SRFFT	**S**plit-**R**adix **F**ast **F**ourier **T**ransform
SDFT	**S**caled **D**iscrete **F**ourier **T**ransform
SDF	**S**ingle-path **D**elay **F**eedback
CSD	**C**anonical **S**igned **D**igit
ASIC	**A**pplication **S**pecific **I**ntegrated **C**ircuit
FPGA	**F**ield **P**rogrammable **G**ate **A**rray
3GPP	3^{rd} **G**eneration **P**artnership **P**roject
LTE	**L**ong **T**erm **E**volution
MDCT	**M**odified **D**iscrete **C**osine **T**ransform

SFFT	**S**parse **F**ast **F**ourier **T**ransform
QFT	**Q**uantum **F**ourier **T**ransform
PEs	**P**rocessing **E**lements
R2SDF	**R**adix-2 **S**ingle-path **D**elay **F**eedback
R4SDF	**R**adix-4 **S**ingle-path **D**elay **F**eedback
BF	**B**utterfly
R2^{2}SDF	**R**adix-2^2 **S**ingle-path **D**elay **F**eedback
MDF	**M**ulti-path **D**elay **F**eedback
R2MDC	**R**adix-2 **M**ultipath **D**elay **C**ommutator
R4MDC	**R**adix-4 **M**ultipath **D**elay **C**ommutator
SDC	**S**ingle-path **D**elay **C**ommutator
R4SDC	**R**adix-4 **S**ingle-path **D**elay **C**ommutator
DAC	**D**igital-to-**A**nalog **C**onverter
ADC	**A**nalog-to-**D**igital **C**onverter
R2^{i}SDF	**R**adix-2^i **S**ingle-path **D**elay **F**eedback
SRMDC	**S**plit-**R**adix **M**ulti-path **D**elay **C**ommutator
LSP	**L**east **S**ignificant **P**art
MSP	**M**ost **S**ignificant **P**art
SCM	**S**ingle **C**onstant **M**ultiplication
MCM	**M**ultiple **C**onstant **M**ultiplication
ROM	**R**ead- **O**nly **M**emory
CSD	**C**anonical **S**igned **D**igit
CCM	**C**omplex **C**onstant **M**ultiplier
CCCM	**C**onfigurable **C**omplex **C**onstant **M**ultiplier
SQNR	**S**ignal-to-**Q**uantization-**N**oise **R**atio
MR-OFDM	**M**ulti-**R**ate and multi-regional **O**rthogonal **F**requency **D**ivision **M**ultiplexing
MR-FSK	**M**ulti-**R**ate and multi-regional **F**requency **S**hift **K**eying

MR-OQPSK	**M**ulti-**R**ate and multi-regional **O**ffset **Q**uadrature **P**hase **S**hift **K**eying
DSSS	**D**irect **S**equence **S**pread **S**pectrum
SUN	**S**mart Metering **U**tility **N**etworks
AMR	**A**utomatic **M**eter **R**eading
RF	**R**adio **F**requency
LR-WPAN	**L**ow **R**ate **W**ireless **P**ersonal **A**rea **N**etwork
WPAN	**W**ireless **P**ersonal **A**rea **N**etwork
DSP	**D**igital **S**ignal **P**rocessing
WLAN	**W**ireless **L**ocal **A**rea **N**etwork
nm	Nanometer
ns	Nanosecond
μm^2	Square Micrometer

Chapter 1

Introduction

The discrete Fourier transform (DFT) is the most widely used tool in digital signal processing (DSP) systems. It has indispensable role in many applications, such as speech, audio and image processing, signal analysis, communication systems, and many others. It maps time domain sequence to a frequency domain sequence of the same length, while the inverse discrete Fourier transform (IDFT) performs the opposite. The brute-force computation of the DFT of length N requires $O(N^2)$ multiplications. Due to such high computational requirement, it was not possible to use that for real-time and online DSP applications until 1965, when Cooley and Tukey [1] developed the famous fast Fourier transform (FFT) algorithm. It could be possible to reduce the operation count of DFT from $O(N^2)$ to $O(N \log_2 N)$, for a DFT of length N. During the last 50

years, the innovations in algorithms and architectures have made remarkable progress in the efficiency of computation of the FFT.

This chapter is organized as follows: In Section 1.1, the specifications of DFT and its computational complexity are presented. The basic technique, namely the divide and conquer approach (FFT algorithm) that reduces the computational complexity of DFT is presented in Section 1.2. The motivation of the thesis is discussed in Section 1.3. This chapter further discusses the main objectives and organization of the thesis in Sections 1.4 and 1.5, respectively.

1.1 Discrete Fourier Transform:

The N-point DFT/IDFT are, respectively, calculated as

$$X(k) = \sum_{n=0}^{N-1} x(n) W_N^{nk}, \quad k = 0, 1, 2, ..., N-1, \tag{1.1}$$

and

$$x(n) = \frac{1}{N} \sum_{k=0}^{N-1} X(k) W_N^{-nk}, \quad n = 0, 1, 2, ..., N-1, \tag{1.2}$$

where n is the time index and k is the frequency index. The twiddle factor W_N^{nk} can be represented as:

$$W_N^{nk} = e^{-j2\pi nk/N} = cos\left(\frac{2\pi nk}{N}\right) - j\ sin\left(\frac{2\pi nk}{N}\right) \tag{1.3}$$

In equations (1.1) and (1.2), the data sequence $x(n)$ may be complex while the k^{th} spectral component $X(k)$ is always complex. These two equations differ only in the sign $(-)$ of the exponent of the twiddle factor W_N and the scale factor $1/N$. Therefore, the algorithms for efficient computation of DFT could be applied for the efficient computation of IDFT [2] by simple and straightforward modifications.

Significance of the DFT

To illustrate the significance of DFT let us consider a 4-point DFT of samples of a sinusoidal signal of $10Hz$ which is expressed as:

$$x(t) = \sin(2\pi \cdot 10 \cdot t) \tag{1.4}$$

For this sine wave, the fundamental period $T_0 = 0.1\ s$ as shown in Fig. 1.1. Let us take the sample rate $f_s = 40Hz$ i.e. the input is sampled at every $1/f_s = T = 0.025\ s$. Because $N = 4$, we need 4 input sample values which could be obtained as follows:

$$x(n) = x(nT) = \sin(2\pi \cdot 10 \cdot nT) = \sin\left(\frac{n\pi}{2}\right)$$

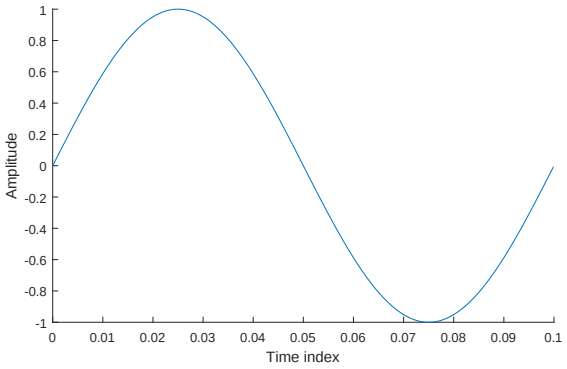

Figure 1.1: A sine wave.

$$\text{at } n = 0, x(0) = \sin(0) = 0$$

$$\text{at } n = 1, x(1) = \sin(\tfrac{\pi}{2}) = 1$$

$$\text{at } n = 2, x(2) = \sin(\pi) = 0$$

$$\text{at } n = 3, x(3) = \sin(\tfrac{3\pi}{2}) = -1$$

The finite-length sequence of $x(n)$ is shown in Fig. 1.2(a), where x-axis represents the values of n and y-axis represents the amplitude. The twiddle factors for $N = 4$ are defined as:

$$W_4^{nk} = \cos\left(\frac{2\pi nk}{4}\right) - j\sin\left(\frac{2\pi nk}{4}\right) \tag{1.5}$$

where $nk = 0$ to $N - 1$ i.e., 0 to 3. From equation (1.5), W_4^{nk} values are: $W_4^0 = 1, W_4^1 = -j, W_4^2 = -1, W_4^3 = j$.

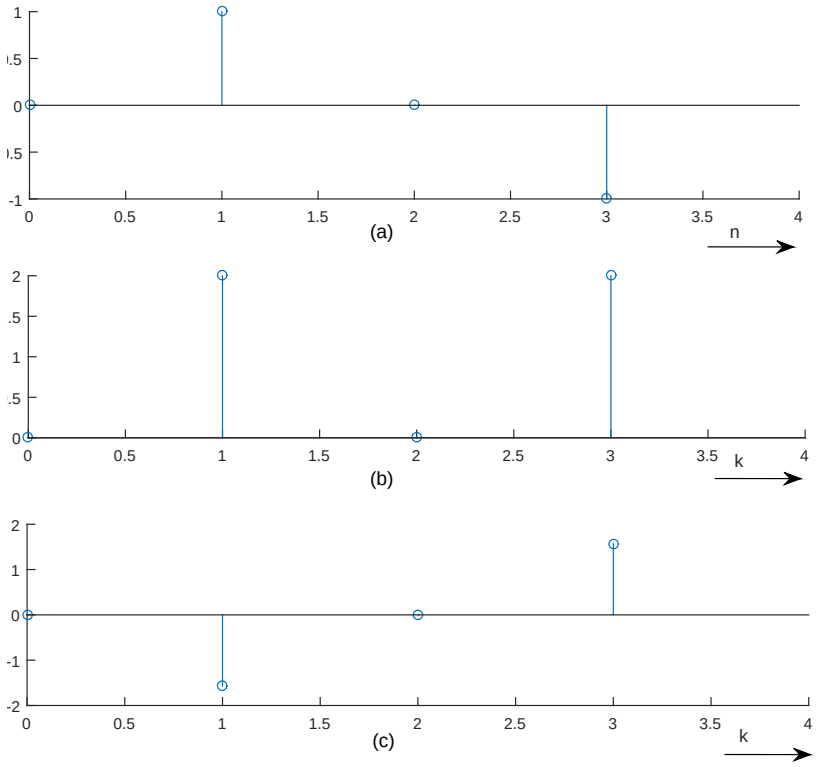

Figure 1.2: Illustration of the DFT for $N = 4$ of a sine wave. (a) Finite-length sequence $x(n)$ (b) DFT magnitude. (c) DFT phase.

The general equation for the 4-point DFT could be written as

$$\begin{aligned} X(k) &= \sum_{n=0}^{3} x(n) W_4^{kn} \\ &= x(0)W_4^{(k)(0)} + x(1)W_4^{(k)(1)} + x(2)W_4^{(k)(2)} \\ &\qquad + x(3)W_4^{(k)(3)} \qquad 0 \leq k \leq 3 \end{aligned} \tag{1.6}$$

The DFT output values are obtained for $k = 0, 1, 2, 3$ as

$$X(k) = [0 \quad -2j \quad 0 \quad 2j]$$

From the DFT output values, the sinusoidal signal can be plotted in terms of its magnitude and phase as shown in Fig. 1.2(b) and 1.2(c), respectively. The value $X(k)$ is said to provide information about the k^{th} frequency bin.

The frequency resolution can be obtained as:

$$\triangle f = \frac{1}{T_0} = \frac{1}{NT} = \frac{f_s}{N} \tag{1.7}$$

As the fundamental period of a sinusoidal signal is 0.1 s, so the frequency resolution is $10Hz$. From equation (1.7), one can observe that to increase the frequency resolution, the number of points of data N must be increased [3].

1.1.1 Computational complexity of DFT

Computation of each DFT component directly using equation (1.1) requires N complex multiplications and $(N-1)$ complex additions. Therefore, to compute all the N values of DFT requires a total number of N^2 complex multiplications and $N(N-1)$ complex additions.

The DFT of N-point complex-valued input sequence, $x(n)$ then can be expressed as

$$\begin{aligned} X(k) &= X_R(k) + jX_I(k) \\ &= \sum_{n=0}^{N-1} \left[x_R(n) + jx_I(n)\right] \left[W_{RN}^{kn} + jW_{IN}^{kn}\right] \\ &= \sum_{n=0}^{N-1} \left[(x_R(n)W_{RN}^{kn} - x_I(n)W_{IN}^{kn})\right] \\ &\quad +j\left[x_R(n)W_{IN}^{kn} + x_I(n)W_{RN}^{kn}\right] \end{aligned} \tag{1.8}$$

where $k = 0, 1, 2, ..., N-1$. Assuming that each complex multiplication in equation (1.8) is realized by 4 real multiplications and 2 real additions, while each complex addition is realized by 2 real additions, the direct computation of equation (1.8) requires $4N^2$ and $2N(N-1)$ number of real multiplications and real additions, respectively [4]. Moreover, the computation of DFT also requires a number of indexing and addressing operations to fetch the input values, intermediate results, and complex coefficients W_N^{kn} and to store the final results. For large values of N, the arithmetic complexity of DFT is very high. Therefore, different algorithms have been proposed to reduce the arithmetic complexity for fast and efficient computation of DFT.

1.2 Fast Computation of DFT: A Historical Perspective

The computational complexity of DFT is substantially reduced by using the following trigonometric symmetry and periodicity of the twiddle factor W_N^{kn}:

$$W_N^{k+\frac{N}{2}} = -W_N^k (Symmetry\ Property) \tag{1.9}$$

$$W_N^{k+N} = W_N^k (Periodicity\ Property) \tag{1.10}$$

These properties were known for a long time even before the inception of digital computation. Heideman et al. [5] have traced the first appearance of the FFT back to Gauss in the year 1805. Gauss developed an algorithm to calculate the DFT which is equivalent to one of the Cooley-Tukey algorithm. However, Gauss never published his algorithm outside his collected works. A prior work of Danielson and Lanczos [6] referred to Runge [7] for their doubling algorithm in X-ray scattering problems. Their algorithm showed how to reduce a DFT in $2N$ points to two DFTs on N points with only slightly more than N operations. The complexity of these algorithms was much less than N^2 but more than $N \log_2 N$.

The early discoveries of the FFT not noticed till the publication of Cooley and Tukey's article in 1965 [1]. This article presented an efficient algorithm based on divide

and conquer approach in order to compute the DFT. Divide and conquer approach was applied to the DFT recursively, such that a DFT of any size $N = N_1 N_2$ computed in terms of smaller DFTs of sizes N_1 and N_2. If N can be factored into $N = N_1 N_2$, the indices n and k in equation (1.1) for N-point DFT can be rewritten as:

$$\begin{aligned} n = N_2 n_1 + n_2 &\begin{cases} 0 \leq n_1 \leq N_1 - 1 \\ 0 \leq n_2 \leq N_2 - 1 \end{cases} \\ k = N_1 k_2 + k_1 &\begin{cases} 0 \leq k_1 \leq N_1 - 1 \\ 0 \leq k_2 \leq N_2 - 1 \end{cases} \end{aligned} \tag{1.11}$$

The index representation of equation (1.11) can be used in equation (1.1) to write $X(k)$ as:

$$X(k) = \underbrace{\sum_{n_2=0}^{N_2-1} \Bigg(\underbrace{\sum_{n_1=0}^{N_1-1} x(N_2 n_1 + n_2) W_{N_1}^{n_1 k_1}}_{\mathrm{N_1-point\ DFT}} \underbrace{W_N^{n_2 k_1}}_{\mathrm{twiddlefactor}} \Bigg) W_{N_2}^{n_2 k_2}}_{\mathrm{N_2-point\ DFT}} \tag{1.12}$$

where $0 \leq k_1 \leq N_1 - 1$ and $0 \leq k_2 \leq N_2 - 1$.

The calculation of *X(k)* according to equation (1.12) can be carried out in three steps: (i) compute N_1-point DFT, (ii) multiply by twiddle factors, and (iii) finally compute N_2-point DFT. The above three-step procedure can be carried out successively till $N_1 = 2$. The computational complexity of the DFT by this recursive divide and conquer approach is reduced from $O(N^2)$ to $O(N \log_2 N)$ operations [1]. This was the

major turning point for real-time DSP applications of the DFT.

1.2.1 Applications of FFT

The fast Fourier transform finds limitless applications in the general areas of signal/image processing. It plays a key role in many applications of digital signal processing, including frequency and phase estimation [8], and to perform operations such as convolutions or multiplying large integers [9, 10]. Accurate frequency and phase estimation are required in many applications such as speech recognition, speech coding, determining the object position in radar systems, biomedicine, multimedia systems etc.

FFT is used in medical imaging for image filtering, image analysis and image reconstruction [11]. In the Fourier representation of images using FFT, spectral magnitude, and phase tend to play different roles. Correlation between phase-only versions [12, 13] of the two images to be aligned is used for image matching. Some of the important applications based on the FFT-based image matching include face recognition, iris recognition, palm print recognition, finger print matching and waveform matching.

FFTs are also extensively used in multi-carrier transmission systems, specifically for applications in Orthogonal Frequency Division Multiplexing (OFDM) systems,

such as Digital Broadcasting [14, 15], Worldwide Interoperability for Microwave Access (WiMAX) [16], IEEE 802.11 standards [17], and Long Term Evolution (LTE) [18].

1.3 Motivation

The FFT algorithm reported by Cooley-Tukey in 1965, named as radix-2 algorithm was shortly followed by its extension to higher radices which include, radix-4 [19], radix-8 [20, 21], and radix-16 [22] algorithms. Higher radix algorithms involve significantly less computational complexity in terms of the number of complex multiplications, but the implementation of these algorithms is not simple. Among numerous further developments, the FFT introduced by Duhamel and Hollmann [23] demonstrated a reduction in the number of multiplications at the cost of input and output mapping. The complexity issue has been studied detail in [4], and showed that the split-radix algorithm requires low arithmetic complexity. However, the implementation of split-radix algorithm is difficult, owing to its irregular structure.

Later in 1996, He and Torkelson [24] proposed a radix-2^2 algorithm using index decomposition technique. It has exactly the same number of complex multipliers as radix-4 algorithm, but has a butterfly structure similar to that of radix-2 algorithm. This led to, radix-2^3 [25], radix-2^4 [26], modified radix-2^4 [26], radix-2^5 [27], modified radix-2^5 [27] and radix-2^i [28] FFT algorithms being proposed by various researchers using the same index decomposition technique, in order to further reduce the number

of multiplications. The main advantages of the generalized radix-2^i [28] algorithm are high throughput and low latency with less area and less power consumption. This makes radix-2^i algorithms more attractive for different applications in communication systems [28].

Twiddle factor multiplication requires both memory and complex multipliers. The implementation of complex multipliers has a large impact on the accuracy, speed, complexity, and power consumption of the design as well. A complex multiplier can be realized by different approaches such as direct implementation of the complex multiplier [29], and algorithms based on constant multiplication [30, 31]. In all these implementations, there is a trade-off between the complexity and the accuracy of the twiddle factor multiplication. In FFT designs the scenario of twiddle factor multiplication is distinct, and impacts the selection of algorithm and architecture. The main aim of this thesis is to select an appropriate FFT algorithm and architecture and to optimize twiddle factor multiplication in the FFT for IEEE 802.11a and IEEE $802.15.4-g$ standard OFDM systems.

1.4 Objectives of the Thesis

The FFT used in OFDM system is the most complex and power hungry block. As this system is used mostly in the battery driven wireless applications, low power

consumption is desired. This thesis aims at designing ASIC and/or FPGA FFTs for OFDM systems. The main objectives of this thesis are as follows:

- To choose the best FFT algorithm and an appropriate architecture with less hardware complexity suitable for IEEE 802.11a and IEEE 802.15.4 $-$ g standard OFDM systems.
- Modify designs at basic processing elements (adder, multiplier and delay buffers) to achieve enhanced performance.
- Simulate the proposed FFT architectures to verify the correctness of the functionality.
- Synthesize the FFT architectures to obtain ASIC and/or FPGA implementation and compare its performance with recent implementations in the literature.

1.5 Organization of the Thesis

The thesis is organized as follows.

Chapter 2 introduces the advancements of FFT Algorithms and provides their overview from a mathematical perspective. These algorithms include complex-valued FFTs, real-valued FFTs and special cases of FFTs.

Chapter 3 is a survey of architectural techniques for creating hardware efficient and low-power implementations of the FFT. These architectures include memory-based and pipelined architectures. The important design choices and considerations are also discussed and investigated here.

Chapter 4 discusses various possibilities to implement twiddle factor multiplication and its hardware. It includes general complex multiplication and constant multiplication.

Chapter 5 introduces the design consideration of a 64-point FFT for OFDM based IEEE 802.11a standard system. A novel area-efficient and low power 16-bit word-width 64-point radix-2^2 and radix-2^3 pipelined Single-path Delay Feedback (SDF) FFT architectures are presented based on modified complex constant multiplier. The remainder of this chapter describes the implementation details and comparisons with recent implementations in literature.

Chapter 6 presents a novel 16/32/64/128-point pipelined SDF FFT architecture based on mixed-radix algorithm for IEEE 802.15.4-g standard system. Both FPGA and ASIC targeted synthesis results are presented. The comparison of this structure with other published results is provided at the end.

Chapter 7 concludes with a summary of total contributions of this thesis. It also discuss and made suggestions for future research possibilities using the optimized twiddle factor multiplication.

Chapter 2

Advancements in FFT Algorithm

2.1 Introduction

The basic principle of divide and conquer approach leads to a variety of efficient algorithms. As these algorithms improves the performance in terms of computation time, these are known as fast algorithms or fast Fourier transform algorithms. In this chapter, we have classified the FFT algorithms as complex valued FFTs (CFFTs), real-valued FFTs (RFFTs) according to the input values and special cases of FFTs. Finally, this chapter concludes with comparison of FFT algorithms that can be helpful to choose the best algorithm for OFDM applications.

2.2 Complex-valued FFT Algorithms

In this subsection, we discuss the popular FFT algorithm followed by some algorithms that can improve the computational speed and reduce the hardware complexity.

2.2.1 Radix-2 FFT algorithms

The basic FFT algorithms are decimation-in-time (DIT) and the decimation-in-frequency (DIF) radix-2 algorithms. These algorithms are applicable to compute the DFT of integer power of 2 lengths.

i) Decimation-in-Time Radix-2 FFT Algorithm

This algorithm decomposes the time domain sequence $\{x(n)\}$ into successively smaller subsequence. Therefore, it is called as decimation-in-time algorithm [32].

The principle of radix-2 DIT FFT algorithm is illustrated in the following by considering $N = 2^M$, where $M = 1, 2, 3, \ldots$. Since N is an even integer, the N-point input data can be split into two $(N/2)$-point sub-sequences $\{x_1(n)\}$ and $\{x_2(n)\}$, which correspond to the even and the odd-indexed samples of the input $\{x(n)\}$, respectively,

that is,

$$\begin{aligned} x_1(n) &= x(2n) \\ x_2(n) &= x(2n+1), n = 0, 1, 2, ..., N/2 - 1 \end{aligned} \tag{2.1}$$

Now the N-point DFT can be derived from two half-length DFTs by the decimation-in-time process as follows:

$$\begin{aligned} X(k) &= \sum_{n=0}^{N-1} x(n) W_N^{nk} \\ &= \sum_{n=0}^{\frac{N}{2}-1} x(2n) W_N^{2nk} + \sum_{n=0}^{\frac{N}{2}-1} x(2n+1) W_N^{(2n+1)k} \\ &= \sum_{n=0}^{\frac{N}{2}-1} x(2n) W_{N/2}^{nk} \\ &\quad + W_N^k \sum_{n=0}^{\frac{N}{2}-1} x(2n+1) W_{N/2}^{nk} \end{aligned} \tag{2.2}$$

Similarly, the $(N/2)$-point DFTs can be computed from a pair of $(N/4)$-point DFTs. The decimation process is continued till it contains only two-point DFTs. For a power of two length sequences, decomposition of N-point DFT into 2-point DFTs could be completed in $M = \log_2 N$ steps of decimation.

Figure 2.1 shows the decomposition of 4-point radix-2 DIT FFT using the simplified butterflies which involves 2 stages, each with 2 butterflies per stage. The input data to this is in bit-reversed order and the DFT output is in normal order.

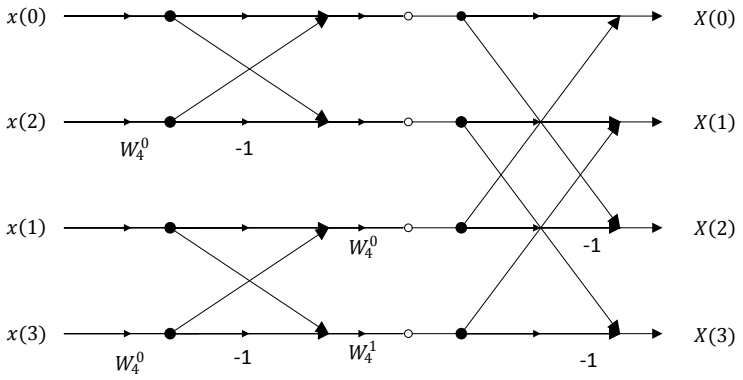

Figure 2.1: Length-4, DIT Radix-2 FFT.

ii) Decimation-in-Frequency Radix-2 FFT Algorithm

This algorithm is based on computing the DFT by decomposition of the sequence of DFT coefficients $X(k)$s into smaller subsequences, hence called as decimation-in-frequency algorithm [32].

In case of radix-2 DIF FFT, the DFT computation is split into two parts such that the first part involves the first $N/2$ data points and the second part involves the next $N/2$ data points, as follows:

$$X(k) = \sum_{n=0}^{\frac{N}{2}-1} x(n)W_N^{nk} + \sum_{n=\frac{N}{2}}^{N-1} x(n)W_N^{nk} \tag{2.3}$$

Since $W_N^{nk} = e^{-j2\Pi k/N}$ and $W_N^{kN/2} = (-1)^k$, equation (2.3) is simplified as:

$$X(k) = \sum_{n=0}^{\frac{N}{2}-1} \left(x(n) + (-1)^k \cdot x\big(n + \frac{N}{2}\big) \right) \cdot W_N^{nk} \tag{2.4}$$

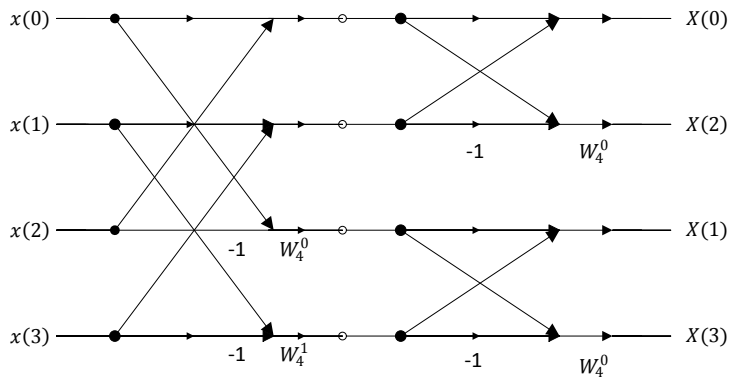

Figure 2.2: Length-4, DIF Radix-2 FFT.

The radix-2 DIF algorithm rearranges equation (2.4) into even-indexed and odd-indexed frequency bins as

$$X(2k) = \sum_{n=0}^{\frac{N}{2}-1} \left(x(n) + x\big(n + \frac{N}{2}\big) \right) \cdot W_{N/2}^{nk} \tag{2.5}$$

$$X(2k+1) = \sum_{n=0}^{\frac{N}{2}-1} \left(\big(x(n) - x\big(n + \frac{N}{2}\big) \cdot W_{N/2}^{nk}\big) \cdot W_{N/2}^{nk} \right) \tag{2.6}$$

According to equations (2.5) and (2.6), the even-indexed and odd-indexed frequency outputs $X(k)$ can be computed by a pair of $N/2$-length DFTs. The entire process involves $M = \log_2 N$ stages of decimation, where each stage involves $N/2$ butterflies. Figure 2.2 shows the flow graph of radix-2 DIF decomposition of a 4-point DFT computation. In this flow graph the input is in normal order and the DFT output is in bit-reversed order. To compute the 4-point DFT, it requires 4 complex multiplications and 8 complex additions.

The computation of N-point DFT via the DIF or DIT FFT algorithms require $(N/2)\log_2 N$ and $N\log_2 N$ number of complex multiplications and complex additions, respectively. For a radix-2 algorithm the operation count can be further reduced by realizing each complex multiplication by 3 real multiplications and 3 real additions (a 3/3 algorithm) [33]. When 3/3 algorithm is used for complex multiplications the arithmetic complexity of radix-2 FFT could be given by:

$$R_M = \frac{3N}{2}\log_2 N - 5N + 8 \tag{2.7}$$

$$R_A = \frac{7N}{2}\log_2 N - 5N + 8 \tag{2.8}$$

where R_M and R_A are the real multiplications and real additions to compute an N-point DFT, respectively.

