Linköping Studies in Science and Technology
Dissertations, No 1423

Optimization of Rotations in FFTs

Fahad Qureshi

Linköpings universitet
TEKNISKA HÖGSKOLAN

Department of Electrical Engineering
Linköping 2012

Linköping Studies in Science and Technology
Dissertations, No 1423

Fahad Qureshi
fahadq@isy.liu.se
www.es.isy.liu.se
Division of Electronics Systems
Department of Electrical Engineering
Linköping University
SE–581 83 Linköping, Sweden

Qureshi, Fahad
Optimization of Rotations in FFTs
ISBN 978-91-7519-973-3
ISSN 0345-7524

Typeset with LaTeX 2ε
Printed by LiU-Tryck, Linköping, Sweden 2012

To my loving parents and family

Abstract

The aims of this thesis are to reduce the complexity and increase the accuracy of rotations carried out in the fast Fourier transform (FFT) at algorithmic and arithmetic level. In FFT algorithms, rotations appear after every hardware stage, which are also referred to as twiddle factor multiplications.

At algorithmic level, the focus is on the development and analysis of FFT algorithms. With this goal, a new approach based on binary tree decomposition is proposed. It uses the Cooley Tukey algorithm to generate a large number of FFT algorithms. These FFT algorithms have identical butterfly operations and data flow but differ in the value of the rotations. Along with this, a technique for computing the indices of the twiddle factors based on the binary tree representation has been proposed. We have analyzed the algorithms in terms of switching activity, coefficient memory size, number of non-trivial multiplications and round-off noise. These parameters have impact on the power consumption, area, and accuracy of the architecture. Furthermore, we have analyzed some specific cases in more detail for subsets of the generated algorithms.

At arithmetic level, the focus is on the hardware implementation of the rotations. These can be implemented using a complex multiplier, the CORDIC algorithm, and constant multiplications. Architectures based on the CORDIC and constant multiplication use shift and add operations, whereas the complex multiplication generally uses four real multiplications and two adders. The sine and cosine coefficients of the rotation angles for a complex multiplier are normally stored in a memory. The implementation of the coefficient memory is analyzed and the best possible approaches are analyzed. Furthermore, a number of twiddle factor multiplication architectures based on constant multiplications is investigated and proposed. In the first approach, the number of twiddle factor coefficients is reduced by trigonometric identities. By considering the addition aware quantization method, the accuracy and adder count of the coefficients are improved. A second architecture based on scaling the rotations such that they no longer have unity gain is proposed. This results in twiddle factor multipliers with even lower complexity and/or higher accuracy compared to the first proposed architecture.

Populärvetenskaplig sammanfattning

Den diskreta Fouriertransformen (DFT) är en digital signalbehandlingsalgoritm som används i många sammanhang. Exempel är vid mätning av vilka frekvenser en signal innehåller samt vissa typer av kommunikation, t ex bredband i telefonnätet (ADSL), trådlösa nätverk (WiFi) och fjärde generationens mobiltelefonsystem (LTE).

Att räkna ut den diskreta Fouriertransformen är mycket beräkningskrävande och det har sedan en lång tid utvecklats så kallade snabba Fouriertransformer, FFT (fast Fourier transform). Även om den kände matematikern Carl Friedrich Gauss beskrev en liknande teknik i början av 1800-talet, fick de inte stor spridning förrän i mitten av 1950-talet då de föreslogs som ett sätt att beräkna en DFT effektivt på de nya datorer som började dyka upp här och var.

FFT-algoritmer baseras på det faktum att man kan dela upp en lång sekvens av data i flera kortare sekvenser och sedan räkna ut DFTer på delarna var för sig samt kombinera ihop resultaten till det slutgiltiga svaret. För att göra detta krävs att delresultaten multipliceras med ett komplext tal vars belopp är ett, vilket kallas för att delresultaten roteras.

Som ett exempel så kan en DFT av längd 16 delas upp i åtta stycken DFTer av längd två. Utdatat från dessa roteras samt skickas in i två stycken DFTer av längd åtta. DFTerna av längd åtta kan delas i mindre delar och så vidare. Dessa uppdelningar kan göras på olika sätt, men hittills har enbart ett fåtal av dessa utforskats, trots att det gått över 50 år sedan FFT-algoritmen populäriserades. Dessa olika uppdelningar leder till FFT-algoritmer som är i grunden lika, men med en viktig skillnad: rotationerna är olika svåra att realisera i hårdvara. I denna avhandling studeras detta faktum och den påverkan det har på implementeringar.