2.2.2 Radix-4 FFT algorithm

It can be used when the DFT length N is a power of 4 (i.e., $N = 4^M$). Unlike the radix-2 FFT algorithm in the radix-4 algorithm during every step, decimation is carried out by a factor of 4 [19].

In the first step of radix-4 DIT FFT, the input N-point data is split into four subsequences as $x(4n)$, $x(4n+1)$, $x(4n+2)$, and $x(4n+3)$, where $n = 0, 1, ..., N/4-1$.

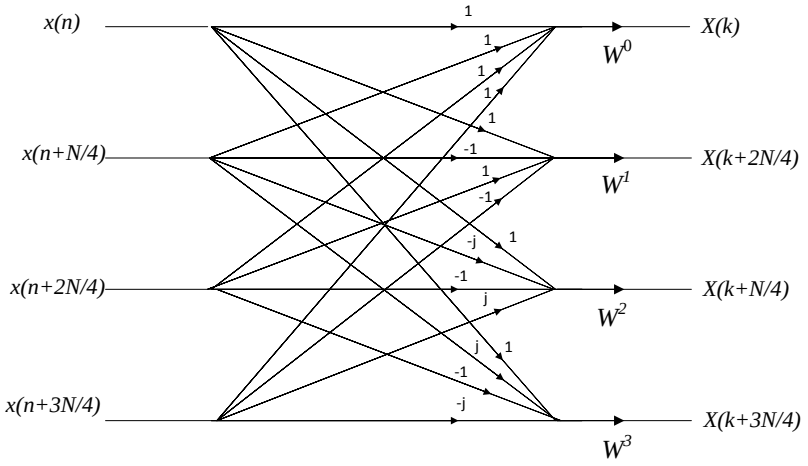

Figure 2.3: Radix-4 FFT butterfly.

Then

$$\begin{aligned} X(k) &= \sum_{n=0}^{N-1} x(n) W_N^{nk} \\ &= \sum_{n=0}^{\frac{N}{4}-1} x(4n) W_{N/4}^{nk} + W_N^k \sum_{n=0}^{\frac{N}{4}-1} x(4n+1) W_{N/4}^{nk} \\ &\quad + W_N^{2k} \sum_{n=0}^{\frac{N}{4}-1} x(4n+2) W_{N/4}^{nk} \\ &\quad + W_N^{3k} \sum_{n=0}^{\frac{N}{4}-1} x(4n+3) W_{N/4}^{nk} \end{aligned} \tag{2.9}$$

As the FFT length of a radix-4 is $N = 4^M$, it requires $M = \log_4 N = \frac{\log_2 N}{2}$ stages of decimation where each stage involves $N/4$ butterflies. The radix-4 FFT butterfly structure is shown in Figure 2.3. The decimation process of each stage is similar to radix-2 algorithm. Since $W_N^0 = 1$, each radix-4 butterfly involves 3 complex multiplications and 8 complex additions [34]. Therefore, the number of complex multiplications

is $\frac{3N}{4}\log_4 N$. Comparing with the radix-2 approach, this requires less number of complex multiplications, although it uses the same number of complex additions. The total operation count for N-point radix-4 FFT is [4]:

$$R_M = \frac{9N}{8}\log_2 N - \frac{43N}{12} + \frac{16}{3} \tag{2.10}$$

$$R_A = \frac{25N}{8}\log_2 N - \frac{43N}{12} + \frac{16}{3} \tag{2.11}$$

2.2.3 Radix-2^i and Higher radix FFT Algorithms

The twiddle factor multiplicative complexity can be reduced by using higher radices like radix-8 [20] or radix-16 [22]. But, the implementation complexity grows as the radix becomes higher. In 1996, He and Torkelson [24] discussed about radix-2^2 and radix-2^3 FFT algorithms. These algorithms have the same number of non-trivial[1] multiplications as radix-4 and radix-8 algorithms, respectively. However, these algorithms differ in the twiddle factors at different FFT stages, but maintain the same butterfly structure of radix-2 algorithm. Followed by He and Torkelson [24], several radix-2^i [28] algorithms are developed for higher radices that include radix-2^4 [26], modified radix-2^4 [26], radix-2^5 [27], and modified radix-2^5 [27] algorithms. The idea of these radix-2^i algorithms is to get simpler butterfly structure with less multiplicative complexity. The following subsection explains the derivation of the radix-2^2 algorithm,

[1]Twiddle factor multiplication by $1, -1$, j and $-j$ are trivial and other multiplications like W_8^1, W_{16}^1 are non-trivial.

which can be extended for higher radices.

i) Radix-2^2 Algorithm

In [24], the authors have proposed a radix-2^2 algorithm using index decomposition technique. To illustrate the derivation of this algorithm, the time and frequency indices for $i = 2$ are decomposed as follows:

$$\begin{aligned} n &= \frac{N}{2}n_1 + \frac{N}{4}n_2 + n_3 \quad \left\{n_1, n_2 = 0, 1, n_3 = 0 \sim \frac{N}{4} - 1\right\} \\ k &= k_1 + 2k_2 + 4k_3 \qquad \left\{k_1, k_2 = 0, 1, k_3 = 0 \sim \frac{N}{4} - 1\right\} \end{aligned} \tag{2.12}$$

Substituting equation (2.12) in (1.1) we can get the following expression:

$$\begin{aligned} X(k_1 + 2k_2 + 4k_3) & \\ &= \sum_{n_3=0}^{\frac{N}{4}-1}\sum_{n_2=0}^{1}\sum_{n_1=0}^{1} x(\frac{N}{2}n_1 + \frac{N}{4}n_2 + n_3) \\ &\quad W_N^{(\frac{N}{2}n_1+\frac{N}{4}n_2+n_3)(k_1+2k_2+4k_3)} \\ &= \sum_{n_3=0}^{\frac{N}{4}-1}\sum_{n_2=0}^{1}\{B_{N/2}^{k_1}(\frac{N}{4}n_2 + n_3)\} \\ &\quad W_N^{(\frac{N}{4}n_2+n_3)(k_1+2k_2+4k_3)} \end{aligned} \tag{2.13}$$

where

$$B_{N/2}^{k_1}(\frac{N}{4}n_2 + n_3) = x(\frac{N}{4}n_2 + n_3) + (-1)^{k_1}x(\frac{N}{4}n_2 + n_3 + \frac{N}{2}) \tag{2.14}$$

The decomposition of common twiddle factor in equation (2.13) is the key difference from the decomposition of the radix-2 algorithm, which can be expressed as

$$W_N^{(\frac{N}{4}n_2+n_3)(k_1+2k_2+4k_3)} = (-j)^{n_2(k_1+2k_2)}W_N^{n_3(k_1+2k_2)}W_{\frac{N}{4}}^{n_3k_3} \tag{2.15}$$

Substituting equation (2.15) in (2.13) the components of N-point DFT could be obtained from four DFTs of length $N/4$ as follows:

$$X(k_1 + 2k_2 + 4k_3) = \sum_{n_3=0}^{\frac{N}{4}-1}\{B_{N/4}^{k_1k_2}(n_3)W_N^{n_3(k_1+2k_2)}\}W_{\frac{N}{4}}^{n_3k_3} \tag{2.16}$$

where

$$B_{N/4}^{k_1k_2}(n_3) = B_{N/2}^{k_1}(n_3) + (-1)^{k_2}(-j)^{k_1}B_{N/2}^{k_1}(n_3 + \frac{N}{4}) \tag{2.17}$$

An N-point DFT is now decomposed into four DFTs of length-($N/4$) DFTs, according to (2.16). Each DFT of length $N/4$ can be further decomposed in the same way until length-2 or length-4 DFTs are reached.

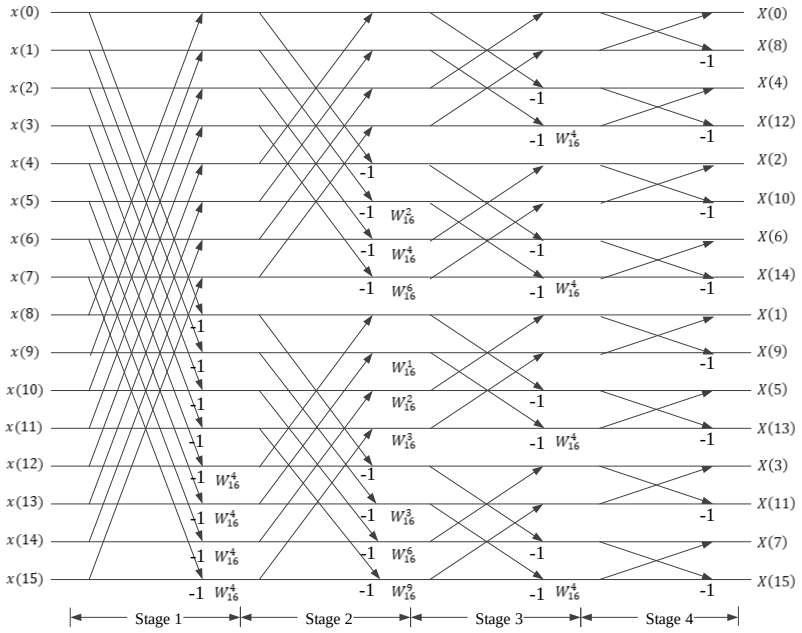

Figure 2.4: Signal flow graph of 16-point radix-2^2 DIF FFT.

Figure 2.4 shows a flow graph of 16-point radix-2^2 DIF FFT. It requires the trivial multiplication by $W_{16}^4 = -j$ in the first and the third stages, whereas it requires non-trivial multiplications in the second stage. This flow graph is different from that of radix-2 algorithm in which non-trivial twiddle factors are needed at the outputs of every stage (except the last one). This algorithm has a great structural advantage compared to other algorithms (radix-2 and radix-4) when they are implemented in pipeline architectures [26].

ii) Higher Radix Algorithms

The linear index decomposition scheme of radix-2^2 algorithm can be extended for higher radices, e.g, radix-2^3, radix-2^4, modified radix-2^4 (Radix-$M.2^4$), radix-2^5 and modified radix-2^5 (Radix-$M.2^5$). The N-point FFT computation with radix-2^i algorithm involves $\log_2 N$ stages. Table 2.1 shows the twiddle factor at each stage to compute the N-point FFT for various radix-2^i algorithms (Number of stages are shown upto 8 in Table 2.1, which can extend to $\log_2 N$ stages). These algorithms have the same butterfly structure but the twiddle factor multiplication structure is varied with the exponent i. The twiddle factor multiplications are classified into trivial (W_4 which is multiplication by $-j$), and other multiplications are non-trivial[28].

From Table 2.1, one can observe that radix-2^3 [24] algorithm requires trivial multiplication at first stage, and non-trivial multiplications at the second and third stages. This type of sequence is repeated for every three stages in order to obtain radix-2^3 algorithm. Radix-2^4 includes trivial multiplication at first stage, and non-trivial multiplications in the next three stages. In [26], a modified radix-2^4 algorithm have been proposed, which requires less number of multiplications. In the modified algorithm of [26], the twiddle factor of third stage W_{16} is transfered to the second stage. A modified radix-2^5 algorithm is suggested in [27], which is a combination of two decomposition methods of radix-2^5 algorithm. In [28], the authors have presented the decomposition

Table 2.1: Twiddle factor at each stage to compute N-point FFT

Algorithm \ Stage	1	2	3	4	5	6	7	8
Radix-2^2	W_4	W_N	W_4	$W_{N/4}$	W_4	$W_{N/16}$	W_4	$W_{N/64}$
Radix-2^3	W_4	W_8	W_N	W_4	W_8	$W_{N/8}$	W_4	W_8
Radix-2^4	W_4	W_8	W_{16}	W_N	W_4	W_8	W_{16}	$W_{N/16}$
Radix$-M.2^4$	W_4	W_{16}	W_4	W_N	W_4	W_{16}	W_4	$W_{N/16}$
Radix-2^5	W_4	W_8	W_{16}	W_{32}	W_N	W_4	W_8	W_{16}
Radix$-M.2^5$	W_4	W_8	W_{32}	W_4	W_N	W_4	W_{16}	W_4

of radix-2^i algorithm which further reduces the number of multiplications. The radix-2^i algorithms have the advantages of lower multiplicative complexity and structural advantage to be used in pipeline architecture.

2.2.4 Split-Radix FFT

The split-radix FFT (SRFFT) algorithm was introduced in [35], but this was clearly described in [23]. The split-radix algorithm decomposes an N-point DFT into one $N/2$-point DFT and two $N/4$-point DFTs as:

$$\begin{aligned} X(k) &= \sum_{n=0}^{N-1} x(n) W_N^{nk} \\ &= \sum_{n=0}^{\frac{N}{2}-1} x(2n) W_{N/2}^{nk} + W_N^k \sum_{n=0}^{\frac{N}{4}-1} x(4n+1) W_{N/4}^{nk} \\ &\quad + W_N^{3k} \sum_{n=0}^{\frac{N}{4}-1} x(4n+3) W_{N/4}^{nk} \end{aligned} \tag{2.18}$$

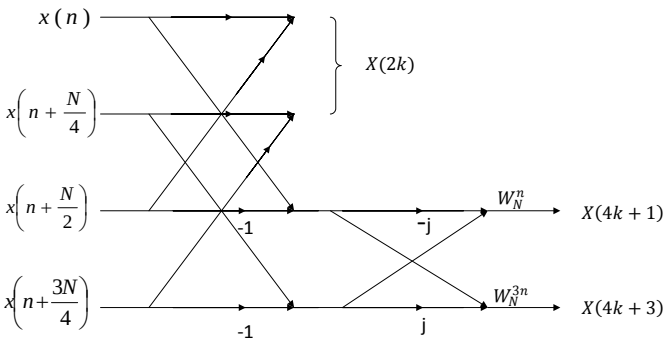

Figure 2.5: Split-Radix FFT.

This algorithm makes use of both radix-2 and radix-4 (radix-2/4) behaviour simultaneously on upper and lower half of the signal flow graph as shown in Figure 2.5. The arithmetic complexity of SRFFT algorithm is given by [23]:

$$R_M = N \log_2 N - 3N + 4 \tag{2.19}$$

$$R_A = 3N \log_2 N - 3N + 4 \tag{2.20}$$

The SRFFT algorithm [23] requires less number of multiplications and additions compared to radix-2 and radix-4 algorithms. Followed by the SRFFT algorithm of [23], many split-radix algorithms [36, 37, 38, 39] were suggested by researchers to further reduce the number of complex multiplications and additions over the radix-2, radix-4 or any higher radix-based algorithms.

Table 2.2: Number of real multiplications to compute a length-N Complex DFT

N	Radix-2	Radix-4/ Radix-2^2	Split Radix
16	24	20	20
32	88	-	68
64	264	208	196
128	712	-	516
256	1800	1392	1284
512	4360	-	3076
1024	10248	7856	7172

Table 2.3: Number of real additions to compute a length-N Complex DFT

N	Radix-2	Radix-4/ Radix-2^2	Split-Radix
16	152	148	148
32	408	-	388
64	1032	976	964
128	2504	-	2308
256	5896	5488	5380
512	13566	-	12292
1024	30728	28336	27652

2.2.5 Computational Complexity for Complex-Valued FFT Algorithms

Table 2.2 and Table 2.3 shows the comparison of the number of real multiplications and real additions to compute an N-point DFT. From these tables one can observe that, the split-radix FFT requires less number of arithmetic operations compared to

the other algorithms. However, the flow graph of this algorithm results in an irregular structure due to the mix of FFTs of different lengths in different parts.

2.3 Real-valued FFT Algorithms

When the input sequence $x(n)$ is real-valued, the DFT components exhibit conjugate symmetry behavior, i.e., $X(k) = X^*(N-k)$. Therefore, we need to compute only half the number of DFT components in this case. But the FFT algorithms for the computation of complex-valued input cannot be used directly to reduce the computational complexity to half, when we want to compute the DFT of real-valued input. FFT of real-valued data and FFT of complex-valued data are generally referred to as real-valued FFT (RFFT) and complex-valued FFT (CFFT), respectively.

Moreover, efficient realization of RFFT has received great attention due to its several important and emerging applications in the area of bio-medical engineering and health-care, audio and video processing, time-series analysis, and many others [40]. Several algorithms are therefore proposed for the RFFT computation. Real-valued FFTs [20] provide area and speed improvement over the CFFTs. The RFFT algorithms are generally tailored for real-valued data by using the trigonometric symmetries and periodicities [20]. In the following sub-section, initially we discuss different approaches for the computation of FFT of real-valued data.

2.3.1 Computation of the RFFT using the CFFT

The simplest way of using the CFFT algorithm to compute the RFFT is to set the real-valued sequence into the real part of complex-valued input and to set the imaginary part of the input values to zero [3]. This approach does not provide significant saving of computation over the CFFT since the intermediate results become complex-valued just after the first stage, when the complex twiddle factors are multiplied. Therefore, doubling algorithm and packing algorithm are proposed to compute the RFFT [3].

i) Doubling Algorithm

In this algorithm a pair of real-valued input sequence is used at a time [27]. The first real-valued data sequence is used as the real part and the second real data sequence as the imaginary part of the complex-valued input sequence of the CFFT. The complex input values thus obtained is expressed as:

$$x(n) = p(n) + j.q(n) \tag{2.21}$$

where $p(n)$ and $q(n)$ are elements of two real-valued data sequences. An N-point CFFT of complex input $\{x(n)\}$ is then obtained as:

$$X(K) = P(K) + j.Q(K) \tag{2.22}$$

Since $p(n)$ and $q(n)$ are real-valued data, the following symmetry holds

$$P^*(N-k) = P(K)$$
$$Q^*(N-k) = Q(K) \tag{2.23}$$

hence follows the output sequence as:

$$X^*(N-k) = P(K) - j.Q(K) \tag{2.24}$$

By using equations (2.22) and (2.24), $P(k)$ and $Q(k)$ can be obtained as:

$$P(k) = \frac{1}{2}\big(X(k) + X^*(N-k)\big)$$
$$Q(k) = \frac{j}{2}\big(X^*(N-k) - X(k)\big) \tag{2.25}$$

In order to separate $P(k)$ and $Q(k)$ according to equation (2.25), $2(N-1)$ extra additions over the normal complex FFT are required. Using the 3/3 algorithm for complex multiplication, the RFFT requires $\frac{1}{2}MN - \frac{3}{2}N + 2$ multiplications and $\frac{3}{2}MN - \frac{1}{2}N$ additions [41]. This algorithm requires almost half of the arithmetic complexity of the CFFT algorithm.

ii) Packing Algorithm

This is another approach to compute N-point FFT of real-valued input using $N/2$-point CFFT [41]. It uses the odd and even indexed samples of the N-point real-valued input sequence to form the $(N/2)$-point complex data. This is called packing algorithm, since it packs the N-point real-valued sequence into $(N/2)$-point complex valued sequence. The real-valued data can be represented in the form of a complex data as:

$$x(n) = x(2n) + j.x(2n+1) \tag{2.26}$$

where $n = 0, 1, 2, ..., N-1$.

Let $p(n) = x(2n)$ and $q(n) = x(2n+1)$, then the DFT output $X(K)$ can be obtained by using CFFT as in doubling algorithm. Therefore, this also requires $2(N-1)$ extra additions to separate the outputs of the CFFT as in the case of doubling algorithm. Moreover, it requires an additional stage to compute the outputs of the RFFT. The corresponding RFFT requires $\frac{1}{2}MN - \frac{5}{4}N$ multiplications and $\frac{3}{2}MN - \frac{1}{4}N - 4$ additions by using 3/3 algorithm for complex multiplications [41].

2.3.2 FFT of Real-Valued data

The reduction in the arithmetic complexity can be obtained by using specific algorithms such as DIT FFT algorithm for the computation of the RFFT. This can be achieved by applying the conjugate symmetric property, and computing only one half of the intermediate outputs in each stage, since the others can be obtained by conjugating those intermediate values. This results with less arithmetic complexity for the radix-2 DIT FFT algorithm [40]. By assuming a 3/3 algorithm, radix-2 DIT FFT for a real-valued sequence require $\frac{3}{4}MN - \frac{5}{2}N + 4$ multiplications and $\frac{7}{4}MN - \frac{7}{2}N + 6$ additions.

The radix-4 and higher radix algorithms [42] for real valued inputs can be obtained in the way similar to that of the radix-2 DIT FFT. As the split-radix algorithm is more efficient in terms of arithmetic complexity than higher radix algorithms. It requires only $\frac{1}{2}MN - \frac{3}{2}N + 2$ multiplications and $\frac{3}{2}MN - \frac{5}{2}N + 4$ additions [40]. However, these algorithms are not valid for the DIF decomposition of the FFT because it is not possible to apply the conjugate symmetry at each stage. In [43], an alternative algorithm is proposed to obtain the same savings for the DIF decomposition. In [44], the authors have proposed a modified radix-2 algorithm for the computation of the RFFT which solves the irregularities of the RFFT. This approach is valid for both DIT and DIF decompositions and could be generalized for any number of points, which is power of 2. In [45], the computation of RFFT was based on a modified

radix-2 algorithm which removes the redundant operations from the flow graph. This modified flow graph contains only real data paths instead of complex data paths in a regular flow graph. In [46], a mathematical formulation was presented for removing the redundancies in the radix-2 DIT RFFT. This formulation regularizes the flow graph in order to compute folded RFFT with a simple control unit.

2.3.3 Fast Hartley Transform

The DFT of real-valued data could be computed from the Discrete Hartley Transform (DHT) [47] of the same data. The DHT of a real valued input sequence is defined as:

$$X(k) = \sum_{n=0}^{N-1} x(n) \left[\cos\left(\frac{2\pi kn}{N}\right) + \sin\left(\frac{2\pi kn}{N}\right) \right] \tag{2.27}$$

for $k = 0, 1, 2.....N-1$.

Unlike the DFT, the DHT takes real-valued input and provides real-valued output. The absence of complex arithmetic makes the DHT faster than the DFT. Algorithms similar to the radix-based FFT can also be applied to DHT computations called as Fast Hartley Transform (FHT) algorithm. Generally, FHT algorithms involve the same multiplications and $(N-2)$ more addition compared to the corresponding FFT

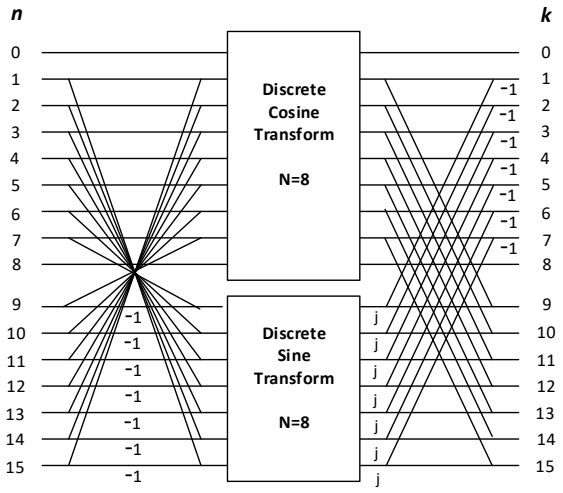

Figure 2.6: Quick Discrete Fourier Transform

algorithm. The split-radix FHT algorithm requires $\frac{2N}{3}\log_2 N - \frac{19N}{9} + 3 + \frac{(-1)^M}{9}$ multiplications and $\frac{4N}{3}\log_2 N - \frac{14N}{9} + 3 + (-1)^M\frac{5}{9}$ additions.

An N-point DFT of real-valued data can be computed from the DHT of the same data as follows:

$$\begin{aligned} Re(DFT(k)) &= \frac{DHT(k) + DHT(N-k)}{2} \\ Im(DFT(k)) &= \frac{DHT(k) - DHT(N-k)}{2} \end{aligned} \tag{2.28}$$

Table 2.4: Number of real multiplications to compute the DFT for a real-valued input

N	CFFT Direct	CFFT Packing	CFFT Doubling	Radix-2 RFFT	Split-radix RFFT	FHT	Quick DFT
16	20	12	10	12	10	12	11
32	68	40	34	44	34	42	37
64	196	112	98	132	98	124	105
128	516	288	258	356	258	330	273
256	1284	704	642	900	642	828	673
512	3076	1664	1538	2180	1538	1994	1601
1024	7172	3840	3586	5124	3586	4668	3713

2.3.4 Quick Discrete Fourier Transform

This algorithm computes the DFT via Discrete Cosine Transform (DCT) and Discrete Sine Transform (DST). It decomposes the N-point DFT into $(N/2+1)$-point DCT and $(N/2-1)$-point DST. The Quick DFT for 16-point data is shown in Figure 2.6. This computes the DCT and the DST seperately by taking the complex operations at the last stage. The arithmetic operations required by this algorithm to compute the N-point DFT are as follows:

$$R_M = \frac{N}{2}\log_2 N - \frac{11}{8}N + 1 \tag{2.29}$$

$$R_A = \frac{7}{4}N\log_2 N - 3N + 2 \tag{2.30}$$

Table 2.5: Number of real additions to compute the DFT for a real-valued input

N	CFFT Direct	CFFT Packing	CFFT Doubling	Radix-2 RFFT	Split-radix RFFT	FHT	Quick DFT
16	148	88	88	62	60	64	66
32	388	228	224	170	164	166	186
64	964	556	544	442	420	416	482
128	2308	1308	1280	1082	1028	998	1186
256	5308	3004	2944	2586	2436	2336	2818
512	12292	6780	6656	5978	5636	5350	6530
1024	27652	15100	14848	13658	12804	12064	14580

2.3.5 Computational Complexity for Real-Valued FFT Algorithms

Although most of the FFT algorithms are developed for complex-valued inputs, by taking the advantages of redundancies and trigonometric symmetries, the computational complexity is reduced in all of these RFFT algorithm. The number of real multiplications and real additions required for the operation of real-valued are shown in Table 2.3 and Table 2.4, respectively. If a CFFT is used directly for real inputs, it requires more arithmetic complexity. The packing and oubling algorithms involve more additions than a Split-radix RFFT algorithm [40] for a real-valued input.

Split-radix FHT requires less number of multiplications and additions than radix-2 RFFT for N greater than 16. However, it requires more number of multiplications and additions than split-radix RFFT. The Quick DFT algorithm requires more real multiplications than the doubling algorithm.

2.4 Special cases of the FFT algorithms

The FFT algorithm could be optimized for some special cases, e.g., when only a part of the output is desired or when there are a large number of zeros in the input or when the input is non-power of two or multidimensional inputs. In this subsection, we discuss some special cases of FFT algorithms that are useful for specific applications like, Third Generation Partnership Project (3GPP) Long-Term Evolution (LTE) [18], and modern microscopy [48], and radar signal processing.

2.4.1 FFT Pruning

If the data sequence contains 2^l non-zero data points out of 2^m data points, where $m > l$, then the corresponding FFT can be computed by means of the pruned FFT which accomplishes time saving. A slight modification to radix-2 DIT algorithm allows a time-saving of approximately *(m-l)/m* where 2^m points are transformed of which only 2^l are non-zero [49].

The FFT pruning for $l = 2, m = 3$ is shown in Figure 2.7. There are four non-zero data points and three stages. Pruning is applied to first stage, but second and third stages cannot be pruned [49]. When pruning is applicable, we compute only the partial butterflies instead of entire butterflies. In general, if there are 2^l non-zero data points in a set of 2^m data points, then the number of stage(s) where pruning can be applied

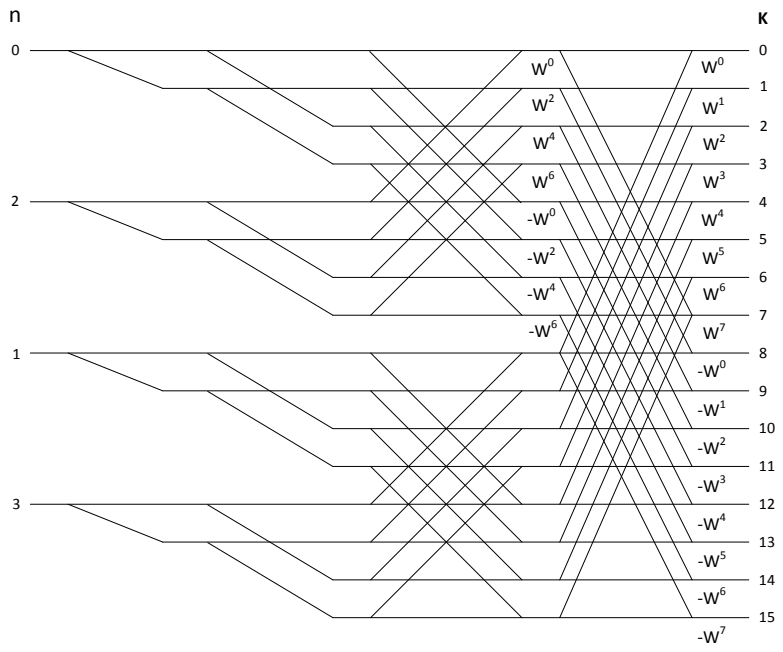

Figure 2.7: FFT pruning.