Bidragen i detta arbete kan delas in i två delar. Dels har vi räknat ut exakt hur många olika sätt man kan göra uppdelningen på och undersökt dessa från olika aspekter, dels har vi föreslagit olika effektiva sätt att realisera en delmängd av rotationerna i hårdvara.

Genom att använda grafer för att representera uppdelningarna har vi kommit fram till antalet olika möjliga uppdelningar. Dessa är många fler än det tiotal som explicits föreslagits i litteraturen tidigare och ökar fort med antalet data som används till DFTn. Vi har studerat dessa baserat på hur många rotationer som behövs. Ibland kan rotationen föenklas, i extremfallet ingen rotation. Vi har även studerat hur mycket de koefficienter man multiplicerar med ändrar sig, något som påverkar hur mycket energi som går åt för multiplikationerna, hur mycket minne som behövs för att spara alla koefficienter, samt hur mycket digitalt brus multiplikationerna genererar. Resultaten är i många fall anmärkningsvärda då det visar sig att den tidigare mest använda FFT-algoritmen på många sätt är den sämsta, samt att man genom att använda andra vanliga algoritmer på “fel” hårdvara kan minska vissa kostnadsmått markant. Däremot så motverkar flera av kostnaderna till viss del varandra, så det finns ingen algoritm som är bäst på allt. Ii praktiken behöver man välja vilken kostnad som är

viktigast. Detta arbete ökar dock antaler möjliga lönlingar signifikant.

Vid multiplikation med en konstant så kan man förenkla hårdvaran genom att dela upp den i multiplikationer med två, som kan implementeras utan några kretsar, samt additioner och subtraktioner, likt multiplikation med tio kan göras genom att lägga till nollor för decimala tal. Till exempel kan man tänka sig en multiplikation med 15. Istället för en vanlig multiplikation kan vi först multiplicera med 16, som är två gånger två gånger två gånger två samt subtrahera med det utsprungliga talet. På så sätt kan multiplikationen utföras med en subtraktion, något som är mycket mindre kostsamt att bygga i hårdvara än en multiplikator. När man skall multiplicera med ett fåtal konstanter kan man använda liknande tekniker, men lägga till hårdvara som kan koppla om ledningarna så att just de utvalda konstanterna kan skapas. Vissa rotationer i en FFT-algoritm kan bara anta ett fåtal värden och vi har utnyttjat detta till att föreslå ett antal effektiva kretsar för multiplikation med just dessa tal. Den första baseras på att använda trigonometriska likheter för att uttrycka ett antal olika rotationskonstanter med hjälp av andra tal. Den andra metoden baseras på att beloppet på rotationen egentligen inte behöver vara ett, så länge det är samma för alla rotationer i en uppdelning. På så sätt har vi lyckats minska antalet additioner och subtraktioner ytterligare och några av resultaten bör vara optimala, vi bör därmed ha så stor noggrannhet som går att uppnå givet ett visst antal additioner och subtraktioner.

Preface

This dissertation thesis presents my research from December 2007 to December 2011 at the Division of Electronics Systems, Department of Electrical Engineering, Linkpöing University, Sweden. The following papers have been included in this thesis.

Paper A

The analysis of different approaches to store the twiddle factor coefficients of radix-2^i FFT algorithms for an SDF pipelined FFT architecture is discussed. The comparative analysis of different approaches is based on complexity when implemented on both FPGAs and ASICs.

- ⋆ **F. Qureshi** and O. Gustafsson, "Analysis of twiddle factor memory complexity of radix-2^i pipelined FFTs," in *Proc. Asilomar Conf. Signals Syst. Comp.*, Pacific Grove, CA, Nov. 1–4, 2009, pp. 217–220.

Paper B

Radix-2^2 equivalent FFT algorithms based on binary tree decomposition are proposed. In this work, we analyze the difference among these algorithms in terms of switching activity of twiddle factor multiplication, which is related to power consumption of the circuit. Experimental results show the that equivalent algorithms can lead to saving of 40% in switching activity.

- ⋆ **F. Qureshi** and O. Gustafsson, "Twiddle factor memory switching activity analysis of radix-2^2 and equivalent FFT algorithms," in *Proc. IEEE Int. Symp. Circuits Syst.*, Paris, France, May. 30–Jun. 2, 2010, pp. 4145–4148.

Paper C

Based on the binary tree decomposition, the radix-2 FFT algorithms for a 4k-point FFT are proposed. Those have been compared in terms of complexity and switching activity of twiddle factor multiplication. The results show the trade off between the complexity and switching activity.

- ⋆ **F. Qureshi**, S. A. Alam, and O. Gustafsson, "4k-point FFT algorithms based on optimized twiddle factor multiplication for FPGAs," in *Proc. IEEE Asia Pacific Postgraduate Research Microelectron. Electron.*, Shanghai, China, Sept. 22–24, 2010, pp. 225–228.

Paper D

A systematic method for generation of possible number of FFT algorithms based on binary tree decomposition is proposed. This method generates radix-2

FFTs algorithms for computation of 2^n-point FFTs. The difference among these FFT algorithms are in terms of twiddle factor multiplication properties, such as switching activity, size of coefficient memories and round-off noise, which are related to the power consumption, area and accuracy of the circuit. Experimental results show the importance of proper selection of algorithm, since algorithms can achieve very high savings in all properties of twiddle factor multiplication.

- ⋆ **F. Qureshi**, M. Garrido, and O. Gustafsson, "Generation of all radix-2 fast Fourier transform algorithms using binary trees and its analysis," manuscript.

Preliminary version of the above work can be found in:

- ⋆ **F. Qureshi** and O. Gustafsson, "Generation of all radix-2 fast Fourier transform algorithms using binary trees," in *Proc. European Conf. Circuit Theory Design.*, Linköping, Sweden, Aug. 29–31, 2011, pp. 677–680.

Paper E

A process denoted as addition aware quantization is proposed. This process can determine multiplication coefficients that have as low complexity as the rounded value but smaller approximation error by searching among coefficients with a longer word length or lower complexity but same approximation error.

- ⋆ O. Gustafsson and **F. Qureshi**, "Addition aware quantization for low complexity and high precision constant multiplication," *IEEE Signal Process. Lett.*, vol. 17, no. 2, pp. 173–176, Feb. 2010.

Paper F

A novel low-complexity structure for computation of twiddle factor in FFT based on shift-and-add-multiplication is presented. This structure computes the twiddle factor multiplication with a resolution of 32 points. In addition, revisits and improves the complexity of a twiddle factor with a resolution of 16 points and for completeness, calculates the coefficients and the complexity of a twiddle factor with a resolution of 8 points. Furthermore, finite word length analysis is performed for all coefficients. The round-off errors and the coefficients optimized by the addition aware quantization for varying requirements have been derived.

- ⋆ **F. Qureshi** and O. Gustafsson, "Low-complexity constant multiplication based on trigonometric identities with applications to FFTs," *IEICE Trans. Fundamentals*, vol. E94-A, no. 11, pp. 2361–2368, Nov. 2011.

Preliminary versions of the above work can be found in:

- ⋆ **F. Qureshi** and O. Gustafsson, "Low-complexity reconfigurable complex constant multiplication for FFTs," in *Proc. IEEE Int. Symp. Circuits Syst.*, Taipei, Taiwan, May 24–27, 2009, pp. 1137–1140.

- ⋆ **F. Qureshi** and O. Gustafsson, "Error analysis of low-complexity reconfigurable complex constant multiplication for FFTs," in *Proc. Swedish System-on-Chip Conf.*, Rusthållargården, Arlid, Sweden, May 4–5, 2009.

Paper G

A low-complexity realization of constant angle rotators based on shifts, adders and subtracters have been realized. The results show that redundant CORDIC and scaled constant multiplication are providing the best results depending on which angle is considered. It is also shown that the precision can vary several bits using the same number of adders and subtractors, and hence the correct choice of rotator architecture is crucial for a low-complexity realization.

- ⋆ **F. Qureshi**, M. Garrido, and O. Gustafsson, "Alternatives for low complexity complex rotators," in *Proc. IEEE Int. Conf. Electron. Circuits Syst.*, Athens, Greece, Dec. 12–15. 2010, pp. 17–20.

Paper H

A novel approach for obtaining the low-adder rotators is presented. The approach is based on scaling the complex coefficients that define the rotations. As a result, this can be used to derive rotators that are efficient both in area and performance.

- ⋆ M. Garrido, **F. Qureshi**, and O. Gustafsson, "Low-adder rotators based on coefficient scaling," manuscript.

Paper I

A unified architecture is proposed to compute 2,3,4,5 and 7 point DFT, which is based on the Winograd Fourier transform algorithm (WFTA). The complexity of the unified architecture is equal to the complexity of 7-point DFT plus six multiplexers.