$(m - l)$ stages. FFT pruning is used when there are a large number of zero's that are known in the input. However, it only allows l to be a power-of-two. The asymptotic run-time of the pruned FFT is $O(N.\log a)$, where N is the FFT length and a is the number of non-zero inputs. The main drawback of pruning is that the data sequence is to be known in advance, so that one can find the non-zero input values.

2.4.2 Fast Fourier Transform of Sparse Input

The computation time of DFT generally corresponds to its size N. However, in most of the applications like spectrum sensing and radar signal processing, only a few selected output of the FFT is used. An algorithm to compute those coefficients of its Fourier

transform is called Sparse FFT (SFFT) [50], whose runtime is sublinear in the signal size N.

In [51], the first sublinear algorithm was presented, which is followed by several other sublinear algorithms [52, 53, 54] were developed for Fourier transform over the complex input. There are several versions of SFFT algorithms described in [50]. A hardware implementation of SFFT algorithm is recently published in [55]. However, it is implemented for a specific signal size. Therefore, in [56], the authors have presented the hardware implementation of a million-point SFFT design, that can provide configurable parameters. Robust Sparse Fourier Transform (RSFT), which is a modification of Sparse Fourier Transform (SFT) is presented in [57] that extends the SFT advantages which are useful for short-range radar signal processing. It is shown that the RSFT is robust in detecting frequencies when exact knowledge of signal sparsity is not available. It has further investigated the trade-off between detection performance and computational complexity [57].

2.4.3 Scaled DFTs

In certain applications like orthogonal frequency division multiplexing demodulation and modern microscopy [58], the input length of DFT is length-$q*2^M$, where q is a odd number. However, fast algorithms for such sequence lengths generally require complex computational structure and are less efficient than that of power-of-two length DFTs.

Zero-padding technique was often used to get the DFT of such sequence lengths. However, this technique requires more computations. Therefore, a scaled DFT has been proposed in [58], which can be flexibly used for length-$q * 2^M$ DFTs. Several algorithms [39, 58, 59, 60] have been proposed thereafter, in order to further reduce the arithmetic complexity for scaled DFT computation.

2.4.4 Multidimensional FFTs

The multidimensional fast Fourier transforms (FFTs in 2D or more dimensions) are used in many applications such as image processing, applied physics etc. These applications require large amount of computations.

The general form of the multidimensional FFT is as follows:

$$\begin{aligned} X(u_1, u_2, ..., u_m) &= \sum_{v_1=0}^{N_1-1} \sum_{v_2=0}^{N_2-1} ..... \sum_{v_m=0}^{N_m-1} W_{N_1}^{u_1 v_1} W_{N_2}^{u_2 v_2} ..... \\ & \quad W_{N_m}^{u_m v_m} x(v_1, v_2, ..., v_m) \end{aligned} \tag{2.31}$$

where $W_{N_k} = exp(\frac{-2\pi j}{N_k})$, $u_k = 0, 1, ..., u_k - 1$; u_k is the length of the k^{th} dimension $k = 1, 2, ..., m$ and $x(v_1, v_2, ..., v_m)$ are the complex input data sequences.

Equation (2.31) is converted into m one-dimensional FFTs in order to simplify the computation as follows:

$$X(u_1, u_2, ..., u_m) = \sum_{v_1=0}^{N_1-1} W_{N_1}^{u_1 v_1} \sum_{v_2=0}^{N_2-1} W_{N_2}^{u_2 v_2} ..... \sum_{v_m=0}^{N_m-1} W_{N_m}^{u_m v_m} x(v_1, v_2, ..., v_m) \quad (2.32)$$

This provides the simplest algorithm where each one-dimensional FFT can be computed by the Cooley-Tukey FFT [1], so this algorithm is known as row-column algorithm [61].

Several algorithms have been proposed for the multidimensional FFTs such as the vector-radix algorithms, the polynomial transform algorithms and the split vector-radix algorithms [62, 63, 64]. These algorithms reduces the complexity over row-column algorithm. In [65], a fast algorithm has been derived based on vector coding for multidimensional integral points. This algorithm has reduced the multiplication complexity and the number of recursive stages without increasing the number of additions. However, the most popular one among these algorithms is the row-column decomposition algorithm, due to its simple structure and easy to program.

2.4.5 Quantum Fourier Transform

The Moore's law [66] has been consistent for several decades, but sustaining the pace of scaling has become increasingly difficult in recent years. To meet the performance

and power requirements of exa-scale systems, quantum computers may be one of the alternatives which possibly could offer exponential speedup for certain types of calculations.

The Quantum Fourier Transform (QFT) is used in Quantum computers, which is similar to FFT [67] . But the QFT operates on quantum bits instead of operating on vector elements. If 2^p elements are considered for both transforms, these can take $p2^p$ operations and $p(p+1)/2$ operations to compute FFT and QFT, respectively. By comparing the quantity of operations, it is evidenced that the QFT is efficient than the FFT. Nowadays, significant attention is given for research to implement the QFT algorithms [68, 69, 70].

Basic quantum computers are developed in many labs across the world. Companies such as Microsoft, IBM and Google are all developed their own prototypes [71]. However, these prototypes are very simple with only a small number of qubits. The Quantum hardware emulation is also critical in developing practical QFT algorithms before large-scale quantum computer becomes viable. Therefore, a comprehensive methodology to perform accurate mapping of quantum algorithm for FPGA emulation purposes have been demonstrated through the emulation of QFT hardware in [68, 69, 70, 72].

2.5 Conclusion

This chapter briefly reviewed three classes of FFT algorithms that comprises of complex valued FFTs, real-valued FFTs and special cases of FFTs. In complex valued FFTs, the radix-2 DIT and DIF algorithms have simple structure that makes easy to implement and is suitable for generic FFT implementation. However, these algorithms require large memory to store data at inner stages, which increases the hardware complexity for implementation. The radix-4 or higher radix algorithms require less multiplications than radix-2 algorithm. However, this algorithm is suitable only when N is a power of 4. This chapter also discussed radix-2^2 and higher-radix algorithms using index decomposition technique. The radix -2^2 algorithm has the same multiplicative complexity as radix-4 algorithm, but retains the butterfly structure of radix-2 algorithm, which is very suitable for ASIC implementation. As compared to radix-based algorithms, the split-radix FFT algorithm require less number of multiplications and additions. However, the FFT implementation is difficult due to its irregular structure.

In real-valued FFTs, the RFFT computations can be obtained by using the CFFT (doubling algorithm and packing algorithm) based on complex conjugate symmetry. In the direct use of CFFT for RFFT computation, the complexity increases as the imaginary part of CFFT is considered by making it as zero. In the other CFFT based techniques (doubling algorithm and packing algorithm), the arithmetic complexity is much less than the direct computation of CFFT. This chapter also discussed specific

algorithms for the computation of RFFT based on radix-2 and split-radix algorithm. Among these split-radix requires fewer operations than a radix-2 or even for higher radix algorithms.

Further we have discussed the special cases of the FFT algorithms based on some constraints. FFT pruning is considered when the data sequence contain large number of zeros. However, the data sequence has to be known in advance. By using sparse FFT, where fewer than N inputs are required and the data sequence does not known in advance. Scaled DFTs are preferred only for length-$q * 2^M$ DFTs. Multidimensional FFTs are required for 2-D or higher dimensional FFTs. A brief description about Quantum Fourier Transform is also discussed which are going to be implement in future. These are the special cases of FFT algorithm that further reduces the hardware complexity of the algorithm. However, these algorithms are considered only for specific applications.

Nowadays, based on literature research, either radix-2^i or combination of radix-2^i algorithms are most suitable for FFT implementation of OFDM system. This thesis is constrained to design FFT for OFDM system. So, more detailed analysis is discussed in the next chapter 3 combined with architecture choice.

Chapter 3

FFT Architectures and Design choice for OFDM systems

The key to high performance in FFT hardware is to have the computational elements organized in such a way that they match the structure of the computational algorithm. In chapter 2, FFT algorithms were discussed particularly about the optimization of the number of operations. In real implementations, often the number of operations is not as important as the amount of resources required and the utilization of those.

The FFT architecture generally consists of one or more processing elements (PEs) that includes butterflies and twiddle factor multipliers with memory and data management circuits. Each butterfly comprises of adder and subtractor, which is usually

reused to compute several butterflies of the FFT algorithm. Similarly, complex twiddle factor multipliers can be reused to perform several twiddle factor multiplications. This reuse of the processing element reduces the area of circuit.

After the Cooley and Tukey's publication along with various FFT algorithms, several different architectures came into existence for implementing those FFT algorithms. The most common FFT architectures are the Pipelined architectures and the Memory based architectures. Among them, the selection of the algorithm depends on the signal processing specifications such as accuracy, application specifications such as latency and throughput, and hardware device specifications which include area and power consumption. In Section 3.1, we discuss about various memory based architectures. Further, we explain different types of pipelined architectures in Section 3.2. At the end of this chapter, we discuss about the design choice (FFT algorithm and architecture) for OFDM based applications.

3.1 Memory based Architectures

Memory based architectures consists of processing elements that calculate all butterflies and twiddle factor multiplication of the algorithm. This requires one or more memories to store the value. For large size FFTs, this is an efficient architecture with less hardware. In this section, we discuss about the single memory, dual memory, array memory and cached memory architectures.

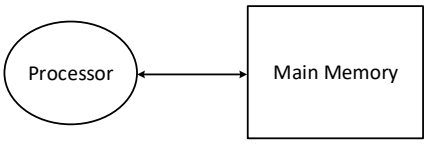

Figure 3.1: Single memory architecture.

3.1.1 Single Memory

In chapter 2, Figure 2.1 shows the traditional 4-point DIT radix-2 FFT flow diagram with two stages. This can be implement by using single memory architecture as shown in Figure 3.1. This architecture contains a processor (an arithmetic block for processing the data) and a memory (to store inputs or outputs) connected to a bi-directional data bus. We can see that for each of the two stages, two butterfly calculations are required and the whole data are read and written back to the main memory in every stage. As the FFT size increases, the number of stages also increases. As a result, the data movement will cause much power consumption and long execution time. In this architecture single memory is used for storing the inputs and the processed outputs which takes more execution time.

3.1.2 Dual Memory

The problem that arises in single memory can be solve by adding extra memory for storing the incoming data. The dual memory structure connects the processor with two memory blocks as shown in Figure 3.2. The data inputs from one memory are

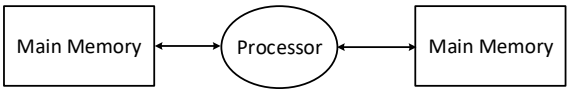

Figure 3.2: Dual memory architecture.

passed through the processing element to another memory and vice versa till the transform is completed. At first, the data is read from the first memory and it is processed in the arithmetic block. Later, the data are written to the other memory. The data are now read from the second memory block and are written into the first memory block. In a pipelined way, the next data are read from the first memory block and processed. The same process repeats for the next stage till the transform is completed. The Honeywell DASP processor [73] use the dual-memory architecture.

3.1.3 Array Memory

The whole processing of an array memory architecture is shown in Figure 3.3, which is divided into a number of independent processing elements with local buffers. They are connected through a network. This architecture is mainly used for large number of data processors. This architecture has high throughput and low latency, but the hardware cost is increased as compared to single memory architecture. The Cobra FFT processor [74] uses an array architecture with multiple chips where each chip contains one processor and one local buffer. The FFT processor by O'Brien et al.,

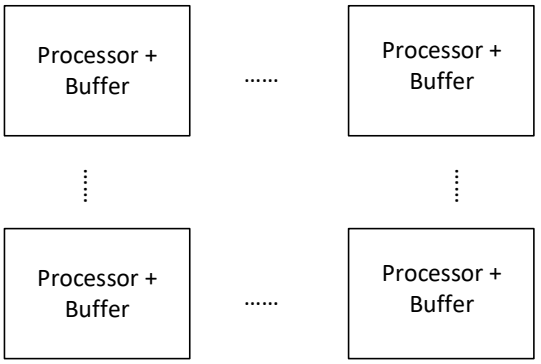

Figure 3.3: Array memory architecture.

[74] uses an array-style architecture on a single chip with four data paths and four memory banks.

3.1.4 Cached Memory

The cached memory architecture [75, 76] is similar to single memory architecture except that a small cache memory is present in between the processor and the memory to reduce the number of main memory access as shown in Figure 3.4. In conventional FFT, architecture data and identical twiddle factors are stored in main memory itself. If the number of points of a FFT is increased, the size of the memory is increased, and simultaneously the speed of the processor is reduced. To overcome this problem, data and identical twiddle factors are stored in cache memory. The size of the cache memory is very small compared to main memory and the speed of the cache memory is very fast compared with main memory. Cached memories architectures are mainly used

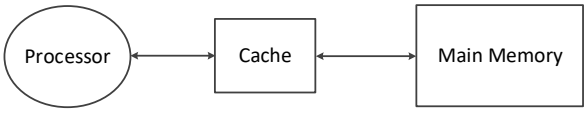

Figure 3.4: Cached memory architecture.

for reducing the power consumption, since the frequent access of the main memory are reduced.

3.2 Pipelined Architectures

A Pipeline implementation of the FFT was first proposed in [77] by Groginsky. In Pipelined architectures, each stage has its own set of processing elements. Each stage reads a series of FFT samples, processes them sequentially and gives the outputs to the next stage. All stages are computed as soon as the data are available. At the input stage all the processing is done in serially, therefore the memory requirement is low to store input data. Furthermore, by adding registers it is possible to increase the clock frequency to divide the critical path. Hence the architectures are suitable for real time applications. In [24], He and Torkelson have been classified the pipeline architectures into two categories: Delay Feedback and Delay Commutator. These architectures are discussed briefly in the subsequent sub-sections.

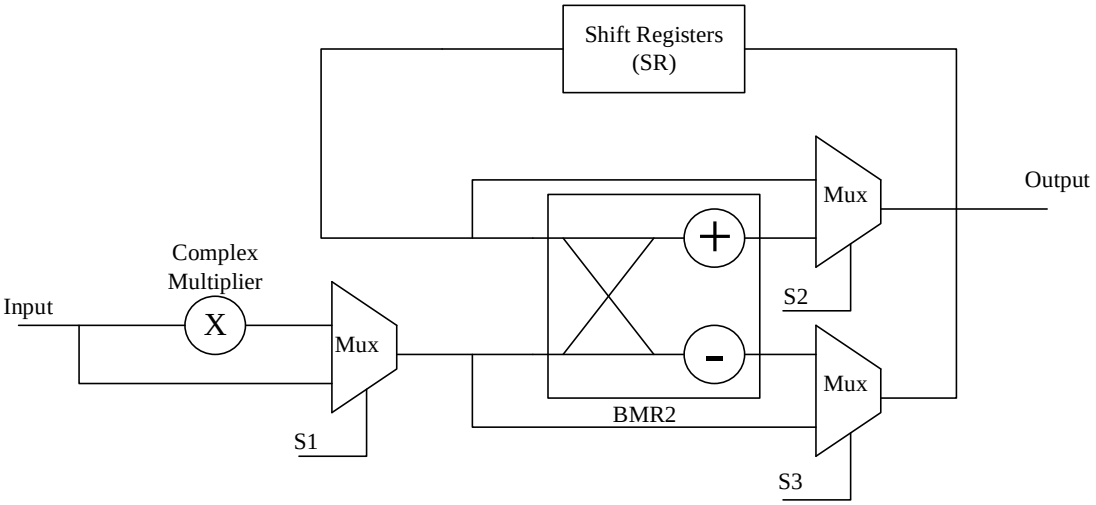

Figure 3.5: Basic building blocks of R2SDF architecture.

3.2.1 Delay Feedback Architectures

3.2.1.1 Single-path Delay Feedback (SDF)

The Single-path Delay Feedback architecture contains a butterfly processing element and a feedback shift register (SR) at each stage. The input and output data of each stage share the same shift register. This makes the shift registers that are more efficiently used in the Radix-2 SDF (R2SDF) and Radix-4 SDF (R4SDF) architectures.

The SDF architectures are either designed based on the DIF FFT algorithm or can be re-designed based on the DIT FFT algorithm. Now we describe, the R2SDF architecture that uses a simple strategy to schedule the computations of the DIF FFT signal flow graph of Figure 2.2. The basic building blocks of this architecture are shown in Figure 3.5. The block diagram comprises of

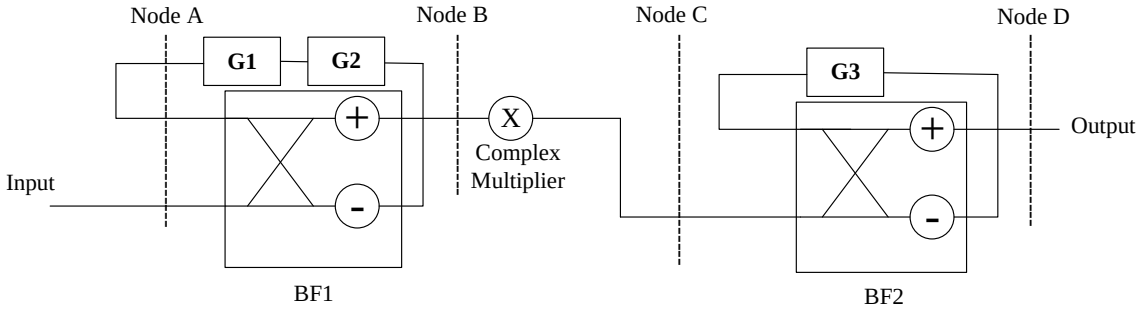

Figure 3.6: R2SDF 4-point DIF FFT architecture.

1. Butterfly module using radix-2 (BMR2) which performs the addition and difference operation.

2. A complex multiplier which performs multiplication of the twiddle factor and the input comes to it.

3. Shift registers to store the intermediate data.

4. Multiplexers to pass the input data into the shift register without any computation or to pass the input data into the butterfly module to perform the computation.

The R2SDF implementation of the 4-point DIF FFT is shown in Figure 3.6. It requires two butterfly modules (BF1 and BF2), three shift registers (G1, G2 and G3) and a complex multiplier. For simplicity multiplexers are not shown in Figure 3.6. It also shows different nodes (Node A, Node B, Node C, Node D), in order to explain the data output order of this architecture as shown in Table 3.1.

Table 3.1: Data output order of the R2SDF pipelined architecture for 4-point FFT

Cycle	Node A	G1	G2	Node B	Node C	G3	Node D
1	-	-	$x(0)$	-	-	-	-
2	-	$x(0)$	$x(1)$	-	-	-	-
3	$x(0)$ $x(2)$	$x(1)$	$x(0)-x(2)$	$x(0)+x(2)$	-	-	-
4	$x(1)$ $x(3)$	$x(0)-x(2)$	$x(1)-x(3)$	$x(1)+x(3)$	-	$x(0)+x(2)$	-
5	-	$x(1)-x(3)$	-	$x(0)-x(2)$	$x(0)+x(2)$ $x(1)+x(3)$	$X(2)$	$X(0)$
6	-	-	-	$x(1)-x(3)$	-	$x(0)-x(2)$	$X(2)$
7	-	-	-	-	$x(0)-x(2)$ $-jx(1)+$ $jx(3)$	$X(3)$	$X(1)$
8	-	-	-	-	-	-	$X(3)$

Basically the operation of R2SDF architecture is as follows. In the first two cycles, the first butterfly (BF1) module allows the inputs $x(0)$ and $x(1)$ to pass unchanged into the shift registers G1 and G2. In the third cycle at node A, the data $x(0)$ in shift register G1 and the incoming data $x(2)$ are inputs to butterfly (BF1) which computes a 2-point DFT. Then, the output $x(0)+x(2)$ is sent to the complex multiplier which multiplies with the twiddle factor, while $x(0)-x(2)$ is sent back to the shift register G2. In the fourth cycle, the second butterfly (BF2) module allows the input $x(0)+x(2)$ to pass unchanged into shift register G3, while BF1 computes the inputs $x(1)$ and $x(3)$ and the outputs $x(1)+x(3)$ and $x(1)-x(3)$ which are computed and sent to the complex multiplier and to the shift register G2 respectively. In the fifth cycle, the second butterfly (BF2) module computes the inputs $x(0)+x(2)$ and $x(1)+x(3)$, where the DFT result $X(0)=x(0)+x(2)+x(1)+x(3)$ appears at the output node D, while $X(2)=x(0)+x(2)-x(1)-x(3)$ is sent to shift register G3. In the next three

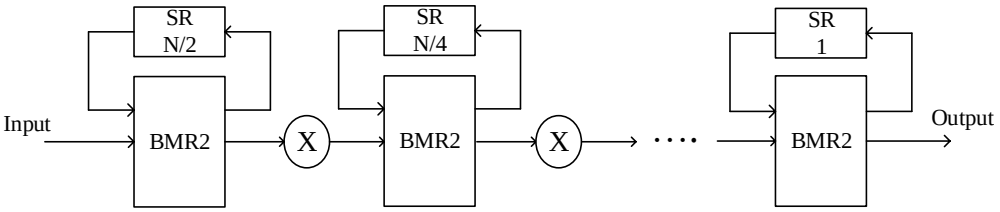

Figure 3.7: N-point R2SDF Architecture.

consecutive cycles, the remaining DFT outputs $X(2)$, $X(1) = x(0) - x(2) - jx(1) + x(3)$ and $X(3) = x(0) - x(2) + jx(1) - jx(3)$ are obtained in the bit-reversed order.

The number of pipeline stages can be extended to N-point DFT as shown in Figure 3.7. For radix-2, multiplexers of first stage switch their position after $N/2$ clock cycles and for every $N/4$ clock cycles in the consequent stages. Likewise, the memory requirement is only $N - 1$ as it requires N cycles to provide first output. In general, a R2SDF architecture for FFT is considered as an optimal choice in terms of the hardware cost and performance for many applications.

Radix-4 Single-path Delay feedback (R4SDF):

This architecture [78] is a radix-4 version of R2SDF. A 64-point DIF R4SDF FFT architecture is illustrated in Figure 3.8. It comprises of butterfly module using radix-4 (BMR4), three shift registers per butterfly with various lengths and complex multipliers. Since radix-4 algorithm is used, the number of multipliers are reduced to $(\log_4 N)$–1 compared to $2((\log_4 N)$–1) for R2SDF. However, the hardware requirement of R4SDF is more compared to R2SDF.

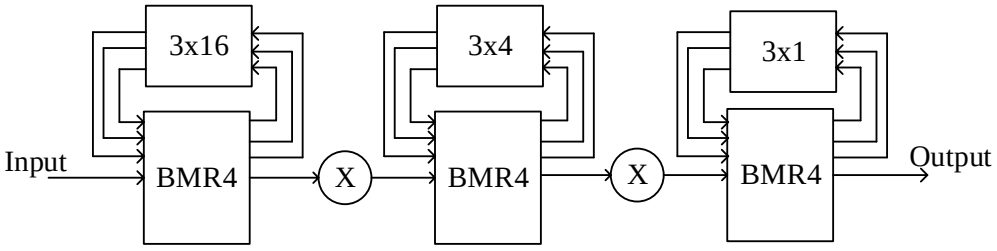

Figure 3.8: Length-64, R4SDF Architecture.

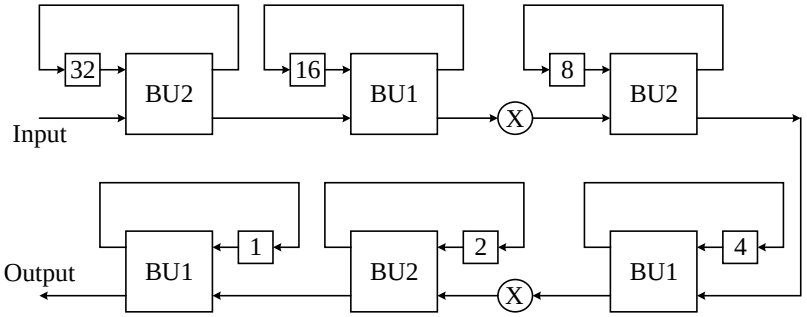

Figure 3.9: Length-64, R2^{2}SDF Architecture.

Radix-2^2 Single-path Delay Feedback (R2^{2}SDF):

R2^{2}SDF architecture is based on radix-2^2 algorithm which is shown in Figure 3.9. By using a pair of modified radix-2 butterflies, this architecture reproduces the radix-4 butterfly elements. In Figure 3.9, BU1 is a standard BMR2, which is similar in the R2SDF pipeline. The BU2 element is slightly modified, that allows selected inputs to be multiplied by ' $-j$ '. This architecture requires non-trivial multipliers at every second stage, whereas in R2SDF it requires at every stage as shown in Figure 3.7. Thus, this architecture reduces the number of multipliers required compared with the R2SDF architecture.

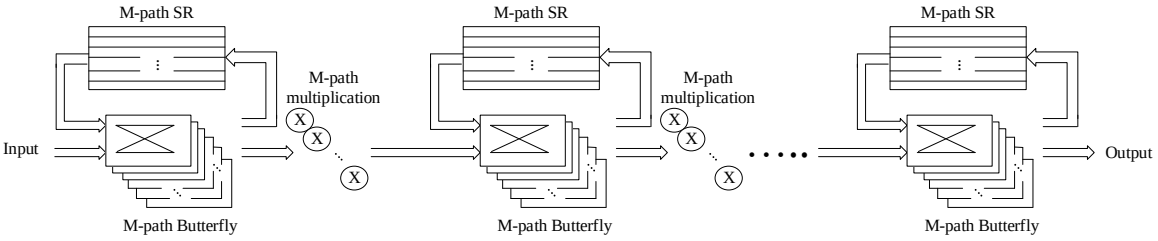

Figure 3.10: Multi-Path Delay Feedback Pipelined FFT Architecture.

3.2.1.2 Multi-path Delay Feedback (MDF)

Multi-path Delay Feedback pipelined FFT architectures are also referred as parallel feedback architectures. These architectures have parallel SDF pipelined FFT architectures for processing the input samples [79, 80, 81, 82]. The MDF architecture utilizes 50% of butterflies. Therefore, parallelism does not improve the utilization ratio. As this architecture process the input samples parallely, it has high throughput at the expense of hardware cost.

MDF architectures can use different radices, such as radix-2 [80], radix-2^2 [82], and radix-2^4 [79]. Figure 3.10 shows the radix-r MDF architecture. This architecture contains radix-r butterfly blocks which are used to compute r-point FFTs. As shown Figure 3.10, it requires complex multipliers in each path for twiddle factor multiplication. In order to reduce the number of complex multipliers, this architecture uses the specified constant multipliers that are based on sharing the same complex multiplier. Furthermore, shift registers are used in the feedback loop, which feed the samples to butterfly stages according to data flow requirements of the circuit.

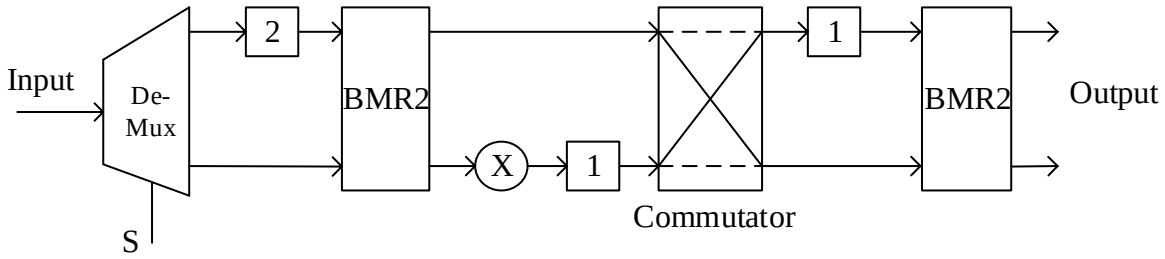

Figure 3.11: R2MDC 4-point FFT architecture.

3.2.2 Delay Commutator Architectures

3.2.2.1 Multi-path Delay Commutator Architectures

The most straight forward implementation of FFT algorithms are Radix-2 Multi-path Delay Commutator (R2MDC) [83] and Radix-4 Multi-path Delay Commutator (R4MDC) [84] pipelined FFT architecture. These architectures operates by rescheduling the butterfly inputs through delay elements, therefore they are known as the feed-forward FFT architectures. The main difference compared to delay feedback architectures is that, these architectures do not have any feedback loop.