- ⋆ **F. Qureshi**, M. Garrido, and O. Gustafsson, "Unified architecture based on Winograd Fourier transform algorithm for 2, 3, 4, 5, and 7 small point DFTs," manuscript.

Contributions are also made in the following publications but the contents are not directly relevant or less relevant to the topic of thesis.

- ⋆ M. Abbas, **F. Qureshi**, Z. Sheikh, O. Gustafsson, H. Johansson, and K. Johansson, "Comparison of multiplierless implementation of nonlinear-phase versus linear-phase FIR filters," in *Proc. Asilomar Conf. Signals Syst. Comp.*, Pacific Grove, CA, Oct. 26–29, 2008, pp. 598–601.

- ⋆ S. Athar, O. Gustafsson, **F. Qureshi**, and I. Kale, "On the efficient computation of single-bit input word length pipelined FFTs," *IEICE Electronics Express,* vol. 8, no. 17, pp. 1437–1443, 2011.

Acknowledgments

All praise and gratitude to almighty ALLAH the most gracious and the most merciful who gave courage and strength and made it possible for me to write down the thesis as a result of four years of continuous effort and research.

Apart from the long term commitment and consistent endeavors, there are people whom I owe a lot. Their kind guidance, subjective information and moral encouragement helped me a lot in the accomplishment of this task. They certainly deserve my warmest gratitude for their contributions and support throughout my thesis.

- ⋆ My supervisor Dr. Oscar Gustafsson for his continuous guidance, patience and selfless helping attitude throughout these four years. His sage advice, insightful criticism, and patient encouragement aided the writing of this thesis in innumerable ways. He made me realize what I am capable of, gave me his valuable time and understood my ideas even I could not convey them.

- ⋆ My co-supervisor Dr. Mario Garrido who played a very positive role in my learning and understanding during my PhD research work.

- ⋆ Prof. Håkan Johansson who first accepted me as a PhD student and provided me a great opportunity and higher seat of learning in a very competitive and motivational research environment at Electronics Systems Division.

- ⋆ Higher Education Commission (HEC) of Pakistan is gratefully acknowledged for its financial support. I am thankful to the Swedish Institute (SI) for coordinating the scholarship program. Linköping University is also deeply appreciated for the overall support.

- ⋆ Dr. Amir Eghbali and Dr. Anton Blad for their help with Latex problems.

- ⋆ Dr. Muhammad Abbas and Zaka Ullah for their valuable suggestions and discussions. I always received their help and cooperation at times when needed.

- ⋆ Peter Johansson for helping me with all administrative issues.

- ⋆ All previous and present members of the Electronics Systems for their moral support and creating such a friendly environment in the group.

- ⋆ My friends Dr. Rashid Ali and Dr. Rashad Ramzan for their extra ordinary help and guidance at the start of my PhD study.

- ⋆ My friends Imran Qureshi, Syed Asad Alam and Muhammad Irfan Kazim for their kind help in proof reading my thesis.

- ⋆ My friends and colleagues in Pakistan for their help and care they have provided during my career, especially Imtiaz Rabbani, Sabz Ali, Muhammad Harris Bhutto, Abdul Jabbar Jamali, and Asif Hussain Memon.
- ⋆ All my friends who provided me support and encouragement during my stay in Sweden, especially Dr. Rizwan Asghar, Dr. Naveed Ahsan, Dr. Ihsan ullah Kashif, Ali Saeed, Saima Athar, Nadeem Afzal, Muhammad Tauqir Pasha, Syed Ahmed Aamir, Fahad Qazi, Zafar Iqbal, Muhammad Saifullah Khan, Dr. Jawad ul Hasan, Muhammad Junaid, Muhammad Farooq, Jawad Ahmed, Muhammad Fahim Ul Haque and many more.
- ⋆ All my family members including my uncles, aunties and cousins for their prays and support during my career.
- ⋆ My younger brothers Fawad Qureshi and Asadullah Qureshi and sister Zuha Qureshi for their support in taking care of the responsibilities at my native home during my absence.
- ⋆ My in-laws for their prays and encouragement throughout these four years.
- ⋆ Special thanks to my mother and my father for their non-stop prays, encouragement, which is definitely part of the reason that I have come this far in life.
- ⋆ Finally, my life-partner Sumera Fahad for her devotion, patience, and unconditional support. Certainly, this work would have never been possible without her. My son Arhm, for taking away all the day-time worries with his innocent acts.