The MDC architectures can also be designed based on DIF FFT algorithm or DIT FFT algorithm. As Figure 3.11 illustrates the architecture of an 4-point R2MDC DIF FFT that consists of two butterfly modules, delay elements, a complex multiplier, a de-multiplexer and one commutator (direct pass/ cris-cross switches).

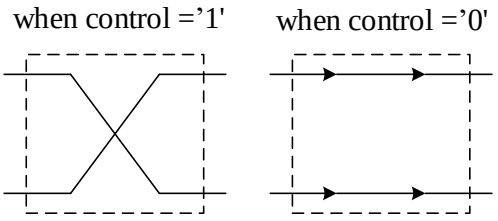

Figure 3.12: Operation of the commutator in R2MDC architecture.

The butterfly module performs the addition and subtraction operations. The delay elements are used to schedule the data properly to the butterflies for proper computation. Complex multiplier performs the complex multiplication of twiddle factors and the complex input data arrives to it. The de-multiplexer is used to send half of the input data to the delay element and remaining half data to the butterfly, which provides correct input order to the butterfly. The operation of the commutator which uses dynamic switches are shown in Figure 3.12. When control='0', the switch is in the direct pass mode till the upper branch outputs are shifted into the delay elements. The switch is toggled to criss-cross mode when control='1', providing the correct ordered input pairs to the next butterfly stage.

The step-by-step process of the R2MDC architecture is shown in Figure 3.13. The input data is considered as $0, 1, 2$, and 3. As data arrives to the R2MDC architecture, the first two input data (0 & 1) are de-multiplexed to the top-left delay elements which are delayed by two samples and the next two data directly to the butterfly. In this way, the first two data (0 & 2) and the next two data (1 & 3) arrives to the butterfly

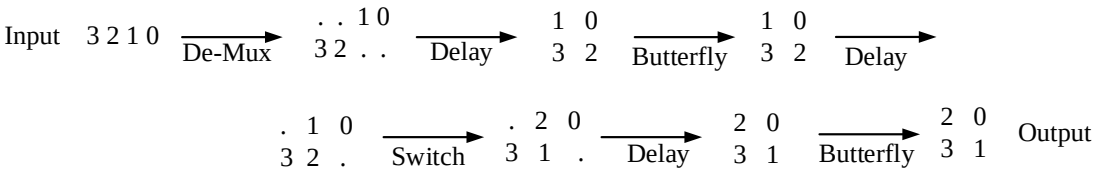

Figure 3.13: Step-by-step process of an 4-point R2MDC FFT architecture.

for processing as a correct pair. Then, the upper branch output of the butterfly is sent directly to the commutator, while the lower branch output is passed through the complex multiplier with one delay sample to the commutator as shown in Figure 3.13. The switch is in the direct pass mode till the first output data is shifted to the delay element, and then the switch is toggled to criss-cross mode thereby providing the correct data to the next butterfly with one unit of delay. Finally, the first two outputs are arrived at the output of the butterfly after processing. In the next cycle, the remaining two outputs also arrive.

The 4-point MDC pipeline structure can be extend to N-point FFT architecture as shown in Figure 3.14. As the architecture computes the input data samples in parallel, it can provide higher throughput compared to SDF architecture but, the hardware cost is more.

The architecture of R4MDC [84] is similar to R2MDC, where the input data are separated by 1-to-4 de-multiplexer and $3N/2$ delay elements at the first stage. A four-path delay commutator is used between two stages. The computation takes place

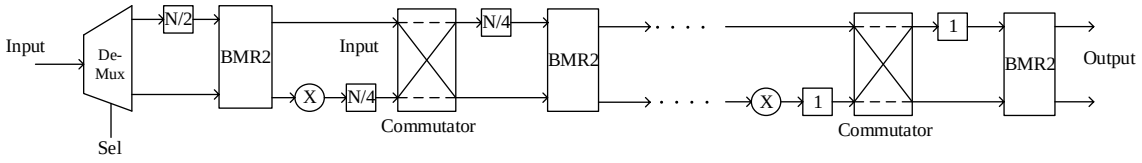

Figure 3.14: R2MDC N-point FFT architecture.

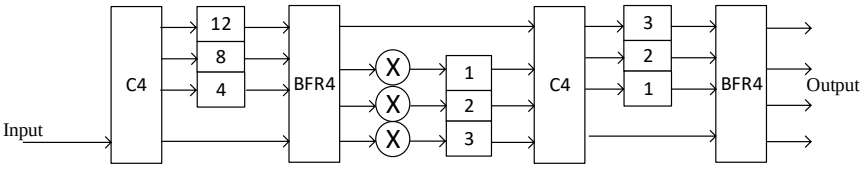

Figure 3.15: Length-16, R4MDC Architecture.

only when the last part of the data is multiplexed to the butterfly. A length-16 DIF Radix-4 Multipath Delay Commutator (R4MDC) FFT is shown in Figure 3.15. Each stage (except the last stage) has 3 multipliers and this architecture requires in total $3(\log_4 N - 1)$ multipliers for an N-point FFT which is more than the R2MDC or R2SDF. Moreover the memory requirement is $5N/2 - 1$, which is the large compared to R2SDF and R2MDC. From the view of hardware and butterfly utilization, it is not a good structure.

3.2.2.2 Single-path Delay Commutator Architectures

In [85], G. Bi and E. V. Jones proposed a simplified radix-4 butterfly, which produces one output when compared to four in the conventional butterfly of R4MDC.

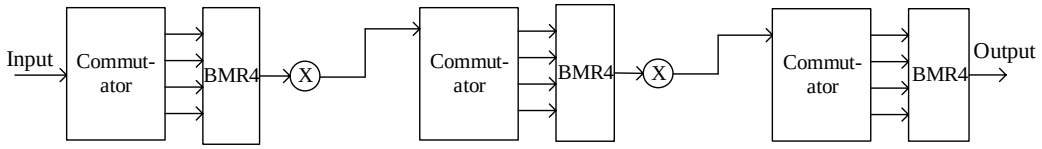

Figure 3.16: Length-64, R4SDC Architecture.

To provide the same four outputs, the butterfly works four times instead of just once. Furthermore, the simplified butterfly needs additional control signals, and so do the commutators. This architecture requires only $2N-2$ memory elements by using combined delay commutators when compared to R4MDC that requires $5N/2-1$ memory elements. The architecture of a 64-point DIF Radix-4 Single-Path Delay Commutator (R4SDC) FFT is shown in Figure 3.16. The main drawback of SDC architecture is low throughput compared to MDC architecture.

3.3 Design choice for OFDM systems

The basic block diagram of FFT based OFDM system is shown in Figure 3.17. The serial input data stream is converted into parallel data stream and mapped to symbols from constellation mapper (Binary Phase Shift Keying/Quadrature Phase Shift Keying/Quadrature Amplitude Modulation). Then these symbols are given to IFFT that generates a digital OFDM symbol with N orthogonal subcarriers. The output of the IFFT is serialized and converted to analog signal using Digital to Analog converter

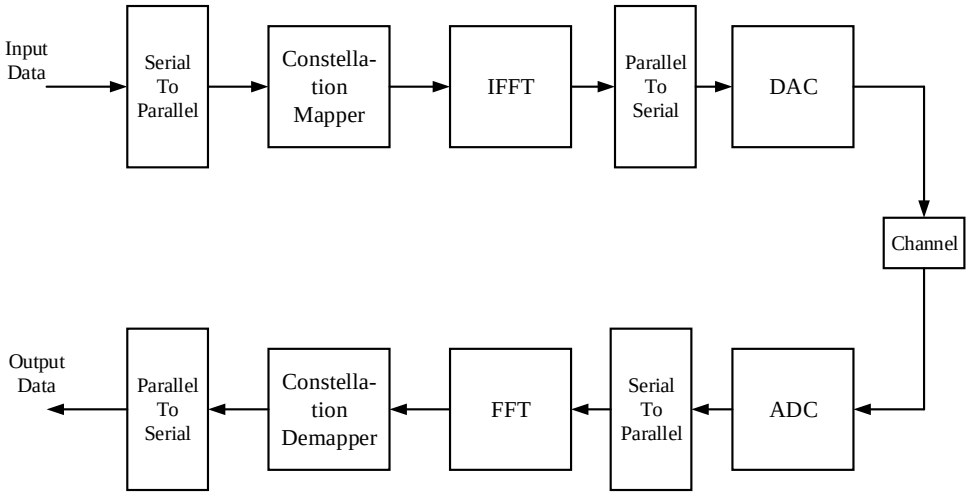

Figure 3.17: Block diagram of FFT based OFDM system .

(DAC). The complete OFDM symbol is transmitted through the channel. On receiver side this OFDM symbol is converted back to parallel stream and converted to digital signal using Analog to Digital Converter (ADC). An FFT is used to decode the OFDM subcarriers and then mapped to constellation demapper. Finally, the received signal is serialized to get the output data. The implementation of FFT and IFFT blocks in OFDM systems require more hardware and also consumes more power. Therefore, there is a need to implement FFT/IFFT block with less hardware and consumes low power.

An FFT processor for OFDM systems (IEEE 802.11a and IEEE 802.15.4 $-$ g) can be implemented by using either memory based architecture or pipeline architecture. The memory based architecture is the one that requires less hardware resources; there is a lot of resource sharing, and the data must be carefully directed to the correct

Table 3.2: Comparison of the number of complex multipliers, adders, and memory units for various pipelined architectures

Architecture	Complex Multipliers	Complex Adders	Memory
R2SDF	$2(\log_4 N - 1)$	$4\log_4 N$	$N-1$
R4SDF	$\log_4 N - 1$	$8\log_4 N$	$N-1$
R2^{2}SDF	$\log_4 N - 1$	$4\log_4 N$	$N-1$
R2^{3}SDF	$\log_4 N - 1$	$4\log_4 N$	$N-1$
R4SDC	$\log_4 N - 1$	$3\log_4 N$	$2N-2$
R2MDC	$2(\log_4 N - 1)$	$4\log_4 N$	$3N/2-1$
R4MDC	$3(\log_4 N - 1)$	$8\log_4 N$	$5N/2-1$
R2^{3}MDC	$2(\log_4 N - 1)$	$2\log_4 N$	$3N/2-1$
SRMDC	$4(\log_4 N - 1)$	$12\log_4 N - 8$	$3N/2-1$

functional units. It also require higher clock frequency to perform all the required butterfly operations along with complex multiplications which consumes more power. Therefore, pipeline architectures can process the FFT at the sampling rate, and exhibit low power consumption.

Table 3.2 tabulates the resource requirements of each of the pipelined FFT architectures discussed. It includes information on complex multipliers, complex adders, and memory requirements. The complex multipliers perform twiddle factor multiplications which require more hardware to implement than complex adders. Therefore, various possibilities to implement complex multipliers will be discussed in the next chapter 4. From Table 3.2, one can observe that, due to the efficient use of shift register elements, SDF architecture has the minimum memory requirement of $(N-1)$ words. The MDC architecture can provide higher throughput than SDF pipelined architecture, as it can compute several samples in parallel at the cost of hardware.

Based on this information, the SDF architectures are better because of the reduced hardware requirements for IEEE 802.11a and IEEE 802.15.4 $-$ g standard OFDM systems.

3.4 Conclusion

The most common FFT architectures are explained in this chapter, which includes memory based and pipelined architectures. The memory based FFT architecture requires the least amount of hardware resources. However, the memory requirement is more and also consumes more power. Furthermore, two popular styles of pipelined hardware architecture for FFT implementation are discussed in this chapter, which comprises of delay commutator and delay feedback architectures. These architectures have their own merits and demerits based on different approaches.

The delay feedback architecture is always efficient than their corresponding Commutator architecture in terms of the memory requirement and the utilization, since the butterfly utilizes the same delay element for the storage of the incoming and outgoing samples. Therefore, delay feedback architectures are considered an optimal choice in terms of the hardware cost and performance for many applications. The butterfly structure of the radix-2^i architectures is simpler than the radix-2 butterflies.

Therefore, the radix-2^i processing elements are better utilized. Also the radix-2^i single path architectures utilize the multiplier in efficient manner as compared to their corresponding radix-2 single path architectures.

In chapter 2, it is discussed that radix-2^i algorithms are the best option with respect to the number of operations. In this chapter, it is shown that SDF architectures are better choice in terms of resources required and its utilization. From these observations, the R2^iSDF can be concluded as the ideal choice for OFDM systems (IEEE 802.11a and IEEE 802.15.4 $-$ g).

Chapter 4

Twiddle Factor Multiplication and its Hardware

The twiddle factor multiplication can be implemented by different techniques, regardless of which FFT algorithm and architecture is chosen. These techniques include general complex multiplication and complex constant multiplication. To implement complex multiplier in FFT hardware, fixed-width multiplier is required in order to prevent the result from growing in size after every multiplication. Based on the trade-off between the hardware cost and the accuracy of the output signals, all the input signals and twiddle factor coefficients have to be considered fixed-width of data. The complex constant multipliers are multiplier-less implementations, where shifters and adders are used to implement the twiddle factor multiplication.

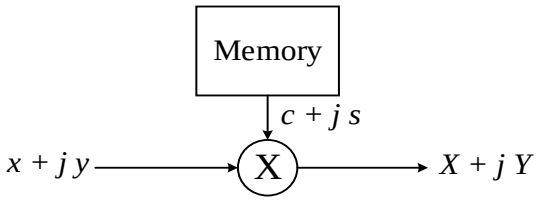

Figure 4.1: General complex multiplier.

In this chapter, Section 4.1 presents the implementation of general complex multiplier with different approaches. Further, we discuss various fixed-width multiplier designs in the literature. In Section 4.2, we present two types of constant multiplications that are single and multiple constant multiplications.

4.1 General Complex Multiplication

The block diagram representation of a general complex multiplication is shown in Figure 4.1. This consists of a memory to store the real and imaginary parts of the twiddle factor coefficients and a complex multiplier. The twiddle factor multiplication of a complex input $(x + jy)$ and twiddle factor $(W_N = c + js)$ can be calculated as:

$$\begin{aligned}(x+jy)\cdot W_N &= (x+jy)(c+js)\\ &= x\cdot c + j\cdot x\cdot s + j\cdot y\cdot c + j^2\cdot y\cdot s\\ &= (x\cdot c - y\cdot s) + j(x\cdot s + y\cdot c)\\ &= X + j\cdot Y \end{aligned} \tag{4.1}$$

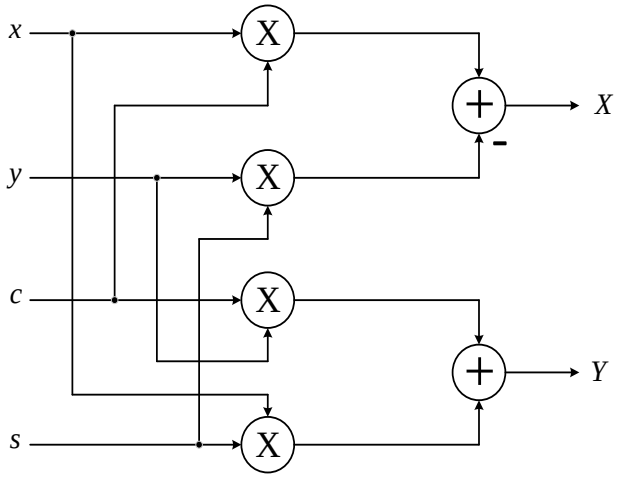

Figure 4.2: Approach I with four multipliers and two adders.

where X and Y are the real and imaginary parts of the result.

Approach I: The common approach of implementing equation (4.1) is by using four real multipliers and two real adders. The direct implementation of the complex twiddle factor multiplication is shown in Figure 4.2. However, this approach require more hardware due to 4 real multipliers.

Approach II: The twiddle factor multiplication in equation (4.1) can be rewritten in the following approach as:

$$\begin{aligned}(x+jy)(c+js) &= [x\cdot(c+s)-(x+y)\cdot s]+j\ [x\cdot(c+s)+(y-x)\cdot s] \\ &= X+j\cdot Y \end{aligned} \tag{4.2}$$

This approach requires three real multipliers instead of four as in Approach I, but the number of real adders are increased to five. The structure of equation (4.2) is shown

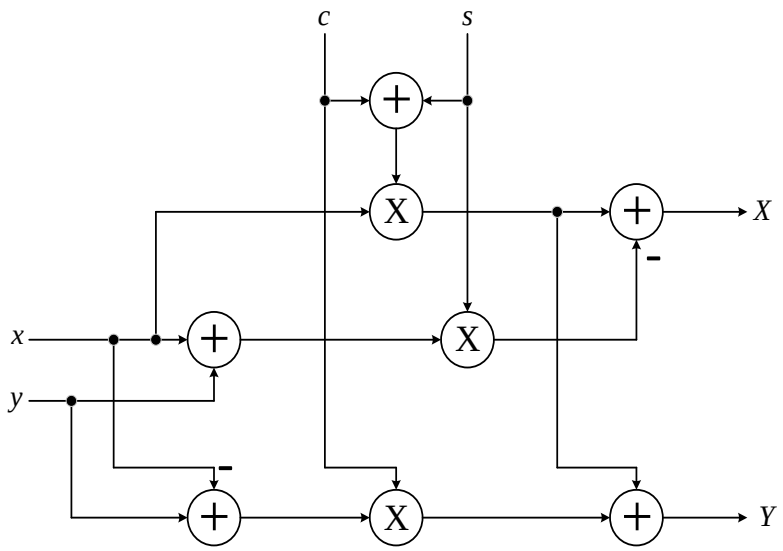

Figure 4.3: Approach II with three multipliers and five adders.

in Figure 4.3.

Approach III: Another approach to rewrite the twiddle factor multiplication (4.1):

$$\begin{aligned}(x+jy)(c+js) &= [(x+y)\cdot c - x\cdot(c+s)] + j\,[(x+y)\cdot c + y\cdot(s-c)] \\ &= X + j\cdot Y \end{aligned} \tag{4.3}$$

Based on equation (4.3), the implementation structure is depicted in Figure 4.4. It requires three multipliers and three adders, therefore this is called as a 3/3 algorithm. However, it needs to store $c+s$, $c-s$ and c coefficients in memory. In most of the FFT applications this approach is considered as this reduces the hardware complexity of twiddle factor multiplication.

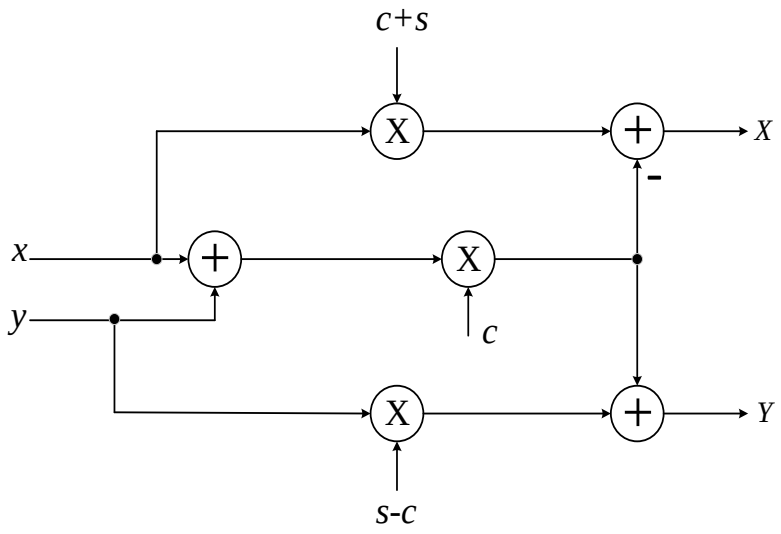

Figure 4.4: Approach III with three multipliers and three adders.

4.1.1 Fixed-width Multiplication

Multipliers play a key role in designing the FFT architectures, as it takes more hardware resources to implement. In literature there are various types of multipliers that include Baugh-Wooley multiplier, Wallace tree multiplier, Booth's multiplier, Modified Booth's multiplier, Vedic multiplier, serial multiplier and serial-parallel multiplier [86, 87, 88, 89, 90, 91, 92]. In general, the multiplication operation of two operands doubles the result when compared with the input bits. In pipelined FFT architectures as discussed in chapter 3, each stage performs twiddle factor multiplication that results with increasing the number of bits at each stage. Therefore, the output of multiplication result need to be truncated or rounded to lower the number of bits. The selection of the number of bits mainly depends on the hardware cost and the accuracy of the output signal.

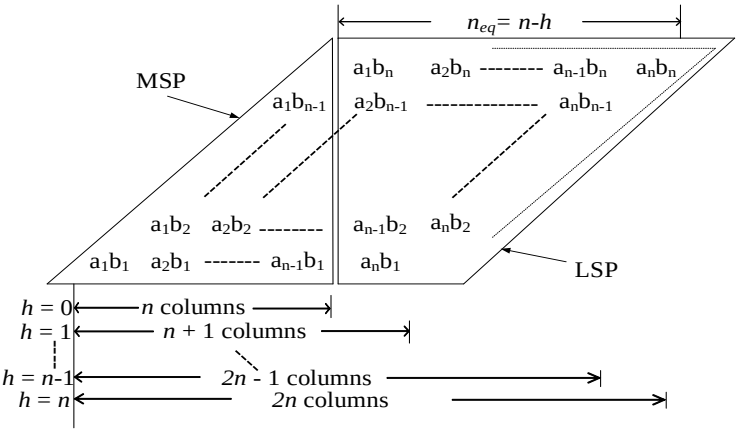

Figure 4.5: Partial product array for $n \times n$ unsigned multiplication.

By directly truncating n-bit Least Significant Part (LSP) partial product array of a full width multiplier, a fixed-width multiplier can be obtained. However, directly truncating the Least Significant Part (LSP) partial product array of the multiplication product leads to truncation error [93]. In order to mitigate this truncation error, many error compensation methods have been proposed in the literature. In [94, 95, 96], simplest techniques are used by compensating the truncation errors with a fixed bias. The accuracy of truncated multiplier is significantly improved in [97, 98, 99, 100, 101, 102, 103, 104, 105, 106, 107], where an error compensation circuit is obtained by adaptive correction method to estimate sum of the LSP terms. In [97, 98], Swartzlander *et al.* proposed a variable correction method for fixed-width multiplication as shown in Figure 4.5. In this multiplier, $n_{eq} = n - h$ where n represents the number of multiplier bits and h is the design parameter that range from 0 to n. This multiplier sums partial product bits by keeping $n + h$ most significant

columns of the partial products to obtain the final product for fixed-width multiplications. There are also different error compensation methods are available in literature [89, 108, 109, 110, 111, 112, 113, 114] for fixed-width multipliers to optimize error performance. The designer has to choose the trade-off between the accuracy and area of the fixed-width multiplier. However, it is quite difficult to conclude that a particular method is suitable for FFT application. The selection can be made based on circuit complexity, speed, and error performance of the fixed-width multiplier.

In [112], the authors have presented a method for compensating the truncation error of fixed-width booth multipliers in contrast with the 128-point FFT. Most of the existing research [115, 116] is using complex Booth multipliers for the twiddle factor multiplication. However, if the twiddle factor has a small number of coefficients, then the complex constant multiplier can be useful for the twiddle factor multiplications. The complex constant multiplier requires less area compared to complex Booth multiplier.

4.2 Constant Multiplication

A constant multiplication can be used when one of the operands is already known, for example the twiddle factor coefficients are already known in advance before the twiddle factor multiplication in the FFT architectures. The implementation of constant

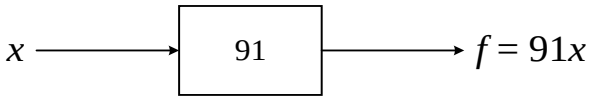

Figure 4.6: General constant multiplier.

multiplication can be done by using shifters, adders and subtractors, instead of using general multipliers. Therefore, it is called as a multiplier-less implementation.

Adders and subtractors have the same arithmetic complexity in the implementation of complex multiplier. Therefore, the term adder can be used for both adders and subtractors in the entire circuit implementation. As shifters does not have any effect on hardware cost, the numbers of adders have to be reduced for an optimal implementation of constant multiplication.

4.2.1 Single Constant Multiplication (SCM)

Figure 4.6 shows the constant multiplier block diagram for the multiplication of a variable x by a known constant number (91). The straight forward single constant multiplication [117] implementation is by translating 1's binary representation of the constant 91 into shifts and adds up the shifted inputs as follows:

$$\begin{aligned} f = 91x &= 01011011_2\ x \\ &= (x << 6) + (x << 4) + (x << 3) + (x << 1) + x \end{aligned} \tag{4.4}$$

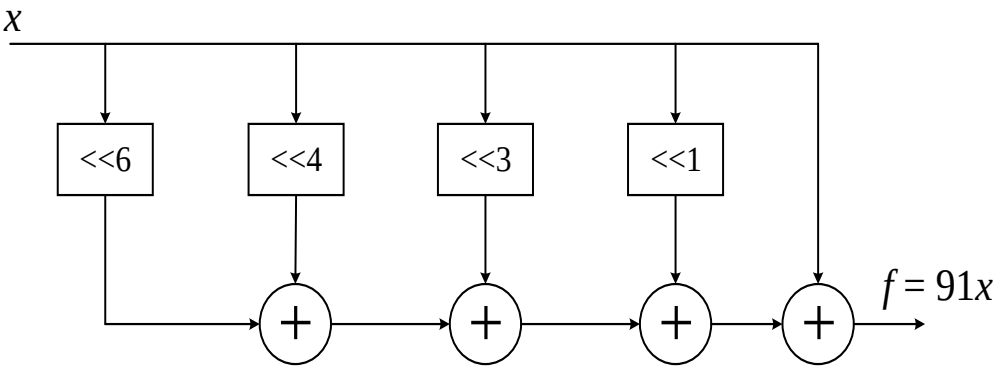

Figure 4.7: Constant multiplier using shifters and adders.

The implementation of above equation (4.4) require four adders as shown in Figure 4.7. However, this direct implementation require more adders which can be reduced by using canonical signed digit (CSD) technique [118].

CSD can reduce the number of non-zero bits in the constant value. It is a useful technique that can reduce the area of hardware implementation of constant multiplier. It includes some properties as follows:

- Represents the constant number in the form of $0, 1$ and $\bar{1}$ (-1).
- The number of non-zero bits are minimal and unique.
- No two consecutive numbers are non-zero.

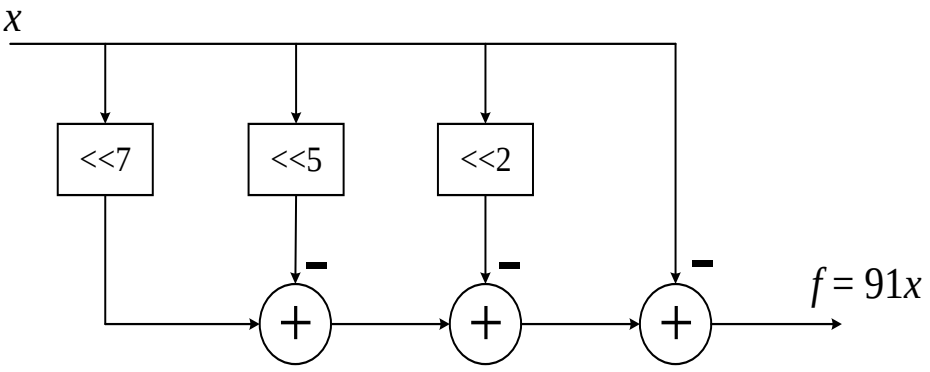

Figure 4.8: Optimized constant multiplier.

Using CSD, the previous example in equation (4.4) can be represented as follows:

$$\begin{aligned} f = 91x &= 01011011_2\ x \\ &= 10\bar{1}00\bar{1}0\bar{1}_2\ x \\ &= (x << 7) - (x << 5) - (x << 2) - x \end{aligned} \tag{4.5}$$

Based on equation (4.5), the structure is shown in Figure 4.8. This technique reduces the numbers of adders from four to three compared to straight forward implementation. This reductions helps to reduce the hardware and also power consumption of the multiplier design.