Contents

Chapter 1

Introduction

The discrete Fourier transform (DFT) is an important signal processing block in various applications, such as communication systems, patient monitoring and speech, signal and image processing. The efficient implementation of DFT is fundamental in many cost and hardware constraint applications. Various methods for efficiently computing the DFT have been the subject of a large number of published literature. These methods are commonly referred to as fast Fourier transform (FFT) algorithms.

In this chapter, the basic concepts of FFT algorithms is presented. This chapter further discusses the different applications of DFT and the importance of rotations in transforms.

1.1 Basic Concept of FFT Algorithms

FFT algorithms are used to compute DFTs efficiently. These algorithms are not based on any approximation, so they have the same results as the direct computation of the DFT. The DFT is defined as:

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

where N represent the size of the DFT, n is the time index, k is the frequency index and W_N^{nk} is the twiddle factor. The twiddle factor W_N^{nk} is defined as:

$$W_N^{nk} = e^{-j2\pi nk/N} = \cos\left(2\pi nk/N\right) - j\sin\left(2\pi nk/N\right). \tag{1.2}$$

The inverse discrete Fourier transform (IDFT) for recovering the original sequence is given by:

$$x(n) = \frac{1}{N} \sum_{k=0}^{N-1} X(k) W_N^{-kn},\ n = 0, 1, \ldots, N-1, \tag{1.3}$$

To efficiently compute the DFT, a number of fast Fourier transform algorithms have been proposed to reduce the computational complexity, including the Cooley-Tukey algorithm [1], prime factor algorithm [2, 3] and Winograd algorithm [4, 5]. The approach, which unified the derivation of these algorithms is *divide and conquer* [6]. In this approach, the N-point DFT is mapped into several sub-DFTs in such a way that it satisfies following condition:

$$\sum_{i=1}^{L} Cost\ (\text{sub-DFT}_{N(i)}) + \sum Cost\ (interconnections) \leq Cost\ (\text{DFT}_N), \tag{1.4}$$

where L is the number of sub-DFTs, Cost (DFT) is the number of operations, Cost (interconnection) is the cost of twiddle factor multiplications and index mapping and DFT_N is the N-point DFT. The power of divide and conquer approach is, to apply decomposition recursively, until the sub-DFTs are sufficiently small. This results in reduction of the order of complexity of FFT algorithm. This approach can also be applied directly to convolution algorithms to break it down into multiple small convolutions [2].

1.2 Applications of DFT

The discrete Fourier transform finds limitless applications in many areas of signal processing. In the era of fast computing it has become increasingly important to enhance the existing FFT algorithms to meet the ever increasing applications in the field of digital signal processing.

DFTs have also been extensively used in multi-carrier transmission systems like orthogonal frequency domain multiplexing (OFDM) as FFT processors. In multi-carrier modulation, such as OFDM and discrete multitone (DMT), data symbols are transmitted in parallel on multiple sub-carriers. Multi-carrier modulation-based transceivers involve real-time DFT computations [7]. FFT-based channel estimation method is derived from the maximum likelihood criterion, which is originally proposed for OFDM systems with pilot preambles. In order to save bandwidth and improve system performance, decision-feedback (DF) data symbols are usually exploited to track channel variations in subsequent OFDM data symbols, and this method is called DF DFT-based channel estimation [8].

DFT is extensively used in sonar and radar systems. These systems use millions of multiplications per second. Seismic processing require extensive processing of seismic data [9]. Computerized tomography is widely used to synthetically form images of internal organs of the human body wherein massive amounts of signal processing is required. Remote sensing is another field employing huge amount of processing. Satellite photographs can be processed digitally to merge several images or enhance features or combine information received on different wavelengths.

1.3 Rotation and Its Importance

A rotation is a multiplication by a complex number whose magnitude is equal to one. This operation must be carried out in many signal processing algorithms, including transforms such as the fast Fourier transform (FFT) [10], the fast discrete cosine transform (DCT) [11], and digital filters such as filter banks, lattice IIR filters [12] and lattice FIR filters [13]. The rotation is often the most expensive arithmetic operation and one of the dominating factor in determining the performances in terms of accuracy, speed and power consumption. Therefore, the number of rotators and their implementation is always an important issue in signal processing algorithms.

Transforms usually consist of a set of additions and rotations. A number of FFT algorithms have been proposed, such as radix-2 [1], radix-2^2 [14], radix-2^3 [15], radix-2^k [16], radix-r^k [17], split radix [2], prime factor algorithms [18], Winograd Fourier transform algorithms [10], and FFT algorithms generated by binary trees [19]. Their aim is to reduce the computational hardware resources such as rotators and adders. However, the main focus of most algorithms is to reduce the number of rotators and also their hardware complexity.