4.2.2 Multiple Constant Multiplication (MCM)

A variable can be multiplied by a given set of fixed-point constants using a multiplier block that consists exclusively of additions, subtractions, and shifts. The generation of a multiplier block from the set of constants is known as the multiple constant multiplication. In equation (4.1), c and s are the known twiddle factor coefficients in the FFT architecture. It can be seen that these constants are multiplied by unknown values x and y. In [119], the authors have proposed an algorithm for obtaining MCM with two constants. A general block diagram of the complex multiplier using MCM is shown in Figure 4.9. This contain two blocks, where constant value of c and s is implemented by shifters and adders. To obtain twiddle factor multiplication two more

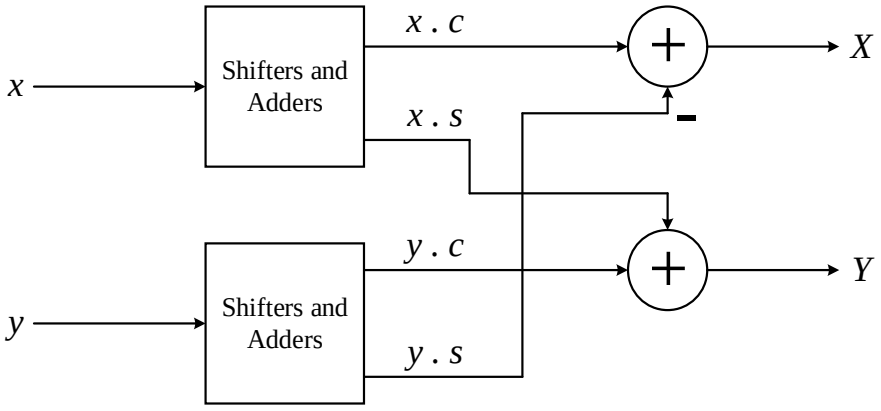

Figure 4.9: Constant multiplier using MCM.

additional adders are used. The complexity of the twiddle factor multiplication based on MCM depends on the accuracy requirement of a particular application. The CSD representation can also be used in the MCM blocks to reduce the hardware complexity of the implementation. In [26, 120] twiddle factor multiplier for $\{W_8, W_{16}$, and $W_{32}\}$ using constant multiplication were proposed. The hardware implementation of these constant multipliers require less area compared to complex Booth multipliers.

4.3 Conclusion

This chapter briefly presented different possibilities to implement the twiddle factor multiplication. These include general complex multiplication and constant multiplication. It described that the general complex multiplication can be realized by different approaches. Among these, a 3/3 algorithm (Approach III) is used in most of the FFT architectures with less number of real multipliers. Further, we have discussed about various types of fixed-width multipliers that are in the literature.

Finally, in this chapter we have presented constant multiplication that uses shifters and adders. That includes single constant multiplication and multiple constant multiplication. In the next chapters 5 and 6, the pipeline FFT architectures are proposed, which uses an optimized MCM approach using sharing mechanism.

Chapter 5

Pipeline FFT Architecture Design for an OFDM-based IEEE 802.11a

In this chapter, an area-efficient and low-power 16-bit word-width 64-point radix-2^2 and radix-2^3 pipelined SDF FFT architectures for an OFDM-based IEEE 802.11a wireless LAN baseband are presented. Based on the analysis discussed in chapter 2 and chapter 3, the designs are derived from radix-2^i algorithm and adopts a SDF architecture for hardware implementation. To eliminate the complex multipliers and memory in twiddle factor multiplication, the proposed 64-point FFT employs a CSD complex constant multiplier (CCM) using adders, and shifters.

Rest of this chapter is organized as follows. In Section 5.1, the design consideration for 64-point FFT is presented. Section 5.2 presents the pipelined radix-2^i SDF FFT

architectures with modified CSD complex constant multipliers. The comparison and performance evaluation of various FFT architectures are then discussed in Section 5.3. Conclusions are given in Section 5.4.

5.1 Design consideration of the FFT for 64-point

Now-a-days, the demand for wireless devices are increasing rapidly that requires less hardware and low power FFT architecture. According to IEEE 802.11a standard, the required sampling frequency (f_s) is 20 MHz and the total number of subcarriers is 64, which determines the FFT size [121]. The FFT has to be computed within 3.2 μs, which is the multiplication of the inverse of sampling frequency with FFT size ($T_{FFT} = 64\frac{1}{f_s}$). To meet these design constraints, pipelined architectures are suitable as these can take less hardware and consumes low power.

In general, the FFT computation require complex multipliers in order to multiply the twiddle factor with input signals and also read-only memory (ROM) to store the required twiddle factors, that takes large area and power consuming. Chu Yu et.al. [108] proposed a ROM-less pipelined FFT to reduce the hardware of the design. However, the design uses a reconfigurable complex multiplier that takes more area. In [121], the authors have presented a ROM-less and multiplier-less FFT using shifters and adders. As the proposed architecture is targeted at the IEEE 802.11a based OFDM system, the main design challenge is to reduce the complexity of multipliers.

Table 5.1: Base number of twiddle factor at each stage to compute 64-point FFT

Algorithm \ Stages	1	2	3	4	5
Radix-2	W_{64}	W_{32}	W_{16}	W_8	$-j$
Radix-2^2 [24]	$-j$	W_{64}	$-j$	W_{16}	$-j$
Radix-2^3 [122]	$-j$	W_8	W_{64}	$-j$	W_8
Radix-2^4 [123]	$-j$	W_{16}	$-j$	W_{64}	$-j$

5.2 Proposed Modified FFT Architectures

An 64-point FFT computation with radix-2^i algorithm is composed of 6 stages. The radix-2^i algorithm retains the structure of radix-2 algorithm and it has the same butterfly structure regardless of i. However, the twiddle factor multiplication position is different for different values of $i = 2, 3$ and 4. Table 5.1 describes the sequence of 64-point FFT twiddle factor computation at each stage for radix-2 and radix-2^i FFT algorithms.

Figure 5.1 shows the architectures of the Radix-2^i SDF (R2^{i}SDF) pipelined FFT, where $k = 2, 3$ and 4, for $N = 64$. The implementation uses two types of butterfly units ($BU1$ and $BU2$), several delay buffers of various lengths and multiplexers for data shuffling to get proper data at the butterfly input. The symbol $\odot$ represents the modified CSD complex constant multipliers (CCM1, CCM2 and CCM3). A 6-bit counter is used to switch the butterfly units between different modes and also to provide a proper twiddle factor multiplication, which is not shown in Figure 5.1. In

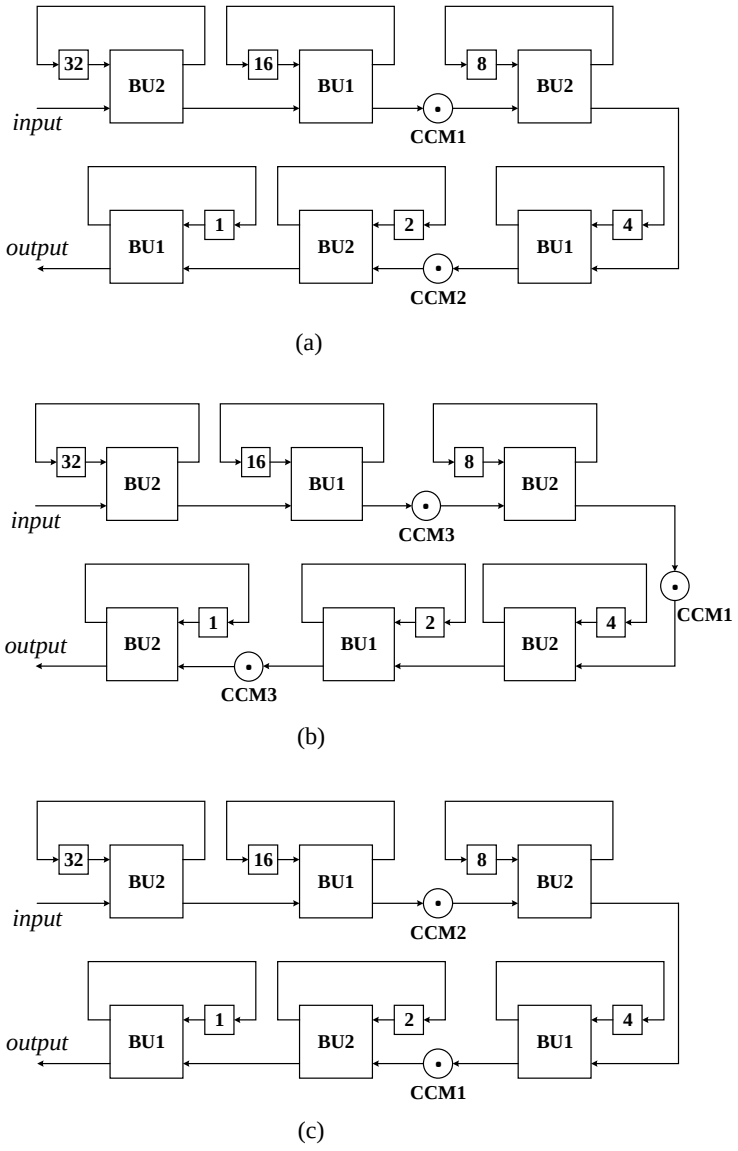

Figure 5.1: Radix-2^i pipelined SDF 64-point FFT : (a) R2^2SDF , (b) R2^3SDF, and (c) R2^4SDF

the next sections, the functions of the butterfly units and the modified CSD complex constant multipliers are explained in detail.

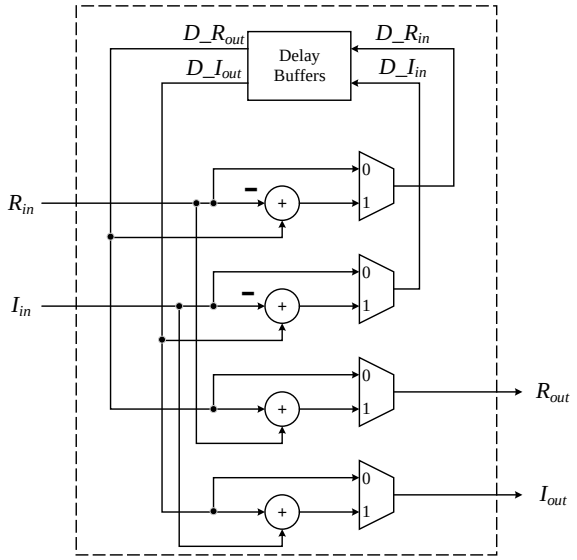

Figure 5.2: Butterfly units BU1

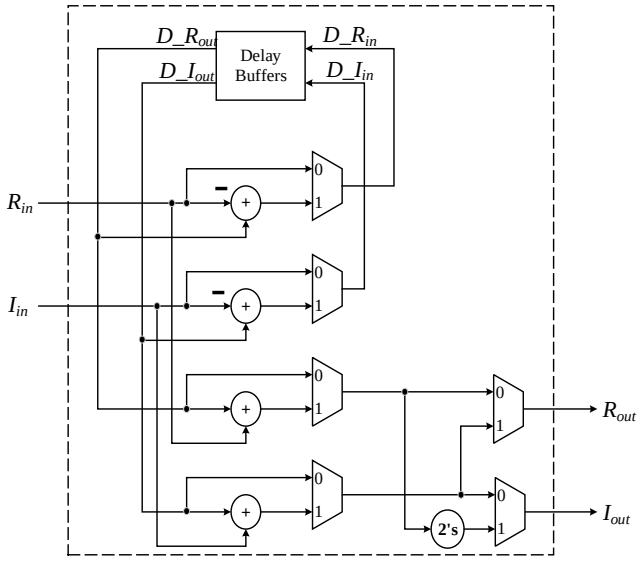

Figure 5.3: Butterfly units BU2

5.2.1 Butterfly unit

In the pipelined architectures of Figure 5.1, two types of butterfly units (BU1 and BU2) [124] are used as illustrated in Figure 5.2, and Figure 5.3, respectively. Butterfly

unit performs complex addition and complex subtraction with the delayed input. In Figure 5.2, R_{in} and I_{in} are the real and imaginary input data, R_{out} and I_{out} are the real and imaginary output data after processing the input data. Similarly D_R_{in}, and D_I_{in}, are real and imaginary inputs to delay buffers, D_R_{out}, and D_I_{out} are real and imaginary outputs of delay buffers. At each stage of BU2 unit, it has to compute the multiplication by $-j$ and 1. Therefore, multiplication by $-j$ is obtained by just swapping the real and imaginary parts and changing the sign by using 2's complement as shown in Figure 5.3.

5.2.2 Modified CSD Complex Constant Multipliers

In general, the output data values from the butterfly unit need to be multiplied by the non-trivial twiddle factors. The twiddle factors $W_N^q = e^{-j2\Pi q/N} = X_q + jY_q$, where $q = 0$ to $N-1$ are divided into eight types. Here, X_q and Y_q are the real and imaginary parts of the twiddle factors. By utilizing the symmetry property of twiddle factor only $(N/8)$ sets of constant values, i.e., $W_N^{q'} = X_{q'} - jY_{q'}$, where q' is from 0 to $(N/8)$ in *Type1* are needed for the twiddle factor multiplication.

The *Type1* operation is represented as follows:

$$Type1 : (R_{in} + jI_{in})W_N^q = (R_{in} + jI_{in})(X_{q'} - jY_{q'}), 0 < q \leq N/8 \qquad (5.1)$$

The twiddle factors of other types can be obtained by mapping q to a number $q^{'}$ in *Type1*, as follows:

$$Type2 : (R_{in} + jI_{in})W_N^q = (R_{in} + jI_{in})(Y_{q'} - jX_{q'}),$$
$$N/8 < q < N/4 \tag{5.2}$$

$$Type3 : (R_{in} + jI_{in})W_N^q = (R_{in} + jI_{in})(-Y_{q'} - jX_{q'}),$$
$$N/4 < q \leq 3N/8 \tag{5.3}$$

$$Type4 : (R_{in} + jI_{in})W_N^q = (R_{in} + jI_{in})(-Xq^{'} - jY_{q'}),$$
$$3N/8 < q < N/2 \tag{5.4}$$

$$Type5 : (R_{in} + jI_{in})W_N^q = (R_{in} + jI_{in})(-Xq^{'} + jY_{q'}),$$
$$N/2 < q \leq 5N/8 \tag{5.5}$$

$$Type6 : (R_{in} + jI_{in})W_N^q = (R_{in} + jI_{in})(-Y_{q'} + jX_{q'}),$$
$$5N/8 < q < 3N/4 \tag{5.6}$$

$$Type7 : (R_{in} + jI_{in})W_N^q = (R_{in} + jI_{in})(Y_{q'} + jX_{q'}),$$
$$3N/4 < q \leq 7N/8 \tag{5.7}$$

$$Type8 : (R_{in} + jI_{in})W_N^q = (R_{in} + jI_{in})(X_{q'} + jY_{q'}),$$
$$7N/8 < q < N \tag{5.8}$$

Based on the eight operation types (*Type1-Type8*), the complex multiplications for N-point FFT will be reduced to the computation of $(N/8)$ sets of constant values.

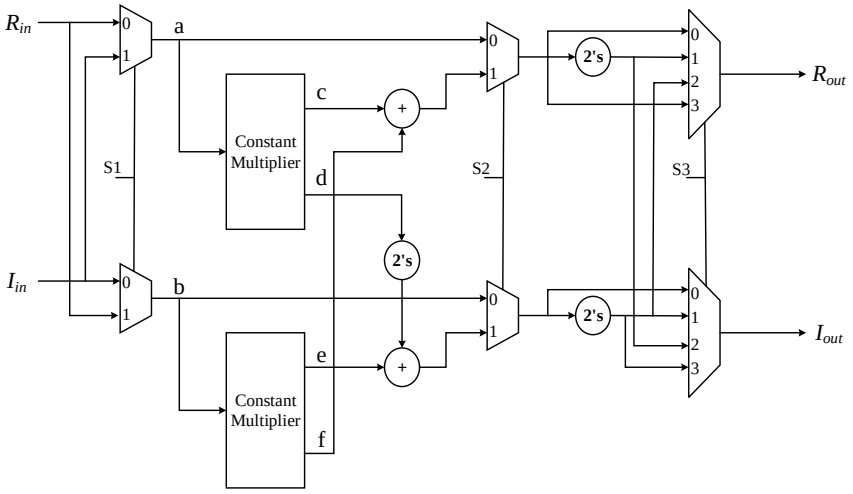

Figure 5.4: Proposed modified CSD multiplier for CCM1 and CCM2

This reduces the number of twiddle factors to compute the FFT. In [108], the authors have designed a reconfigurable complex constant multipliers that reduces the size of ROM using these $(N/8)$ sets of constant values for twiddle factor multiplication. However, this design uses complex multiplier that takes more area. Some simplification scheme has been presented in [121], by using CSD based constant multipliers instead of complex multipliers. However, this design requires more adders to design complex constant multiplier. Therefore, we employ CSD representation and common sub-expression sharing methods for complex constant multiplier in order to reduce the area cost.

Figure 5.4 shows the proposed modified CSD complex constant multiplier. Here, a and b are the inputs to the constant multiplier that are obtained based on the select signal S1. The outputs of the constant multiplier units are c, d, e and f. These output are the product of the complex input and the twiddle factor constants of the constant

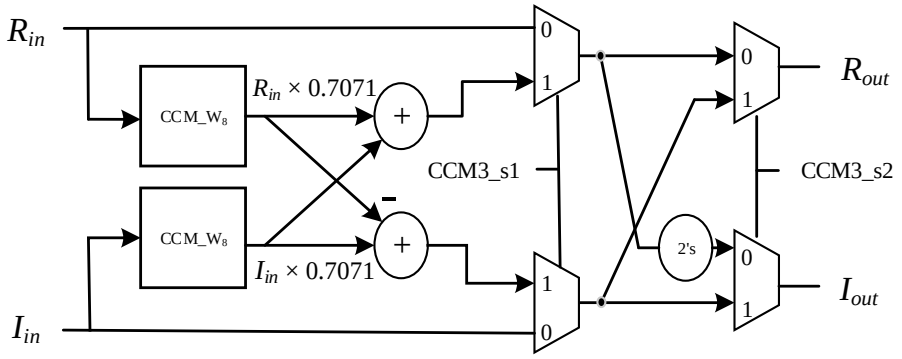

Figure 5.5: CSD complex constant multiplier for CCM3

multiplier units. The final complex multiplication output is obtained by the selection of the signals S2 and S3.

For a 64-point FFT, various complex constant multipliers (CCM1, CCM2 and CCM3) are required as shown in Figure 5.1. Here, the non-trivial type of operations for CCM1 structure can be obtain by using CCM_W_{64} multiplier unit of Figure 5.6 as a constant multiplier and similarly CCM2 can be obtain by using CCM_W_{16} of Figure 5.7 multiplier unit as a constant multiplier in Figure 5.4. The trivial type of operations can be obtain without selecting the CCM_W_{64} or CCM_W_{16} multiplier units in Figure 5.4. The CSD complex constant multiplier for CCM3 is shown in Figure 5.5, it can be designed by less hardware compared to CCM1 and CCM2. The detailed description of these constant multipliers is described in the next subsections.

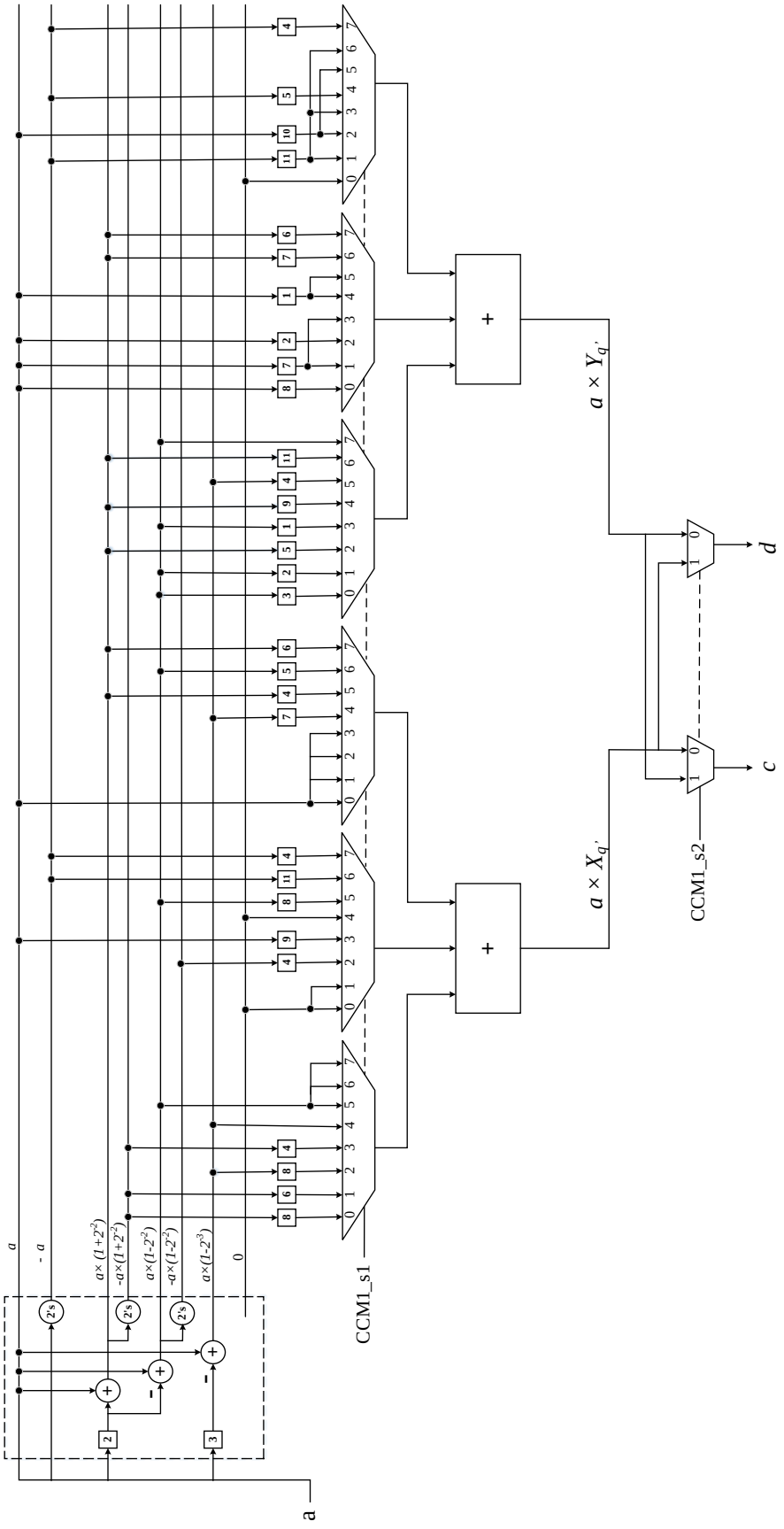

Figure 5.6: Block diagram of CCM_W_{64} multiplier unit

5.2.2.1 CCM_W_{64} multiplier unit

In [121] for 64-point FFT, it is stated that 49 non-trivial constants (W_{64}^{q}) are to be multiplied to the intermediate results. As the twiddle factors have 1/8 symmetric property, only eight sets of non-trivial constant values are required for the multiplication operation. They are (0.9952, 0.0980), (0.9808, 0.1951), (0.9569, 0.2903), (0.9239, 0.3826), (0.8819, 0.4714), (0.8315, 0.5556), (0.7730, 0.6344), (0.7071, 0.7071), where, in each set, the first entry corresponds to real part ($X_{q'}$) and the second one corresponds to the imaginary part ($Y_{q'}$) in the expansion of W_{64}^{q}.

The eight sets of constants can be represented in CSD form as shown in Table 5.2. Furthermore, hardware-less realization of CCM_W_{64} constant multiplier can be achieved by utilizing the common sub-expression sharing of the eight sets of constants. In Table 5.2, one of the term "-10-1"as enclosed by black ellipses is used as one of the common sub-expression. The remaining seven terms "0", "1", "-1", "101", "10-1", "-101", and "100-1" are also used to realize the common sub-expression sharing block. Exploring the advantage of presence of such common sub-expressions, a hardware efficient CCM_W_{64} multiplier structure is given in Figure 5.6. The sharing block consisting of shifters, adders and two's complement are shown in Figure 5.6 within dashed line box. The square boxes represents right shifted values of the input given to it. These shifters are realized using simple hardware connections. For one among the eight twiddle factor constant multiplications, six 8-to-1 multiplexers are used to

Table 5.2: CSD representations of eight sets constant values for composing twiddle factors

Set	Constant value of the real part	CSD representation of real part											Constant value of the imaginary	CSD representation of imaginary part										
		2^0	2^{-1}	2^{-2}	2^{-3}	2^{-4}	2^{-5}	2^{-6}	2^{-7}	2^{-8}	2^{-9}	2^{-10}	2^{-11}		2^0	2^{-1}	2^{-2}	2^{-3}	2^{-4}	2^{-5}	2^{-6}	2^{-7}	2^{-8}	2^{-9}	2^{-10}	2^{-11}
1	0.9952	1	0	0	0	0	0	0	0	-1	0	-1	0	0.0980	0	0	0	1	0	-1	0	0	1	0	0	0
2	0.9808	1	0	0	0	0	0	-1	0	-1	0	0	0	0.1951	0	0	1	0	-1	0	0	1	0	0	0	-1
3	0.9569	1	0	0	0	-1	0	1	0	1	0	0	-1	0.2903	0	0	1	0	0	1	0	1	0	0	1	0
4	0.9239	1	0	0	0	-1	0	-1	0	0	1	0	0	0.3826	0	1	0	-1	0	0	0	1	0	0	0	0
5	0.8819	1	0	0	-1	0	0	0	1	0	0	-1	0	0.4714	0	1	0	0	0	-1	0	0	0	1	0	1
6	0.8315	1	0	-1	0	1	0	1	0	1	0	-1	0	0.5556	0	1	0	0	1	0	0	-1	0	0	1	0
7	0.7730	1	0	-1	0	0	1	0	-1	0	0	0	1	0.6344	0	1	0	1	0	0	0	1	0	1	0	-1
8	0.7071	1	0	-1	0	-1	0	1	0	1	0	0	0	0.7071	1	0	-1	0	-1	0	1	0	1	0	0	0

Table 5.3: Selection of the Twiddle Factors in CCM1

Twiddle Factors / Control Signals	1	W_{64}^{1}-W_{64}^{8}	W_{64}^{9}-W_{64}^{15}	$-j$	W_{64}^{17}-W_{64}^{24}	W_{64}^{25}-W_{64}^{31}	-1	W_{64}^{33}-W_{64}^{40}	W_{64}^{41}-W_{64}^{47}
S1	0	0	0	1	1	1	0	0	0
S2	0	1	1	0	1	1	0	1	1
S3	0	0	0	3	2	2	1	1	1
CCM1_s1	x	0-7	7-1	x	0-7	7-0	x	0-7	7-0
CCM1_s2	x	0	1	x	1	0	x	0	1

get the appropriate result.

The multiplication operations of the complex inputs and the twiddle factors are conducted using five control signals as given in Table 5.3. The S1, S2 and S3 are the control signals in Figure 5.4 and CCM1_s1, CCM1_s2 are the control signals of the CCM_W_{64} multiplier unit in Figure 5.6, which is used as a constant multiplier in Figure 5.4. Here, 'x' denotes the don't care value. The trivial type of operations $(1, -j, -1)$ can be obtained by selecting only three control signals S1, S2 and S3. The non-trivial type of operations can be obtained by proper selecting the five control signals as shown in Table 5.3.

The procedure of complex multiplication can be explained as follows. If the complex input $(R_{in} + jI_{in})$ is to be multiplied with W_{64}^{1} (*Type1*) constants 0.9952 and 0.0980 (real and imaginary part of the first set of non-trivial constant). The output

of the CCM1 module is

$$R_{out} = R_{in} \times 0.9952 + I_{in} \times 0.0980 \tag{5.9}$$

$$I_{out} = -R_{in} \times 0.0980 + I_{in} \times 0.9952 \tag{5.10}$$

In order to obtain the complex multiplication output, the control signal S1 is selected as 0 that passes the R_{in} and I_{in} to a and b respectively. The constants 0.9952 and 0.0980 are decomposed in terms of power of 2 as $(1 - 2^{-8} - 2^{-10})$ and $(2^{-3} - 2^{-5} + 2^{-8})$ respectively. The outputs of the constant multiplier units are $a \times 0.9952$, $a \times 0.0980$, $b \times 0.9952$ and $b \times 0.0980$ represented as c, d, e and f respectively. These outputs are obtained by selecting the constant multiplier control signals CCM1_s1 and CCM1_s2 as 0 and 0 respectively. The final real and imaginary outputs of the CCM1 module are obtained as shown in equations 5.9 and 5.10, by selecting the control signals S2 and S3 as 1 and 0 respectively. Thus with the CSD representation and common sub-expression sharing block, the multiplication of $(R_{in} + jI_{in})$ with these constants effectively turns into additions or subtractions of a series of right shifted values.

5.2.2.2 CCM_W_{16} multiplier unit

In the similar way of utilizing symmetric property of complex sinusoidal functions for W_{16}^{q}, both real and imaginary parts of twiddle factors can be derived from two sets of non-trivial constant values, (0.9239, 0.3826) and (0.7071, 0.7071). In Table 5.2, Set4 and Set8 represents the CSD form of these two sets of constants respectively. The constant multiplier operation in terms of power of 2 can be expressed as

$$out1 = a \times 0.7071 = a \times \{1 - 2^{-2} - 2^{-4} + 2^{-6} + 2^{-8}\} \tag{5.11}$$

$$out2 = a \times 0.9239 = a \times \{1 - 2^{-4} - 2^{-6} + 2^{-9}\} \tag{5.12}$$

$$out3 = a \times 0.3826 = a \times \{2^{-1} - 2^{-3} + 2^{-7}\} \tag{5.13}$$

The three multiplication operations can be obtained by simply using seven additions and six shift operations as shown in Figure 5.7. The twiddle factor selection in CCM2 with control signals are given in Table 5.4. In order to select the two sets of twiddle factor multiplications, two 2-to-1 multiplexers are used in the CCM_W_{16} multiplier unit. If CCM2_s=0, the two multiplexers selects the multiplication operation of the input 'a' and the first set of constants 0.9239 and 0.3826. If CCM2_s=1, it selects the multiplication with the second set of constants 0.7071 and 0.7071. The area cost of the CCM_W_{16} multiplier unit is much less than the area cost of CCM_W_{64}, as it can be realized using two sets of constants.

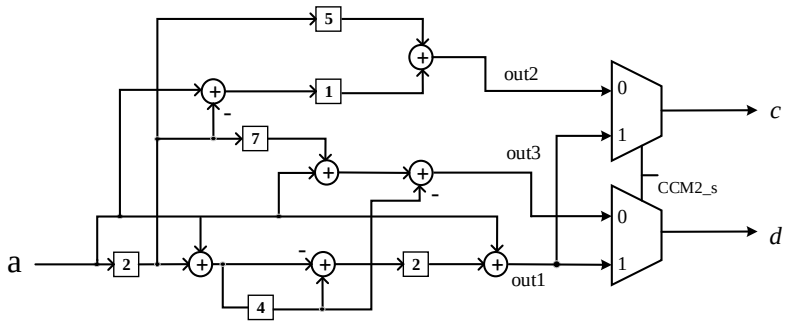

Figure 5.7: Block diagram of CCM_W_{16} multiplier unit

Table 5.4: Selection of the Twiddle Factors in CCM2

Twiddle Factors / Control Signals	1	W_{16}^{1}	W_{16}^{2}	W_{16}^{3}	$-j$	W_{16}^{6}	W_{16}^{9}
S1	0	0	0	1	1	1	0
S2	0	1	1	1	0	1	1
S3	0	0	0	0	3	1	1
CCM2_s	x	0	1	1	x	1	1

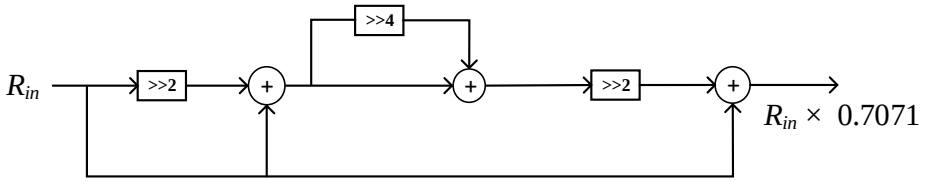

Figure 5.8: Block diagram of CCM_W_8 multiplier unit

5.2.2.3 CCM_W_8 multiplier unit

This multiplier unit requires only one set of constant value $(0.7071, 0.7071)$ for multiplication operation. Figure 5.8 shows the CCM_W_8 multiplier unit for CCM3 in Figure 5.5. In CCM3, the multiplication operations of the complex input and the twiddle factors, 1, W_8^1, $-j$ and W_8^3 can be obtained by using two control signals

Table 5.5: Selection of the Twiddle Factors in CCM3

Twiddle Factors / Control Signals	1	W_8^1	$-j$	W_8^3
CCM3_s1	0	1	0	1
CCM3_s2	0	0	1	1

$CCM3\_s1$ and $CCM3\_s2$ as shown in Table 5.5. By employing this multiplier unit, CCM3 can be designed using only eight adders and six multiplexers.

5.3 Comparison and Experimental results

Table 5.6 compares the hardware requirement and performance of the proposed R2^{i}SDF architectures to other pipelined architectures for the computation of a 64-point FFT. The hardware requirement is measured in terms of number of complex multipliers, CSD constant multipliers, complex adders and memory. The performance is represented by critical path delay, latency and throughput. The critical path delay is defined as the path in the pipelined architecture with the maximum delay. Latency is the number of clock cycles that the architecture takes to process an input sequence by considering a continuous flow of data. Finally, the throughput indicates the number of samples processed by the architecture per clock cycle.

The proposed architectures does not require any complex multipliers, as the multipliers are realized with the modified CSD complex constant multiplier unit. The

Table 5.6: Comparison of the proposed R2^{i}SDF architectures to other architectures for the computation of a 64-point FFT

Architecture	Complex Multipliers	CSD Constant Multipliers	Complex Adders	Memory	Critical path delay	Latency	Throughput
R2SDF [125]	4	-	12	63	$T_M + 2T_A + T_{MUX}$	127	R
R4SDF [125]	2	-	24	63	$T_M + 3T_A + T_{MUX}$	127	R
R4SDC [125]	2	-	9	126	$T_M + 3T_A + T_{MUX}$	127	R
R2^{2}SDF [125]	2	-	12	63	$T_M + 2T_A + T_{MUX}$	127	R
R2^{3}SDF [126]	1	2	20	63	$T_M + 2T_A + 2T_{MUX}$	127	R
R2SDF [108]	1	2	12	63	$T_M + 3T_A + 5T_{MUX} + 2T_C$	127	R
R8SDC [121]	0	3	48	171	$9T_A + 2T_{MUX}$	64	8R
Proposed R2^{2}SDF	0	2	12	63	$5T_A + 5T_{MUX} + 3T_C$	127	R
Proposed R2^{3}SDF	0	3	12	63	$5T_A + 6T_{MUX} + 3T_C$	127	R
Proposed R2^{4}SDF	0	2	12	63	$5T_A + 5T_{MUX} + 3T_C$	127	R

Table 5.7: Comparison of number of adders for CCM_W_{64} multiplier unit

CCM_W_{64} multiplier unit	Adders
[121]	54
Proposed	14

hardware complexity of the proposed architectures is reduced by adopting the multiplication units using CSD representation and common sub-expression sharing. Table 5.7 shows the comparison of the number of adders required for the proposed modified CSD complex constant multiplier with the constant multiplier in [121]. The proposed modified constant multipliers requires less adders, that reduces the total area of the FFT design.

From Table 5.6, even though the number of complex adders are less for R4SDC architecture, but the memory requirement is more and also control unit is complex.

The number of complex adders and memory requirement is more in [121]. The proposed modified architectures requires the same number of complex adders and memory compared to that of other SDF architectures with simple control.

The critical path delay [127] of the proposed designs is $5T_A + 5T_{Mux} + 3T_C$, and $5T_A + 6T_{Mux} + 3T_C$ for R2^{2}SDF and R2^{3}SDF, respectively, where T_A, T_{Mux} and T_C are computation time of an adder, multiplexer and two's complement, respectively. The proposed architecture latency and throughput is same as the other SDF architectures [108, 121, 125]. From Table 5.6, it can be observed that R2^{2}SDF and R2^{4}SDF have the same hardware complexity. Therefore, we have considered only R2^{2}SDF and R2^{3}SDF for implementation.

To have a more generalized comparison, the word length is chosen as 16-bit in the proposed implementations. Before the hardware implementation, a proper twiddle factor word length of the proposed architectures is determined by a fixed-point simulation using MATLAB. Figure 5.9 shows the Signal-to-Quantization-Noise Ratio (SQNR) evaluation versus word length of the twiddle factor inputs. It can be observed that the SQNR increases as the word length of the twiddle factor increases from 8 to 12 bit, after that increasing the word length does not improve the performance. Therefore, the twiddle factor word length of 12-bit is selected for both R2^{2}SDF and R2^{3}SDF architectures.

The functionality of the proposed FFT architectures is verified using MATLAB

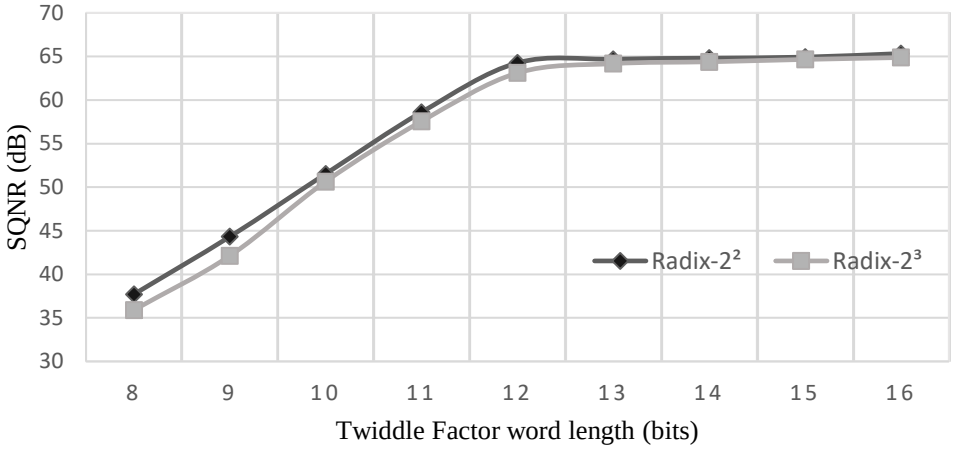

Figure 5.9: SQNR versus twiddle factor word length for R2^{2}SDF and R2^{3}SDF

Table 5.8: Comparison of various 64-point FFT architectures

Design	Word Length	Gate Counts	Technology	Power	Normalized Power
[121]	16	–	250nm,1.8 V	41 mW @20 MHz	9.88
[128]	16	–	180nm,1.8 V	68.95 mW @50 MHz	6.65
[109]	16	–	130nm,1.2 V	2.27 mW @5 MHz	4.92
[99]	In: 12, Out: 20	38,168	350nm,3.3 V	507.85 mW @150 MHz	4.85
[108]	16	33,590	180nm,1.8 V	9.79 mW @20 MHz	2.36
[129]	12	-	180nm,1.8 V	21.43 mW @50 MHz	2.06
Proposed $R2^2SDF$	16	28,880	180nm,1.8 V	7.86 mW @20 MHz	1.89
Proposed $R2^3SDF$	16	28,826	180nm,1.8 V	7.89 mW @20 MHz	1.90

and the models are converted into Verilog by HDL coder. Then the proposed models are synthesized using Synopsys Design Compiler with TSMC 180 nm, 1.8 V Standard Cell Library. Power dissipation is estimated from the synthesis netlists by feeding them into Synopsys Prime Time to perform full transistor-level power simulation.

The performance comparison of the proposed architectures with various recent 64-point FFT architectures is summarized in Table 5.8. To have meaningful comparisons among the different implementations, the gate count and normalized power, irrespective of frequency and FFT size are considered as comparison parameters. The normalized power per FFT is considered as follows [108]:

$$Normalized\ power\ per\ FFT = \frac{Power}{(Voltage)^2 \times (FFTsize) \times Frequency} \times 1000 \tag{5.14}$$

With respect to normalized power under the same process technology, the power consumption of the proposed R2^{2}SDF and R2^{3}SDF implementations is around 1.08 times lower than the design in [129]. As the proposed R2^{2}SDF and R2^{3}SDF architectures adopt modified CSD complex constant multiplier, the gate count is lesser compared to the architectures proposed in [99] and [108]. The maximum operating frequency of the proposed architectures is up to 125 MHz. The operation speed can be further improved by proper placement of pipelined registers in the design. These features show that though it has been developed primarily for application in the IEEE 802.11a standard, it can be used for any application that requires less hardware as well as low power consumption.

5.4 Conclusion

In this chapter, we have proposed R2^{2}SDF and R2^{3}SDF pipelined 64-point FFT architectures that are suitable for OFDM based IEEE 802.11a wireless systems. The proposed architectures employs radix-2^i algorithm and pipelined SDF architecture that provides lower multiplicative complexity and less hardware. The hardware cost of the architectures is further reduced as the complex multiplier is realized by using modified CSD complex constant multipliers. The implementation results show that at 20 MHz frequency the proposed R2^{2}SDF and R2^{3}SDF architectures require 28.8K total gates and dissipates only 7.86 mW and 7.89 mW respectively. As the proposed design is cost-effective with low power consumption, it can be also useful for many other OFDM based wireless systems like IEEE 802.11$b/n/ac$. However, the proposed design is suitable only for a fixed length of 64-point FFT and it is not suitable for variable-length FFTs. Therefore, in the next chapter, we propose a modified constant multiplier that can be suitable for variable-length FFTs.

Chapter 6

Variable-Length FFT Architecture for MR-OFDM

A fixed length of 64-point FFT is presented in chapter 5. In this chapter, we present a 16/32/64/128-point SDF pipeline FFT architecture targeting the Multi-Rate Multi-Regional (MR)-Orthogonal Frequency Division Multiplexing (OFDM) physical layer of IEEE 802.15.4$-g$. In the following sub-sections, Section 6.1 provides a brief description about IEEE 802.15.4$-g$ standard systems. In Section 6.2, we give an introduction to the decomposition of four different FFT lengths for $N = 128,\ 64,\ 32$, and 16 and their FFT twiddle factors. Section 6.3 presents the proposed FFT architecture for the SUN applications. The hardware requirements and performance evaluation of various

FFT architectures are discussed in Section 6.4. Finally, Section 6.5 concludes this work.

6.1 Introduction

Currently, manual or semi-manual operation is performed for remote-metering of electricity, water, gas, etc. To improve service efficiency with cost effectiveness, the utility service providers need more intelligent metering systems. Automatic Meter Reading (AMR) [130] Radio Frequency (RF) modules are combined with smart meters to eliminate the expense of human meter readers. In 2012, IEEE released an amendment to the IEEE 802.15.4 standard, which is named as IEEE $802.15.4-g$ [110]. This amendment addresses the needs of Smart Utility Networks (SUN), in the context of Low Rate (LR) Wireless Personal Area Network (WPAN). The standard is designed to achieve data rates up to 800 kbps and works in several frequency bands, from 450 MHz to 2.4 GHz. The standard proposes three Physical Layers named as Multi-Rate and Multi-Regional (MR) orthogonal frequency division multiplexing (OFDM), MR frequency shift keying (MR-FSK) and MR offset quadrature phase shift keying (MR-OQPSK).

MR-FSK [110] provides good transmitter power efficiency due to the constant envelope of the transmitted signal. It supports the data rates from 5 kbps to 400 kbps. It is a simple solution for applications requiring hardware simplicity, at the expense of compromised system performance. The MR-OQPSK [110] uses Multiplexing Direct

Sequence Spread Spectrum (MDSSS) or Direct Sequence Spread Spectrum (DSSS). This supports multiple data rates ranging from 6 kbps up to 500 kbps. Nowadays wireless systems require high data rate and robustness against frequency selective fading channels. In this regard, MR-OFDM [110] provides attractive technical advantages, at the expense of a more complicated circuit design. This can support data rate from 50 kbps up to 800 kbps. The MR-OFDM is a suitable candidate for SUN applications operating in challenging environments with severe fast fading degradation. In urban environments where obstructions and reflectors are the major cause of a lossy propagation channel, MR-OFDM can prove to be an ideal solution.

The key cores in the MR-OFDM are fast Fourier transform and inverse fast Fourier transform (IFFT). The MR-OFDM physical layer can operate in four modes, each with different FFT sizes as $128, 64, 32$ and 16, with modulation schemes binary phase shift keying (BPSK), quadrature phase shift keying (QPSK) and 16-quadrature amplitude modulation (QAM). Table 6.1 summerizes the main parameters for four operation modes. The implementation of an FFT processor is one of the most difficult parts in the realization of MR-OFDM systems as its hardware complexity is very high. Many FFT algorithms and architectures evolved in order to reduce hardware complexity with high speed.

The main area to be concentrated in FFT architectures is twiddle factor multiplication. In [131], the input stage of an N-point decimation-in-frequency (DIF) Radix-2 SDF FFT unit of an OFDM transmitter is designed based on pass logic. It

Table 6.1: MR-OFDM parameters

Parameters	Option 1	Option2	Option 3	Option 4
Sampling Rate (MSamples/s)	1.333	0.666	0.333	0.166
FFT Size	128	64	32	16
FFT Duration (μs)	96	96	96	96
Dta Rate (kbps)	100∼800	50∼800	50∼600	50∼300

is used for specific OFDM applications with BPSK and QPSK modulation. However, this design results in high hardware cost because of large Read-only memory (ROM) used to store the required twiddle factors. Therefore, to remove the ROM for area-efficient consideration, in [132] the authors proposed an efficient ROM-less 64-point FFT/IFFT processor. However, the hardware cost because of the multipliers used is high. In chapter 5, a modified complex constant multiplier using common sub-expression sharing block is proposed to reduce the hardware cost of multipliers. However, these architectures [131, 132] are not suitable for variable-length FFTs.

The main motivation of the proposed work is to design an alternative architecture for FFT computation that satisfies the standard IEEE $802.15.4-g$ with less area and low power consumption. Therefore, we propose an efficient SDF pipeline FFT with variable lengths using a hardware sharing mechanism. The important contributions of this proposed work are:

- A mixed-radix (radix-2^2 and radix-2) FFT algorithm is adopted to reduce the complexity of twiddle factor multiplications for the proposed design.

Table 6.2: Decomposition of four different FFT lengths

N	1	2	3	4
16	2^2	2^2		
32	2^2	2^2	2	
64	2^2	2^2	2^2	
128	2^2	2^2	2^2	2

- A configurable complex constant multiplier (CCCM) is proposed for twiddle factor W_{32}, W_{64} and W_{128} multiplications, as it is required for variable-length FFT.

- A hardware-sharing mechanism is employed to reduce the memory space requirement for the proposed variable-length FFT.

6.2 Decomposition and Twiddle Factors at each stage of FFT

According to IEEE 802.15.4 − g standards, four FFT lengths are required N = 128, 64, 32, and 16. Although radix-r (where r is power of 2) FFT algorithms are often adopted to reduce the FFT computation complexity, it still needs large amount of twiddle factor multiplications. Therefore, in this chapter, a hardware-efficient mixed-radix (radix-2^2 and radix-2) FFT algorithm is employed to reduce the number of complex multiplications. The decomposition of four different FFT lengths is shown in Table 6.2. Here, we fold the four FFT lengths using radix-2^2 butterflies as many

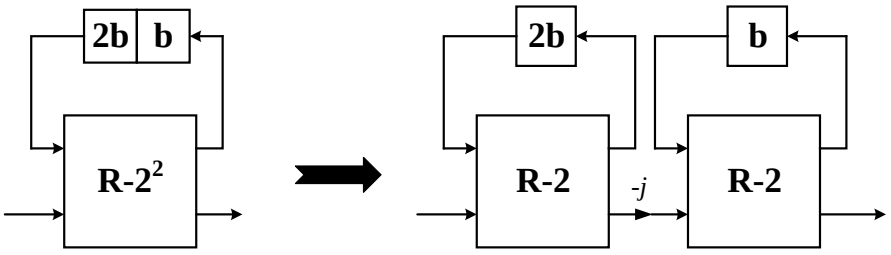

Figure 6.1: Radix-2^2 butterfly constructed from two Radix-2 butterflies

Table 6.3: Sequence of the 16/32/64/128-point FFT twiddle factor computation for mixed-radix FFT Algorithms

N	1	2	3
16	W_{16}		
32	W_{16}	W_{32}	
64	W_{16}	W_{64}	
128	W_{16}	W_{128}	W_8

times as possible. From Table 6.2, one can observe that the first two stages of the four FFT lengths can share the same hardware. The last two stages are configurable for both radix-2^2 and radix-2 computation. Therefore, the proposed design can perform variable-length FFT computation using multiplexer switches at stage 3 and stage 4. The Radix-2^2 (R-2^2) butterfly is constructed from two Radix-2 (R-2) butterflies, separated with a trivial multiplication ($-j$), as shown in Figure 6.1. Here 'b' denotes the delay buffer length. Another two sub-stages are formed for a single Radix-2^2 butterfly.

Table 6.3 shows the sequence of twiddle factors for 16/32/64/128-point FFT in Table 6.2. At stage 1 and stage 3, it requires W_{16} and W_8 multipliers respectively that can be designed by using constant multipliers. W_{128} multiplier is a non-trivial multiplier so far designed by using complex Booth multipliers. We have designed a

novel configurable constant W_{128} multiplier that can acts as W_{128}, W_{64} and W_{32} multiplier. The detailed description about configurable multiplier is discussed in section 6.3. Note that the proposed design shares W_{16} and W_{128} multipliers at stage 1 and stage 2 respectively and W_8 multiplier at stage 3 such that it obtains variable lengths of FFT. Sharing the multipliers can reduce the total hardware of the design.

6.3 Proposed Architecture

The SDF pipeline 16/32/64/128-point FFT architecture proposed for $802.15.4 - g$ standard is shown in Figure 6.2. This architecture uses the radix-2^2 and radix-2 butterflies for mixed-radix FFT computation. Here, the first four stages are four radix-2 butteries and are common for 16/32/64/128 computation. The last three stages are configured using multiplexers and switched depending on FFT computation requirement. So, the architecture has seven stages (ST1-ST7). As radix-2^2 contain two radix-2 butterfly structures, the number of stages considered seven (ST1-ST7) including the last stage with radix-2.

The proposed architecture comprises of two butterfly units (marked as BU1 and BU2 which are similar to Figures 5.2 and 5.3), several delay buffers of various lengths (indicated by numbers enclosed in rectangles), six 4-to-1 multiplexers (data shuffling), four 2-to-1 multiplexers (to select variable-length FFT) and three complex constant

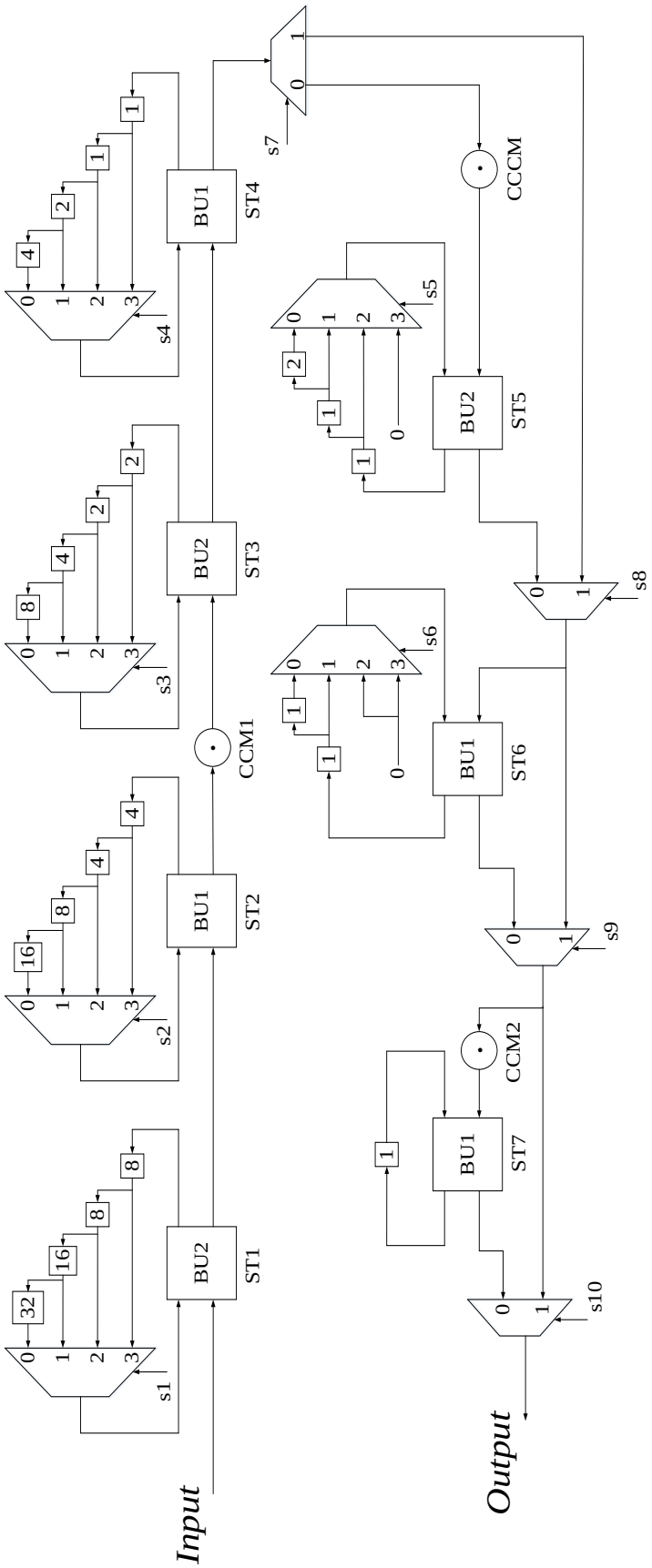

Figure 6.2: Proposed 16/32/64/128-point SDF pipeline FFT architecture

Table 6.4: Selection of variable-length FFT

N	s1	s2	s3	s4	s5	s6	s7	s8	s9	s10
16	3	3	3	3	x	x	1	1	1	1
32	2	2	2	2	2	x	0	0	1	1
64	1	1	1	1	1	1	0	0	0	1
128	0	0	0	0	0	0	0	0	0	0

multipliers (CCM1, CCM2 and CCCM using shifters and adders). In the next subsections, the description about multiplexer switchings and the design of configurable constant W_{128} multiplier are explained in detail.

6.3.1 Multiplexer switching to perform variable-length FFT

The proposed design performs 16/32/64/128-point FFT computations using the seven butterfly unit stages by means of multiplexer switching is shown in Figure 6.2. Several delay buffers of various lengths and multiplexers (MX1-MX6) with select lines s1-s6 are being used for data shuffling to get proper data at the butterfly input. The outputs of the variable lengths of FFTs are separated by one de-multiplexer (DMX7) and three multiplexers (MX8-MX10) located in the last four stages with select lines s7-s10.

For an 128-point computation, the multiplexer select lines are selected as shown in Table 6.4. A 128-point FFT computation is performed using the seven butterfly stages ST1-ST7, in which the sizes of the delay buffers in the seven butterfly stages are 64, 32, 16, 8, 4, 2 and 1, respectively using multiplexers MX1-MX6 with select lines

as '0'. The outputs of the 128-point FFT are obtained by selecting the DMX7 and MX8-MX10 as '0'. Based on the select lines as shown in Table 6.4, 64, 32 and 16-point FFT computations can also be obtained. Here 'x' denotes the don't care value. The reconfigurability of this architecture renders flexibility to perform 16/32/64/128-point FFT operations with less hardware.

6.3.2 Modified Complex Constant Multipliers

In [116], the authors have designed a configurable complex constant multipliers that reduces the size of ROM using $(N/8)$ sets of constant values for twiddle factor multiplication. However, the FFT architecture in [116] uses complex Booth multiplier that takes up more area and it is not suitable for variable-length FFTs. In this work, we proposed a configurable constant multiplier CCCM after stage 4 as shown in Figure 6.2. This multiplier can select any one of the coefficient W_{32}, W_{64} and W_{128} for multiplication, which is suitable for variable-length FFT.

For 16/32/64/128-point FFT, three complex constant multipliers (CCM1, CCM2 and CCCM) are employed as shown in Figure 6.2. Here, the non-trivial type of operations for CCM1 and CCCM structure can be obtained by using W_{16} and W_{128} multiplier units as a constant multiplier in Figure 6.3, respectively. The W_{16} constant multiplier is considered as shown in Figure 5.7. The CSD complex constant multiplier for CCM2 is shown in Figure 6.4, that uses a W_8 multiplier unit which is shown in

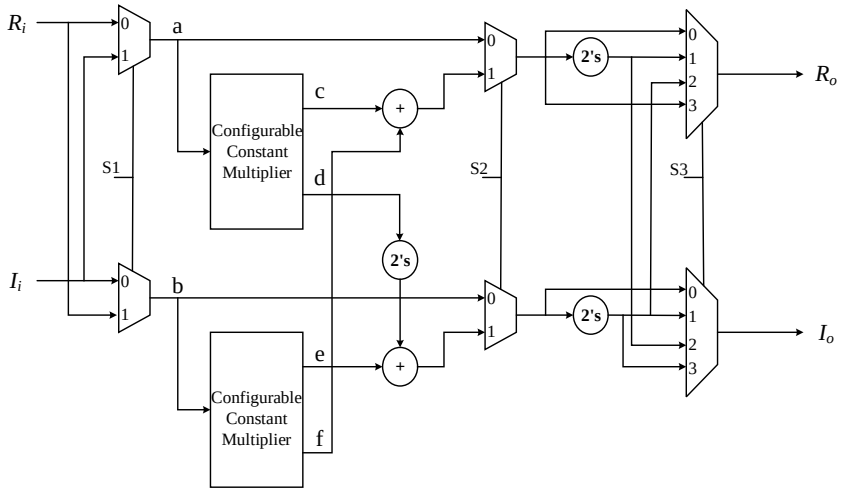

Figure 6.3: CSD complex constant multiplier for CCM1 and CCCM

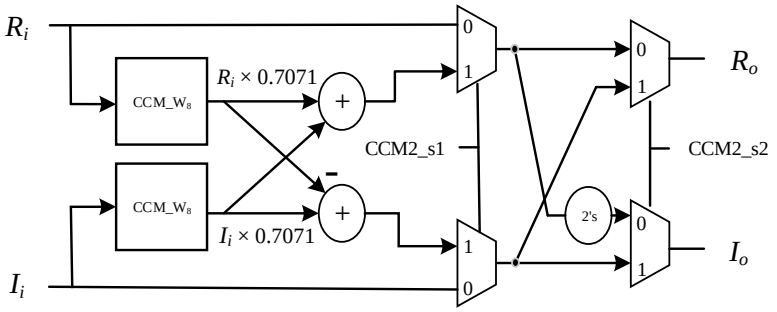

Figure 6.4: CSD complex constant multiplier for CCM2.

Figure 5.8. The W_{16} and W_8 constant multipliers are considered that are described in chapter 5 in sub-sections 5.2.2.2 and 5.2.2.3 respectively. The detailed description of the W_{128} constant multiplier is described below.

Table 6.5: CSD representations of 16 sets constant values for composing twiddle factors

Set	Constant value of the real part	CSD representation of real part												Constant value of the imaginary	CSD representation of imaginary part											
		2^0	2^{-1}	2^{-2}	2^{-3}	2^{-4}	2^{-5}	2^{-6}	2^{-7}	2^{-8}	2^{-9}	2^{-10}	2^{-11}		2^0	2^{-1}	2^{-2}	2^{-3}	2^{-4}	2^{-5}	2^{-6}	2^{-7}	2^{-8}	2^{-9}	2^{-10}	2^{-11}
1	0.9988	1	0	0	0	0	0	0	0	0	-1	0	1	0.0490	0	0	0	0	1	0	-1	0	0	1	0	0
2	0.9952	1	0	0	0	0	0	0	0	-1	0	-1	0	0.0980	0	0	0	1	0	-1	0	0	1	0	0	0
3	0.9892	1	0	0	0	0	0	-1	0	1	0	0	1	0.1467	0	0	0	1	0	1	0	-1	0	-1	0	0
4	0.9808	1	0	0	0	0	0	-1	0	-1	0	0	0	0.1951	0	0	1	0	-1	0	0	1	0	0	0	-1
5	0.9569	1	0	0	0	0	-1	0	0	0	0	1	0	0.2429	0	0	1	0	0	0	0	-1	0	0	0	1
6	0.9569	1	0	0	0	-1	0	1	0	1	0	0	-1	0.2903	0	0	1	0	0	1	0	1	0	0	1	0
7	0.9415	1	0	0	0	-1	0	0	0	1	0	0	0	0.3369	0	1	0	-1	0	-1	0	-1	0	0	0	1
8	0.9239	1	0	0	0	-1	0	-1	0	0	1	0	0	0.3827	0	1	0	-1	0	0	0	1	0	0	0	0
9	0.9040	1	0	0	-1	0	1	0	0	0	-1	0	-1	0.4276	0	1	0	0	-1	0	0	-1	0	-1	0	-1
10	0.8819	1	0	0	-1	0	0	0	1	0	0	-1	0	0.4714	0	1	0	0	0	-1	0	0	0	1	0	1
11	0.8577	1	0	0	-1	0	0	-1	0	0	-1	0	0	0.5141	0	1	0	0	0	0	1	0	0	-1	0	0
12	0.8315	1	0	-1	0	1	0	1	0	1	0	-1	0	0.5556	0	1	0	0	1	0	0	-1	0	0	1	0
13	0.8032	1	0	-1	0	1	0	0	-1	0	-1	0	0	0.5957	0	1	0	1	0	-1	0	0	0	1	0	-1
14	0.7730	1	0	-1	0	0	1	0	-1	0	0	0	1	0.6344	0	1	0	1	0	0	0	1	0	1	0	-1
15	0.7410	1	0	-1	0	0	0	0	-1	0	-1	0	1	0.6716	1	0	-1	0	-1	0	-1	0	0	0	0	-1
16	0.7071	1	0	-1	0	-1	0	1	0	1	0	0	0	0.7071	1	0	-1	0	-1	0	1	0	1	0	0	0

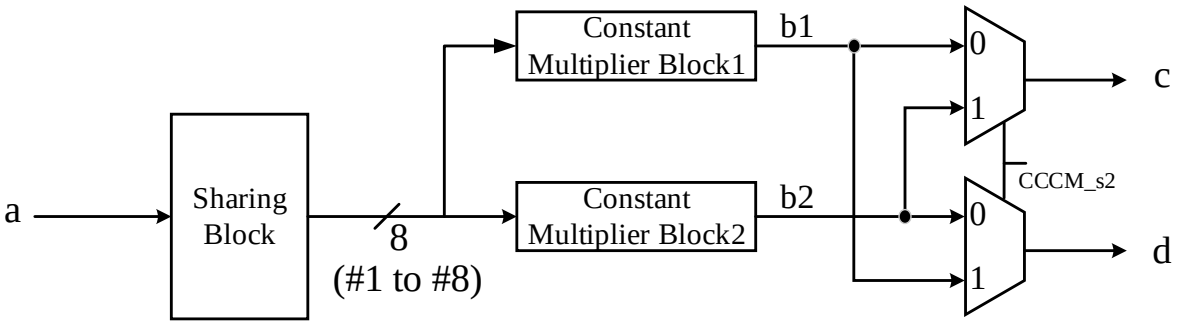

Figure 6.5: W_{128} configurable constant multiplier block diagram

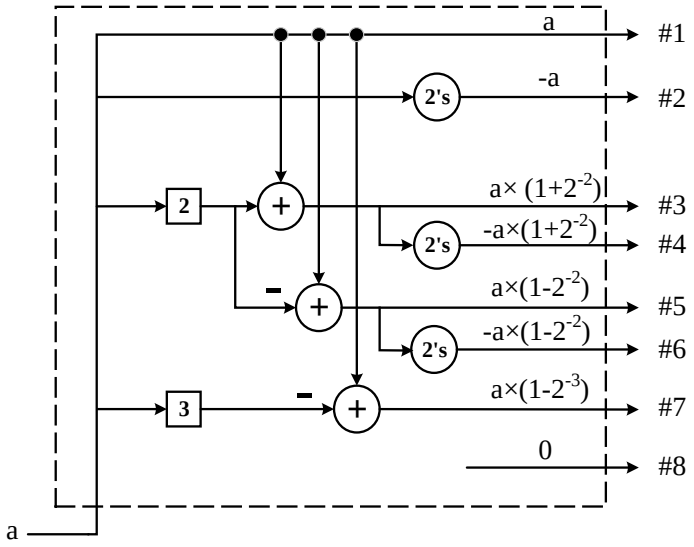

Figure 6.6: Sharing block

6.3.2.1 Configurable W_{128} Multiplier

In the proposed FFT architecture shown in Figure 6.2, the output data from stage 4 (ST4) will be multiplied by proper twiddle factors W_{32}, W_{64} and W_{128} for 32, 64 and 128-point FFT respectively. To reduce the multiplier area, a new configurable

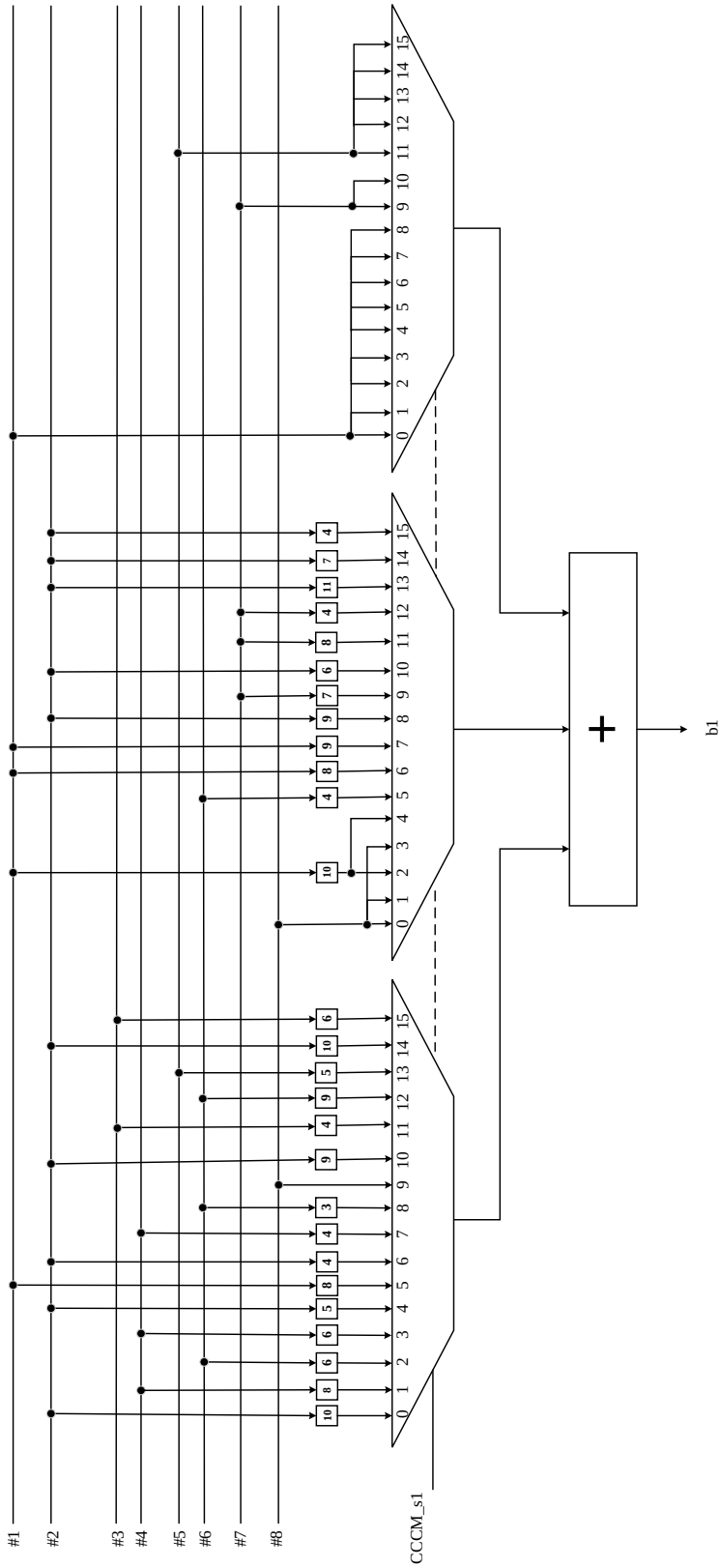

Figure 6.7: Constant Multiplier Block1

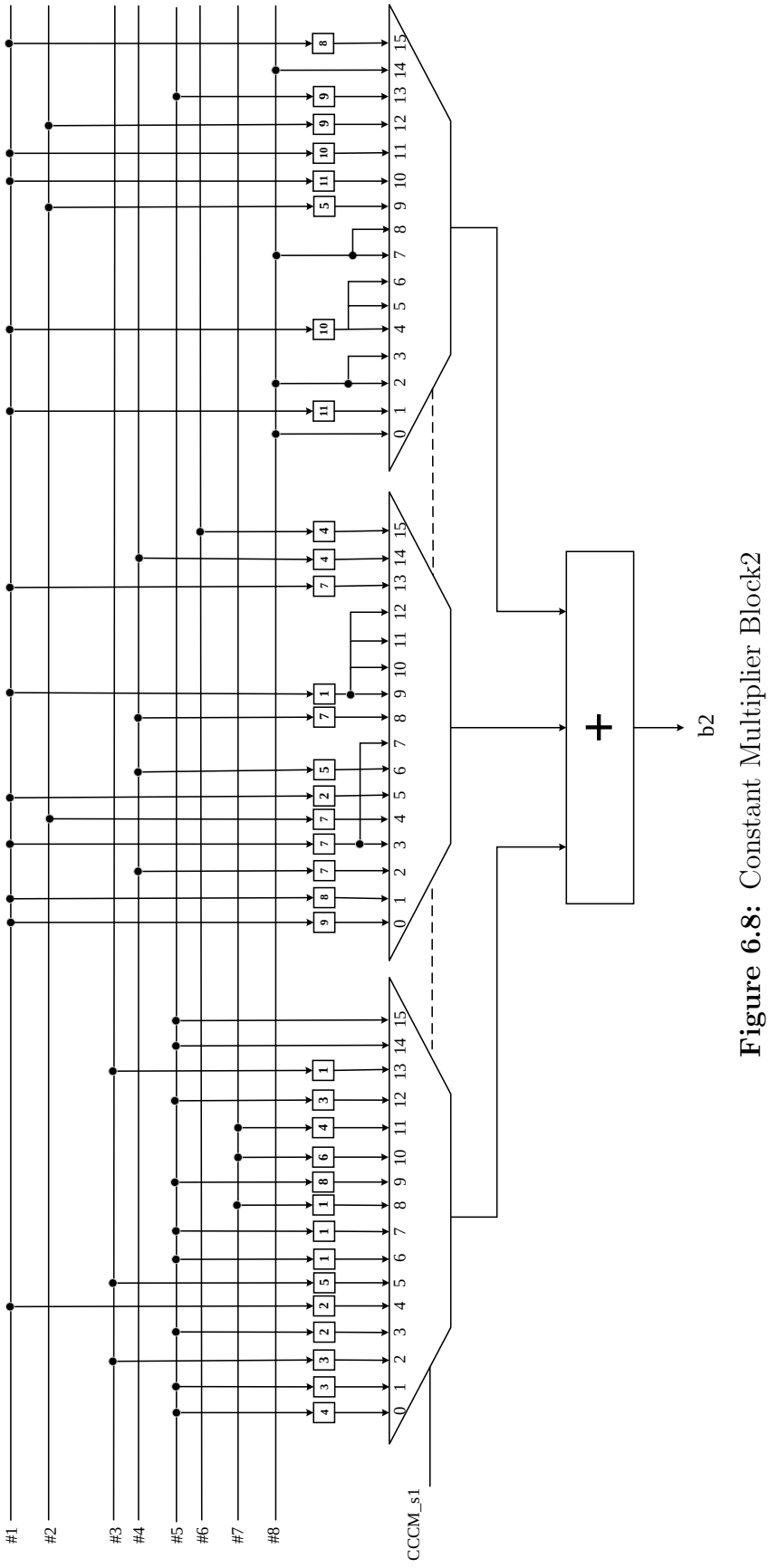

Figure 6.8: Constant Multiplier Block2

multiplier structure for twiddle factor multiplication is considered as shown in Figure 6.3. The corresponding block diagram of a constant multiplier W_{128} is shown in Figure 6.5. It contains a sharing block, two constant multiplier blocks (Block1 and Block2) for multiplication of real and imaginary coefficients with the data coming from the butterfly unit and two 2-to-1 multiplexers with select line $CCCM\_s2$.

For the multiplication operation of 128-point FFT, by considering the $N/8$ symmetry property only 16 sets of non-trivial constant values are required. A simplified design of the multiplier unit can be done by exploiting the fact that the complex constants to be multiplied are known a priori. Table 6.5 shows the CSD representation of 16 sets of complex constant values for composing the twiddle factors. Similar to the common sub-expression sharing methods as in chapter 5 (Table 5.2), the eight terms ("0", "1", "-1", "101", "10-1", "-101", "-10-1", and "100-1") in CSD form are used to construct the common sub-expression blocks. The sharing block consisting of shifters, adders and two's complement units is shown in Figure 6.6. The square boxes in Figure 6.6 represent right shifted values of the input given to it. These shifters are realized using simple hardware connections. For one among the 16 twiddle factor constant multiplications, six 16-to-1 multiplexers are used in the constant Multiplier Block1 and Block2 of Figure 6.7 and Figure 6.8 respectively, to get the appropriate result. The data of each path are fed to appropriate constant W_{32}, W_{64} and W_{128} multipliers according to scheduling of the twiddle factor, as shown in Table 6.6. For example, W_{128}^{1}-W_{128}^{16} multiplication can be obtain by selecting the multiplexer select

Table 6.6: Selection of the Twiddle Factors in CCCM

Twiddle Factors / Control Signals	1	W_{128}^{1}-W_{128}^{16}	W_{128}^{17}-W_{128}^{31}	$-j$	W_{128}^{33}-W_{128}^{48}	W_{128}^{49}-W_{128}^{63}	-1	W_{128}^{65}-W_{128}^{80}	W_{128}^{81}-W_{128}^{95}
S1	0	0	0	1	1	1	0	0	0
S2	0	1	1	0	1	1	0	1	1
S3	0	0	0	3	2	2	1	1	1
CCCM_s1	x	0-15	14-0	x	0-15	14-0	x	0-15	14-0
CCCM_s2	x	0	1	x	1	0	x	0	1

lines $CCCM\_s1$ and $CCCM\_s2$ as $(0-15)$ and '0' respectively.

Due to the recursion property $W_N^2 = W_{N/2}$, W_{32} and W_{64} twiddle factor coefficeints can be obtain from W_{128}. For example, $W_{128}^2 = W_{64}^1$ and $W_{128}^4 = W_{32}^1$, the other values can also obtain from W_{128}. Therefore, the entire constant multiplication calculation can be implemented using 16 sets of constant values by swapping the real and imaginary parts appropriately and choosing the appropriate sign, following the mapping table (Table 6.6).

So far, the complex constant multipliers are proposed in the literature [132, 133, 134] for the twiddle factor multiplications of W_8, W_{16}, W_{32} and W_{64}. However, all of these multipliers are fixed constant multipliers that cannot be useful for variable-length FFT architectures. Therefore, we proposed W_{128} configurable complex constant multiplier. It can perform the twiddle factor W_{32} and W_{64} multiplication, which are required for variable-length (16/32/64/128-point) selection of the proposed FFT. However, the constant multipliers are preferred for less than 256-point FFT, as the number of non-trivial constant values increases for greater than 128-point FFT.

6.4 Comparison and Results

The hardware requirement and performance comparison of the proposed SDF architecture with other pipelined architectures for the computation of a 128-point FFT is shown in Table 6.7. The first column shows the type of architecture. The second, third, fourth and fifth columns show the hardware requirements for the architecture: complex adders, complex multipliers, CSD complex constant multipliers, and memory respectively. The last two columns compares the performance in terms of critical path delay and throughput. The critical path delay is defined as the maximum delay that it takes in the pipeline architecture. Finally, the throughput (R) indicates the number of samples processed by the architecture per clock cycle.

In Table 6.7, it can be observed that all SDF architectures require same number of complex adders and memory with a throughput of R. These architectures differ in terms of complex multipliers and critical path delay. In [24], the FFT architecture employed six complex multipliers (Booth's multiplier) for the multiplication operation which takes more area. In [131], the multiplier in the n^{th} stage of R2SDF FFT is replaced by the pass logic that reduced the area of multiplier. However, the number of complex multipliers are still high. Therefore, in [125, 135] the number of complex multipliers are reduced to three by using Booth's multiplier. However, most of the existing designs [24, 125, 131, 135] uses complex Booth's multiplier for the twiddle factor W_{128} multiplication. In the proposed design, we have designed a CCCM for

Table 6.7: Comparison of the proposed architecture to other architectures for the computation of a 128-point FFT

Architecture	Complex Adders	Complex Multipliers	CSD Constant Multipliers	Memory	Critical Path Delay	Throughput
R2SDF [24]	14	6	0	127	$T_M + 2T_A + T_{MUX}$	R
R2^2SDF [125]	14	3	0	127	$T_M + 3T_A + T_{MUX}$	R
R2^3SDF [135]	14	3	0	127	$T_M + 3T_A + T_{MUX}$	R
R2SDF [131]	14	5	1	127	$T_M + 2T_A + T_{MUX}$	R
Proposed	14	0	3	127	$5T_A + 3T_C + 5T_{MUX}$	R

the twiddle factor W_{128} multiplication. Further, we have employed two CSD complex constant multipliers CCM1, and and CCM2 for W_{16} and W_8 twiddle factor multiplication respectively. As the proposed constant multipliers does not require any complex Booth's multiplier, these can reduce the area of entire FFT design.

The critical path delay of the proposed design is $5T_A + 3T_C + 5T_{MUX}$, where T_A, T_C and T_{MUX} are computation time of an adder, two's complement, and multiplexer respectively. The critical path delay of the proposed architecture is less compared to other architectures as it does not include the computation time of multiplier (T_M).

At first, the proposed 128/64/32/16-point FFT architecture is modeled in simulink using MATLAB. Based on the simulation results using MATLAB, we determined the word length of the proposed FFT to be 12-bits in both real and imaginary parts in order to meet IEEE $802.15.4-g$ system requirements. Then the design is converted to Verilog using HDL coder. Finally, it has been implemented in Virtex-5 FPGA device XC5VLX110T-1FF1136. Table 6.8 shows the utilization report in terms of slices and DSP blocks. Compared to the architectures in [24, 131], the proposed architecture

Table 6.8: Comparison of the proposed architecture to other architectures for the computation of a 128-point FFT

Virtex-5	Algorithm	Slices	DSP blocks
[24]	R2SDF	9939	64
[131]	R2SDF	9468	64
Proposed	Mixed-radix	7952	0

utilizes less hardware slices and also does not used any DSP blocks. The reduction in area of the proposed design is due to the usage of complex constant multipliers.

Furthermore, the proposed architecture is implemented using UMC 90 nm CMOS Technology with a supply voltage of 1 V. The design is synthesized using Synopsis Design Compiler and Prime Time is used to calculate the power consumption. Table 6.9 compares the post-synthesis results of the proposed design to other 128-point FFT architectures reported in the literature [17, 131]. As various designs use different CMOS technologies and supply voltage, to have a meaningful comparison, the total gate count based on 2 x 1 NAND gates and normalized power are considered as comparison parameters. The normalized power is calculated as

$$Normalized\ power = \frac{Power\ consumption}{(Tech./90\ nm) \times (V_{DD}/1.0)^2} \tag{6.1}$$

From Table 6.9, it is evident that the proposed architecture gives an advantage in terms of hardware complexity compared to the architecture in [131]. The reduction in hardware is due to CCCM structure and hardware-sharing mechanism which reduces

Table 6.9: Comparison of various 128-point FFT architectures

	[17]	[131]	Proposed
Process (nm)	180	180	90
Level of FFT length / streams	Variable/Single or Multiple	Variable/Single	Variable/Single
Operation Mode	128/64	8 to 128	16 to 128
Wordlength (bits)	16	16	12
Voltage (V)	1.8	1.8	1
Gate Count	–	69,665	22,329
Power (mW)	65	99.5	3.832
Normalized Power (mW)	10.030	15.354	3.832

the memory space requirements. With respect to normalized power, the proposed design outperforms compared to the architectures reported in [17, 131].

6.5 Conclusion

A novel 16/32/64/128-point SDF pipeline FFT architecture based on mixed-radix algorithm has been proposed for IEEE $802.15.4-g$ systems. The proposed architecture is the most area-efficient architecture by employing several performance-enhancement techniques, including a reformulated radix-2^2 and radix-2 algorithms, a new configurable complex constant multiplier and a hardware-sharing mechanism to map the memory. The FPGA implementation results of the proposed architecture shows that it is more area-efficient compared to the earlier reported design. Furthermore, ASIC implementation of the proposed design reveals that it reduces the total gate count and power consumption.

Chapter 7

Conclusions and Future Work

In this thesis, the possibilities to improve complexity and performance of the twiddle factor multiplications in FFT at algorithmic, architecture and arithmetic level were investigated. Based on the optimization of the number of multiplication operations at algorithmic level, it is shown that either radix-2^i or mixed-radix algorithm is most suitable for FFT implementation of IEEE 802.11a and IEEE 802.15.4 $-$ g standard OFDM systems. These FFT algorithms will have the same butterfly operations and data flow of radix-2 algorithm, but differ in the twiddle factor multiplication. The difference among these FFT algorithms lies in terms of the number of non-trivial multiplications, the coefficient memory complexity, and the accuracy, which are related to the area, power consumption and performance of the circuit.

Different FFT processor architectures are compared on the architecture level and it is found that SDF architecture is the ideal choice for OFDM based IEEE 802.11a and IEEE 802.15.4 $-$ g systems with regard to the memories and arithmetic blocks utilization. Important FFT architecture choices and considerations are also examined and concluded.

The thesis also investigated two types of techniques to implement twiddle factor multiplication, which includes general complex multiplication and constant multiplication. Among these techniques, the constant multiplication is preferred for lower twiddle factor coefficients (less than 256-point FFTs), as it is achieved by using shifters and adders with less area than the general complex multiplication.

Based on the analysis of FFT at various levels, R2^{2}SDF and R2^{3}SDF pipelined 64-point FFT architectures are proposed that are suitable for IEEE 802.11a OFDM systems. The proposed architectures focus on improving the efficiency of complex multipliers for non-trivial twiddle factors. It takes small-area and consumes low-power for IEEE 802.11a systems. The functionality of the proposed FFT architectures are verified and ASIC based synthesis results are presented.

For MR-OFDM physical layer of IEEE 802.15.4 $-$ g system, a 16/32/64/128-point SDF pipeline FFT architecture is proposed. To reduce the area of the architecture, it employed a mixed-radix algorithm, a new configurable complex constant multiplier and a hardware sharing mechanism. Both ASIC and FPGA based synthesis results

are provided for the proposed variable-length FFT architecture. The results show that the proposed FFT architecture takes less area and consumes low power by considering modified constant multipliers.

7.1 Future Work

This thesis has focused on the SDF FFT architecture with twiddle factor multiplications implemented using shift-and-add. The SDF architecture was chosen due to its serial nature for low data rate applications. However, it is possible to apply this method to parallel pipelined architectures for high throughput applications. The hardware complexity increases for high throughput applications, therefore there is a scope to reduce the hardware complexity by using the modified constant multipliers in FFT architectures.

Bibliography

[1] James Cooley and John Tukey. An algorithm for the Machine Calculation of Complex fourier series. *Mathematics of Computation*, 19(90):297–301, 1965.

[2] Sanjit Kumar Mitra and Yonghong Kuo. *Digital signal processing: a computer-based approach*, volume 2. McGraw-Hill New York, 2006.

[3] Winthrop W Smith and Joanne M Smith. *Handbook of real-time fast Fourier transforms*, volume 55. IEEE press New York, 1995.

[4] Pierre Duhamel and Martin Vetterli. Fast Fourier transforms: a tutorial review and a state of the art. *Signal processing*, 19(4):259–299, 1990.

[5] Michael T Heideman, Don H Johnson, and C Sidney Burrus. Gauss and the history of the fast Fourier transform. *Archive for history of exact sciences*, 34 (3):265–277, 1985.

[6] Gordon Charles Danielson and Cornelius Lanczos. Some improvements in practical Fourier analysis and their application to X-ray scattering from liquids. *Journal of the Franklin Institute*, 233(4):365–380, 1942.

[7] C Runge. Zeit. f. *Math. u. Phys*, 48:443–456, 1903.

[8] Yizheng Liao. *Phase and Frequency Estimation–High-Accuracy and Low-Complexity Techniques*. PhD thesis, Worcester Polytechnic Institute, 2011.

[9] Steven W Smith et al. The scientist and engineer's guide to digital signal processing. 1997.

[10] Martin Fürer. Faster integer multiplication. *SIAM Journal on Computing*, 39 (3):979–1005, 2009.

[11] Alan V Oppenheim and Jae S Lim. The importance of phase in signals. *Proceedings of the IEEE*, 69(5):529–541, 1981.

[12] Xuelei Sherry Ni and Xiaoming Huo. Statistical interpretation of the importance of phase information in signal and image reconstruction. *Statistics & probability letters*, 77(4):447–454, 2007.

[13] Karl R Gegenfurtner, Doris I Braun, and Felix A Wichmann. The importance of phase information for recognizing natural images. *Journal of Vision*, 3(9): 519–519, 2003.

[14] Jun-Rim Choi, Soo-Bok Park, Dong-Seok Han, and Se-Ho Park. A 2048 complex point fft architecture for digital audio broadcasting system. In *IEEE International Symposium on Circuits and Systems*, volume 5, pages 693–696. IEEE, 2000.

[15] Richard M Jiang. An area-efficient fft architecture for ofdm digital video broadcasting. *IEEE Transactions on Consumer Electronics*, 53(4):1322–1326, 2007.

[16] Shen-Jui Huang and Sau-Gee Chen. A memory-efficient continuous-flow FFT processor for Wimax application. In *IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 17–20. IEEE, 2012.

[17] Bo Fu and Paul Ampadu. An area efficient FFT/IFFT processor for MIMO-OFDM WLAN 802.11 n. *Journal of Signal Processing Systems*, 56(1):59–68, 2009.

[18] Sheng-Yeng Peng, Kai-Ting Shr, Chao-Ming Chen, and Yuan-Hao Huang. Energy-efficient 128 2048/1536-point FFT processor with resource block mapping for 3GPP-LTE system. In *International Conference on Green Circuits and Systems (ICGCS)*, pages 14–17. IEEE, 2010.

[19] W Morven Gentleman and Gordon Sande. Fast fourier transforms: for fun and profit. In *Proceedings of the November 7-10, 1966, fall joint computer conference*, pages 563–578. ACM, 1966.

[20] Glenn D Bergland. Numerical Analysis: A fast Fourier transform algorithm for real-valued series. *Communications of the ACM*, 11(10):703–710, 1968.

[21] Qing Li, Nengchao Wang, Baochang Shi, and Chuguang Zheng. Extendible look-up table of twiddle factors and radix-8 based fast fourier transform. *Signal processing*, 82(4):643–648, 2002.

[22] Daisuke Takahashi. A radix-16 fft algorithm suitable for multiply-add instruction based on goedecker method. In *International Conference on Multimedia and Expo*, volume 2, pages II–845. IEEE, 2003.

[23] P. Duhamel and H. Hollmann. 'Split radix' FFT algorithm. *Electronics Letters*, 20(1):14–16, 1984. ISSN 0013-5194. doi: 10.1049/el:19840012.

[24] Shousheng He and M. Torkelson. A new approach to pipeline FFT processor. In *The 10th International Parallel Processing Symposium*, pages 766–770, 1996. doi: 10.1109/IPPS.1996.508145.

[25] Shousheng He and Torkelson. Designing pipeline FFT processor for OFDM (De) modulation. In *URSI International Symposium on Signals, Systems, and Electronics*, pages 257–262. IEEE, 1998.

[26] OH Jung-Yeol and LIM Myoung-Seob. New radix-2 to the 4th power pipeline FFT processor. *IEICE transactions on electronics*, 88(8):1740–1746, 2005.

[27] Taesang Cho, Hanho Lee, Jounsup Park, and Chulgyun Park. A high-speed low-complexity modified radix-2^5 FFT processor for gigabit WPAN applications. In

IEEE International Symposium on Circuits and Systems (ISCAS), pages 1259–1262, 2011. doi: 10.1109/ISCAS.2011.5937799.

[28] M. Garrido, J. Grajal, M.A. Sanchez, and O. Gustafsson. Pipelined Radix-2^k Feedforward FFT Architectures. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 21(1):23–32, 2013. ISSN 1063-8210. doi: 10.1109/TVLSI.2011.2178275.

[29] Keshab K Parhi. *VLSI digital signal processing systems: design and implementation*. John Wiley & Sons, 2007.

[30] Koushik Maharatna, Eckhard Grass, and Ulrich Jagdhold. A 64-point fourier transform chip for high-speed wireless LAN application using OFDM. *IEEE Journal of Solid-State Circuits*, 39(3):484–493, 2004.

[31] Fahad Qureshi and Oscar Gustafsson. Low-complexity constant multiplication based on trigonometric identities with applications to ffts. *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, 94 (11):2361–2368, 2011.

[32] William T Cochran, James W Cooley, David L Favin, Howard D Helms, Reginald A Kaenel, William W Lang, George C Maling Jr, David E Nelson, Charles M Rader, and Peter D Welch. What is the fast Fourier transform? *Proceedings of the IEEE*, 55(10):1664–1674, 1967.

[33] Oscar Buneman. Inversion of the helmholtz (or laplace-poisson) operator for slab geometry. *Journal of Computational Physics*, 12(1):124–130, 1973.

[34] Fred J. Taylor, G Papadourakis, Alexander Skavantzos, and A Stouraitis. A radix-4 FFT using complex RNS arithmetic. *IEEE Transactions on Computers*, 100(6):573–576, 1985.

[35] R Yavne. An economical method for calculating the discrete fourier transform. In *Proceedings of the December 9-11, 1968, fall joint computer conference, part I*, pages 115–125. ACM, 1968.

[36] P. Duhamel. Implementation of "Split-radix" FFT algorithms for complex, real, and real-symmetric data. *IEEE Transactions on Acoustics, Speech and Signal Processing*, 34(2):285–295, 1986. ISSN 0096-3518. doi: 10.1109/TASSP.1986.1164811.

[37] Martin Vetterli and Pierre Duhamel. Split-radix algorithms for length-p/sup m/dft's. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 37(1): 57–64, 1989.

[38] Daisuke Takahashi. An extended split-radix fft algorithm. *IEEE Signal Processing Letters*, 8(5):145–147, 2001.

[39] Weihua Zheng, Kenli Li, and Keqin Li. A fast algorithm based on srfft for length dfts. *IEEE Transactions on Circuits and Systems II: Express Briefs*, 61 (2):110–114, 2014.

[40] H V Sorensen, D Jones, Ml Heideman, and C Burrus. Real-valued fast Fourier transform algorithms. *IEEE Transactions on Acoustics, Speech and Signal Processing*, 35(6):849–863, 1987.

[41] E Oran Brigham. The fast fourier transform and its applications. *UK: Prentice Hall*, 1988.

[42] G Bergland. A radix-eight fast Fourier transform subroutine for real-valued series. *IEEE Transactions on Audio and Electroacoustics*, 17(2):138–144, 1969.

[43] B Raja Sekhar and KMM Prabhu. Radix-2 decimation-in-frequency algorithm for the computation of the real-valued fft. *IEEE transactions on signal processing*, 47(4):1181–1184, 1999.

[44] Mario Garrido, Keshab K Parhi, and Jesús Grajal. A pipelined FFT architecture for real-valued signals. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 56(12):2634–2643, 2009.

[45] M. Ayinala, Y. Lao, and K. K. Parhi. An in-place fft architecture for real-valued signals. *IEEE Transactions on Circuits and Systems II: Express Briefs*, 60(10): 652–656, Oct 2013. ISSN 1549-7747. doi: 10.1109/TCSII.2013.2273841.

[46] Pramod Kumar Meher, Basant Kumar Mohanty, Sujit Kumar Patel, Soumya Ganguly, and Thambipillai Srikanthan. Efficient vlsi architecture for decimation-in-time fast fourier transform of real-valued data. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 62(12):2836–2845, 2015.

[47] Ronald N Bracewell. Discrete hartley transform. *JOSA*, 73(12):1832–1835, 1983.

[48] Huazhong Shu, Xudong Bao, Christine Toumoulin, and Limin Luo. Radix-3 algorithm for the fast computation of forward and inverse mdct. *IEEE Signal Processing Letters*, 14(2):93–96, 2007.

[49] J. Markel. FFT pruning. *IEEE Transactions on Audio and Electroacoustics*, 19 (4):305–311, 1971. ISSN 0018-9278. doi: 10.1109/TAU.1971.1162205.

[50] Haitham Hassanieh, Piotr Indyk, Dina Katabi, and Eric Price. Nearly Optimal Sparse Fourier Transform. *CoRR*, abs/1201.2501, 2012.

[51] Eyal Kushilevitz and Yishay Mansour. Learning decision trees using the Fourier spectrum. *SIAM Journal on Computing*, 22(6):1331–1348, 1993.

[52] Anna C Gilbert, Sudipto Guha, Piotr Indyk, S Muthukrishnan, and Martin Strauss. Near-optimal sparse Fourier representations via sampling. In *Proceedings of the thiry-fourth annual ACM symposium on Theory of computing*, pages 152–161. ACM, 2002.

[53] Anna C Gilbert, S Muthukrishnan, and M Strauss. Improved time bounds for near-optimal sparse Fourier representations. In *Optics & Photonics*, pages 59141A–59141A. International Society for Optics and Photonics, 2005.

[54] Mark A Iwen. Combinatorial sublinear-time Fourier algorithms. *Foundations of Computational Mathematics*, 10(3):303–338, 2010.

[55] Omid Abari, Ezz Hamed, Haitham Hassanieh, Abhinav Agarwal, Dina Katabi, Anantha P Chandrakasan, and Vladimir Stojanovic. 27.4 a 0.75-million-point fourier-transform chip for frequency-sparse signals. In *IEEE International Solid-State Circuits Conference Digest of Technical Papers (ISSCC)*, pages 458–459. IEEE, 2014.

[56] Abhinav Agarwal, Haitham Hassanieh, Omid Abari, Ezz Hamed, Dina Katabi, et al. High-throughput implementation of a million-point sparse fourier transform. In *24th International Conference on Field Programmable Logic and Applications (FPL)*, pages 1–6. IEEE, 2014.

[57] Shaogang Wang, Vishal M Patel, and Athina Petropulu. The robust sparse fourier transform (rsft) and its application in radar signal processing. *IEEE Trans. Aerosp. Electron. Syst.*, 53(6):2735–2755, 2017.

[58] Guoan Bi and Yan Qiu Chen. Fast dft algorithms for length n= q* 2 m. *IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing*, 45(6):685–690, 1998.

[59] Saad Bouguezel, M Omair Ahmad, and MN Srikanta Swamy. A new radix-2/8 fft algorithm for length-q× 2 m dfts. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 51(9):1723–1732, 2004.

[60] Kenli Li, Weihua Zheng, and Keqin Li. A fast algorithm with less operations for length-dfts. *IEEE Transactions on Signal Processing*, 63(3):673–683, 2015.

[61] Charles Van Loan. *Computational frameworks for the fast Fourier transform.* SIAM, 1992.

[62] D Harris, James H McClellan, D Chan, and H Schuessler. Vector radix fast fourier transform. In *IEEE International Conference on Acoustics, Speech, and Signal Processing*, volume 2, pages 548–551. IEEE, 1977.

[63] Henri J. Nussbaumer and Philippe Quandalle. Computation of convolutions and discrete fourier transforms by polynomial transforms. *IBM Journal of Research and Development*, 22(2):134–144, 1978.

[64] Stanley C Chan and KL Ho. Split vector-radix fast fourier transform. *Signal Processing, IEEE Transactions on*, 40(8):2029–2039, 1992.

[65] Zhaodou Chen and Lijing Zhang. Vector coding algorithms for multidimensional discrete fourier transform. *Journal of Computational and Applied Mathematics*, 212(1):63–74, 2008.

[66] I PRESENT. Cramming more components onto integrated circuits. *Readings in computer architecture*, 56, 2000.

[67] Hoi-Kwong Lo, Tim Spiller, and Sandu Popescu. *Introduction to quantum computation and information.* World Scientific, 1998.

[68] Mahdi Aminian, Mehdi Saeedi, Morteza Saheb Zamani, and Mehdi Sedighi. Fpga-based circuit model emulation of quantum algorithms. In *IEEE Computer Society Annual Symposium on VLSI*, pages 399–404. IEEE, 2008.

[69] José F Rivera-Miranda, Álvaro J Caicedo-Beltrán, Juan D Valencia-Payán, John M Espinosa-Duran, and Jaime Velasco-Medina. Hardware emulation of quantum fourier transform. In *IEEE Second Latin American Symposium on Circuits and Systems*, pages 1–4. IEEE, 2011.

[70] M Khalil-Hani, YH Lee, and MN Marsono. An accurate fpga-based hardware emulation on quantum fourier transform. *Quantum*, 1:a1b3, 2015.

[71] Gabriel Popkin. Quest for qubits, 2016.

[72] Ahmed Usman Khalid, Zeljko Zilic, and Katarzyna Radecka. Fpga emulation of quantum circuits. In *IEEE International Conference on Computer Design: VLSI in Computers and Processors*, pages 310–315. IEEE, 2004.

[73] S Magar, S Shen, G Luikuo, M Fleming, and R Aguilar. An application specific DSP chip set for 100 MHz data rates. In *International Conference on Acoustics, Speech, and Signal Processing*, pages 1989–1992. IEEE, 1988.

[74] Glen Sunada, Jain Jin, Matt Berzins, and Tom Chen. COBRA: An 1.2 million transistor expandable column FFT chip. In *IEEE International Conference on Computer Design: VLSI in Computers and Processors*, pages 546–550. IEEE, 1994.

[75] Bevan M Baas. A low-power, high-performance, 1024-point FFT processor. *IEEE Journal of Solid-State Circuits*, 34(3):380–387, 1999.

[76] Bevan M Baas. A generalized cached-fft algorithm. In *IEEE International Conference on Acoustics, Speech, and Signal Processing*, volume 5, pages v–89. IEEE, 2005.

[77] Herbert L Groginsky and George A Works. A pipeline fast Fourier transform. *IEEE Transactions on Computers*, 100(11):1015–1019, 1970.

[78] N. Aghaee and M. Eshghi. Design of a pipelined r4sdf processor. In *2009 17th European Signal Processing Conference*, pages 963–967, Aug 2009.

[79] Song-Nien Tang, Jui-Wei Tsai, and Tsin-Yuan Chang. A 2.4-GS/s FFT processor for OFDM-based WPAN applications. *IEEE Transactions on Circuits and Systems II: Express Briefs*, 57(6):451–455, 2010.

[80] Erling H Wold and Alvin M. Despain. Pipeline and parallel-pipeline FFT processors for VLSI implementations. *IEEE Transactions on Computers*, 100(5): 414–426, 1984.

[81] Hang Liu and Hanho Lee. A high performance four-parallel 128/64-point radix-2^4 FFT/IFFT processor for MIMO-OFDM systems. In *IEEE Asia Pacific Conference on Circuits and Systems*, pages 834–837. IEEE, 2008.

[82] Nuo Li and NP Van Der Meijs. A Radix-2^2 based parallel pipeline FFT processor for MB-OFDM UWB system. In *IEEE International SOC Conference*, pages 383–386. IEEE, 2009.

[83] Lawrence R Rabiner and Bernard Gold. Theory and application of digital signal processing. *Englewood Cliffs, NJ, Prentice-Hall, Inc., 1975. 777 p.*, 1975.

[84] Earl E Swartzlander, Wendell KW Young, and Saul J Joseph. A radix 4 delay commutator for fast Fourier transform processor implementation. *IEEE Journal of Solid-State Circuits*, 19(5):702–709, 1984.

[85] Guan Bi and EV Jones. A pipelined FFT processor for word-sequential data. *IEEE Transactions on Acoustics, Speech and Signal Processing*, 37(12):1982–1985, 1989.

[86] Charles R Baugh and Bruce A Wooley. A two's complement parallel array multiplication algorithm. *IEEE Transactions on Computers*, 22(12):1045–1047, 1973.

[87] Christopher S Wallace. A suggestion for a fast multiplier. *IEEE Transactions on electronic Computers*, (1):14–17, 1964.

[88] Andrew D Booth. A signed binary multiplication technique. *The Quarterly Journal of Mechanics and Applied Mathematics*, 4(2):236–240, 1951.

[89] Yuan-Ho Chen. An accuracy-adjustment fixed-width booth multiplier based on multilevel conditional probability. *IEEE Trans. Very Large Scale Integr. (VLSI) Syst*, 23(1):203–207, 2015.

[90] G Ganesh Kumar and V Charishma. Design of high speed vedic multiplier using vedic mathematics techniques. *International Journal of Scientific and Research Publications*, 2(3):1, 2012.

[91] EE Swartzlander. The quasi-serial multiplier. *IEEE Transactions on Computers*, 100(4):317–321, 1973.

[92] Hawkins H Yao and EE Swartzlander. Serial-parallel multipliers. In *Signals, Systems and Computers, 1993. 1993 Conference Record of The Twenty-Seventh Asilomar Conference on*, pages 359–363. IEEE, 1993.

[93] Sunder S Kidambi, Fayez El-Guibaly, and Andreas Antoniou. Area-efficient multipliers for digital signal processing applications. *IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing*, 43(2):90–95, 1996.

[94] YC Lim. Single-precision multiplier with reduced circuit complexity for signal processing applications. *IEEE Transactions on Computers*, 41(10):1333–1336, 1992.

[95] Michael J Schulte and Earl E Swartzlander Jr. Truncated multiplication with correction constant [for dsp]. In *[Workshop on] VLSI Signal Processing*, pages 388–396. IEEE, 1993.

[96] Michael J Schulte, James E Stine, and John G Jansen. Reduced power dissipation through truncated multiplication. In *IEEE Alessandro Volta Memorial Workshop on Low-Power Design*, pages 61–69. IEEE, 1999.

[97] Eric J King and Earl E Swartzlander Jr. Data-dependent truncation scheme for parallel multipliers. In *Conference Record of the Thirty-First Asilomar Conference on Signals, Systems & Computers*, volume 2, pages 1178–1182. IEEE, 1997.

[98] Earl E Swartzlander Jr. Truncated multiplication with approximate rounding. In *Conference Record of the Thirty-Third Asilomar Conference on Signals, Systems, and Computers*, volume 2, pages 1480–1483. IEEE, 1999.

[99] James E Stine and Oliver M Duverne. Variations on truncated multiplication. In *Euromicro Symposium on Digital System Design*, pages 112–119. IEEE, 2003.

[100] Hyuk Park and Earl E Swartzlander Jr. Truncated multiplication with symmetric correction. In *Fortieth Asilomar Conference on Signals, Systems and Computers*, pages 931–934. IEEE, 2006.

[101] Yen-Chin Liao, Hsie-Chia Chang, and Chih-Wei Liu. Carry estimation for two's complement fixed-width multipliers. In *IEEE Workshop on Signal Processing Systems Design and Implementation*, pages 345–350. IEEE, 2006.

[102] Jer Min Jou, Shiann Rong Kuang, and Ren Der Chen. Design of low-error fixed-width multipliers for dsp applications. *IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing*, 46(6):836–842, 1999.

[103] Lan-Da Van, Shuenn-Shyang Wang, and Wu-Shiung Feng. Design of the lower error fixed-width multiplier and its application. *IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing*, 47(10):1112–1118, 2000.

[104] Lan-Da Van and Chih-Chyau Yang. Generalized low-error area-efficient fixed-width multipliers. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 52(8):1608–1619, 2005.

[105] Antonio GM Strollo, Nicola Petra, and Davide De Caro. Dual-tree error compensation for high performance fixed-width multipliers. *IEEE Transactions on Circuits and Systems II: Express Briefs*, 52(8):501–507, 2005.

[106] Nicola Petra, Davide De Caro, and Antonio GM Strollo. Design of fixed-width multipliers with minimum mean square error. In *18th European Conference on Circuit Theory and Design*, pages 464–467. IEEE, 2007.

[107] Nicola Petra, Davide De Caro, Valeria Garofalo, Ettore Napoli, and Antonio GM Strollo. Truncated binary multipliers with variable correction and minimum mean square error. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 57(6):1312–1325, 2010.

[108] Nicola Petra, Davide De Caro, Valeria Garofalo, Ettore Napoli, and Antonio Giuseppe Maria Strollo. Design of fixed-width multipliers with linear compensation function. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 58(5):947–960, 2011.

[109] Davide De Caro, Nicola Petra, Antonio Giuseppe Maria Strollo, Flaviano Tessitore, and Ettore Napoli. Fixed-width multipliers and multipliers-accumulators with min-max approximation error. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 60(9):2375–2388, 2013.

[110] I Wey, Chun-Chien Wang, et al. Low-error and hardware-efficient fixed-width multiplier by using the dual-group minor input correction vector to lower input correction vector compensation error. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 20(10):1923–1928, 2012.

[111] I-Chyn Wey, Chien-Chang Peng, and Feng-Yu Liao. Reliable low-power multiplier design using fixed-width replica redundancy block. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 23(1):78–87, 2015.

[112] Hong-An Huang, Yen-Chin Liao, and Hsie-Chia Chang. A self-compensation fixed-width booth multiplier and its 128-point fft applications. In *2006 IEEE International Symposium on Circuits and Systems*, pages 4–pp. IEEE, 2006.

[113] Yuan-Ho Chen, Chung-Yi Li, and Tsin-Yuan Chang. Area-effective and power-efficient fixed-width booth multipliers using generalized probabilistic estimation bias. *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, 1(3):277–288, 2011.

[114] Yuan-Ho Chen and Tsin-Yuan Chang. A high-accuracy adaptive conditional-probability estimator for fixed-width booth multipliers. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 59(3):594–603, 2012.

[115] Sang-In Cho and Kyu-Min Kang. A low-complexity 128-point mixed-radix FFT processor for MB-OFDM UWB systems. *ETRI journal*, 32(1):1–10, 2010.

[116] Chu Yu, Mao-Hsu Yen, Pao-Ann Hsiung, and Sao-Jie Chen. A low-power 64-point pipeline FFT/IFFT processor for OFDM applications. *IEEE transactions on consumer electronics*, 57(1), 2011.

[117] Jason Thong and Nicola Nicolici. An optimal and practical approach to single constant multiplication. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 30(9):1373–1386, 2011.

[118] Rui Guo and Linda S DeBrunner. A novel fast canonical-signed-digit conversion technique for multiplication. In *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 1637–1640. IEEE, 2011.

[119] Andrew G Dempster and Malcolm D Macleod. Multiplication by two integers using the minimum number of adders. In *IEEE International Symposium on Circuits and Systems*, pages 1814–1817. IEEE, 2005.

[120] Fahad Qureshi and Oscar Gustafsson. Low-complexity reconfigurable complex constant multiplication for ffts. In *IEEE International Symposium on Circuits and Systems*, pages 1137–1140. IEEE, 2009.

[121] Koushik Maharatna, Eckhard Grass, and Ulrich Jagdhold. A 64-point Fourier transform chip for high-speed wireless LAN application using OFDM. *IEEE Journal of Solid-State Circuits*, 39(3):484–493, 2004.

[122] Yunho Jung, Hongil Yoon, and Jaeseok Kim. New efficient FFT algorithm and pipeline implementation results for FDM/DMT applications. *IEEE Transactions on Consumer Electronics*, 49(1):14–20, 2003.

[123] OH Jung-Yeol and LIM Myoung-Seob. New radix-2 to the 4th power pipeline FFT processor. *IEICE transactions on electronics*, 88(8):1740–1746, 2005.

[124] Sang-In Cho and Kyu-Min Kang. A low-complexity 128-point mixed-radix FFT processor for MB-OFDM UWB systems. *ETRI journal*, 32(1):1–10, 2010.

[125] Shousheng He and Mats Torkelson. Design and implementation of a 1024-point pipeline FFT processor. In *Proceedings of the IEEE Custom Integrated Circuits Conference*, pages 131–134. IEEE, 1998.

[126] Lan-Da Van and Jin-Hao Tu. Power-efficient pipelined reconfigurable fixed-width baugh-wooley multipliers. *IEEE Transactions on Computer*, 58(10):1346–1355, 2009.

[127] Zeke Wang, Xue Liu, Bingsheng He, and Feng Yu. A combined SDC-SDF architecture for normal I/O pipelined radix-2 FFT. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 23(5):973–977, 2015.

[128] Wei Han, T Arslan, AT Erdogan, and M Hasan. Novel low power pipelined FFT based on subexpression sharing for wireless LAN applications. In *IEEE Workshop on Signal Processing Systems*, pages 83–88. IEEE, 2004.

[129] Denise C Alves, Gabriel S da Silva, Eduardo R de Lima, Cesar G Chaves, Daniel Urdaneta, Tiago Perez, and Maique Garcia. Architecture design and implementation of key components of an OFDM transceiver for IEEE 802.15. 4g. In *IEEE International Symposium on Circuits and Systems*, pages 550–553. IEEE, 2016.

[130] Md Rahat Hossain, Amanullah Maung Than Oo, and ABM Shawkat Ali. Evolution of smart grid and some pertinent issues. In *20th Australasian Universities Power Engineering Conference*, pages 1–6. IEEE, 2010.

[131] V Arunachalam and Alex Noel Joseph Raj. Efficient vlsi implementation of fft for orthogonal frequency division multiplexing applications. *IET Circuits, Devices & Systems*, 8(6):526–531, 2014.

[132] Koushik Maharatna, Eckhard Grass, and Ulrich Jagdhold. A 64-point fourier transform chip for high-speed wireless lan application using ofdm. *IEEE Journal of Solid-State Circuits*, 39(3):484–493, 2004.

[133] Taesang Cho, Hanho Lee, Jounsup Park, and Chulgyun Park. A high-speed low-complexity modified radix-2^5 FFT processor for gigabit WPAN applications.

In *IEEE International Symposium on Circuits and Systems*, pages 1259–1262. IEEE, 2011.

[134] Yu-Wei Lin, Hsuan-Yu Liu, and Chen-Yi Lee. A 1-gs/s fft/ifft processor for uwb applications. *IEEE Journal of solid-state circuits*, 40(8):1726–1735, 2005.

[135] Trio Adiono, Muh Syafiq Irsyadi, Yan Syafri Hidayat, and Ade Irawan. 64-point fast efficient FFT architecture using radix-2^3 single path delay feedback. In *International Conference on Electrical Engineering and Informatics*, volume 2, pages 654–658. IEEE, 2009.

Publications Related to Thesis

[1] G. Ganesh Kumar and Subhendu K sahoo, "An area-efficient and low-power 64-point pipeline fast Fourier transform for OFDM applications," in *Integration, the VLSI Journal*, Vol 57, pp. 125-131, 2017.

[2] G. Ganesh Kumar and Subhendu K sahoo, "A High-Performance Signed/Unsigned Multiplier using Vedic Mathematics ," *International Journal of Information Technology (Springer)*, (Accepted).

[3] G. Ganesh Kumar and Subhendu K sahoo, "Power-Delay Product Minimization in High-Performance Fixed-Width Multiplier," in *International Technical Conference of IEEE Region 10 (TENCON 2015)*, pp. 1-4, IEEE, 2015.

[4] G. Ganesh Kumar and Subhendu K sahoo, "Implementation of A High Speed Multiplier for High-Performance and Low Power Applications," in 19^{th} *International Symposium on VLSI Design and Test (VDAT)*, pp. 1-4, IEEE, 2015.

[5] G. Ganesh Kumar, Pramod Kumar Meher and Subhendu K sahoo, "50 Years of FFT Algorithms and Applications," *IEEE Transactions on Circuits and Systems I: Regular Papers* , (communicated).

[6] G. Ganesh Kumar and Subhendu K sahoo, "An Area and Power-Efficient Variable-Length Fast Fourier Transform for MR-OFDM Physical Layer of IEEE 802.15.4-g," *IET Signal Processing*, (communicated).

Biography of the Candidate

G Ganesh Kumar obtained his Master of Technology in VLSI System Design from Jawaharlal Nehru Technological University (JNTU) - Anantapur, Anantapur. He has been working as a research scholar at BITS Pilani, Hyderabad Campus from 2013-2018 under the supervision of Prof. Subhendu K Sahoo. His research interests focus on VLSI arithmetic circuits and Digital Signal Processing.

Biography of the Supervisor

Subhendu Kumar Sahoo completed his B.E. in Electronics and telecommunication engineering from Utkal University, Orissa, India in the year 1994 with honors securing fifth position in the university. He obtained his M.E. in Electronic Systems and Communication from R.E.C.(NIT) Rourkela in 1998. He received the Ph. D. degree in Electrical Engineering from Birla Institute of Technology and Science, Pilani, in 2006. He was working as a faculty in Electrical and Electronics Engineering department from 2009 till 2011. Presently he is working as Associate Professor in Birla Institute of Technology and Science, Pilani Hyderabad campus. His areas of research are high performance arithmetic circuits and VLSI circuits for Digital Signal Processing application.