The implementation of rotations has a large impact on the complexity, accuracy, and power consumption of the architecture as well. A rotation can be implemented by a number of different architectures, including general complex multiplier [20], CORDIC algorithm [21, 22] and algorithms based on constant multiplication [23–26]. In all these implementations, there is a tradeoff between the accuracy and the complexity of the rotator. In transforms the scenario of the rotation is different, which has large impact on the selection of architecture. Sometimes rotations are independent, like in lattice FIR filters [13] or some special DCT architectures. Conversely, in common DCT and FFT architectures, the number of rotations are together at each stage. The accuracy in these algorithms can be increased by scaling the coefficients of rotation without any additional hardware cost [27]. The scaling can be applied to both scenarios of the rotations.

Chapter 2

FFT Algorithms

This chapter starts with an explanation of the most popular FFT algorithm, called the Cooley-Tukey FFT algorithm. The rest of the chapter focuses on FFT algorithms, including Winograd Fourier transform algorithm (WFTA), FFT algorithms based on convolution and recursive algorithms.

2.1 Cooley-Tukey FFT Algorithm

The Cooley-Tukey FFT algorithm is the most commonly used algorithm to compute the DFT, as it provides a systematic solution with a moderate computational complexity [1]. It reduces the computational complexity from $O(N^2)$ to $O(N \log N)$ by making efficient use of symmetry and periodic properties of the twiddle factors, which are following:

$$W_N^{k+N} = W_N^k, \tag{2.1}$$

$$W_N^{k+N/2} = -W_N^k. \tag{2.2}$$

These properties are used to eliminate the redundant complex multiplication. To derive the general expression of the Cooley-Tukey FFT algorithm, the N-point DFT is decomposed into its two product factors, that is:

$$N = PQ, \tag{2.3}$$

where P and Q are integers. The input and output data sequences are mapped into two dimensions: $x(n)$ into $x(n_1, n_2)$ and $X(n)$ into $X(k_1, k_2)$. Thus, the input and output data sequences are obtained by:

$$n = n_1 + Qn_2 \begin{cases} n_1 = 0, 1, 2, ...Q - 1, \\ n_2 = 0, 1, 2, ...P - 1. \end{cases} \tag{2.4}$$

$$k = Pk_1 + k_2 \begin{cases} k_1 = 0, 1, 2, ...Q-1, \\ k_2 = 0, 1, 2, ...P-1. \end{cases} \tag{2.5}$$

Using (2.4) and (2.5), the DFT (2) can be written as:

$$X(k) = \sum_{n_1=0}^{P-1} \sum_{n_2=0}^{Q-1} x(n) W_N^{(n_1+Pn_2)(Qk_1+k_2)}. \tag{2.6}$$

Further development considering that $W_N^{n_1 Q k_1} = W_P^{n_1 k_1}$, $W_N^{n_2 P k_2} = W_Q^{n_2 k_2}$ and $W_N^{P n_2 Q k_1} = 1$ leads to:

$$\begin{aligned} X(Qk_1 + k_2) = & \sum_{n_1=0}^{P-1} \sum_{n_2=0}^{Q-1} x\left(n_1 + Pn_2\right) W_P^{n_1 k_1} W_N^{n_1 k_2} W_Q^{n_2 k_2}, \\ = & \underbrace{\sum_{n_1=0}^{P-1} \left[\underbrace{\left(\sum_{n_2=0}^{Q-1} x(n_1 + Pn_2) W_Q^{n_2 k_2} \right)}_{Q-point\ DFT} \underbrace{W_N^{n_1 k_2}}_{Twiddle factor} \right] W_P^{n_1 k_1}}_{P-point\ DFT}, \end{aligned} \tag{2.7}$$

where the two summations are indexed by n_1 and n_2, which are referred to as inner and outer DFTs. As a result, an N-point DFT is broken down into P-point and Q-point DFTs. The output of the inner DFT is multiplied by $W_N^{n_1 k_2}$, which is called twiddle factor multiplication. The scheme of decomposition is shown in Fig. 2.1, where the left and right sides represent the P-point inner DFT and Q-point outer DFT, respectively. Between those DFTs, a twiddle factor multiplication indicates a rotation by $W_N^{\phi} = e^{-j\frac{2\pi}{N}\phi}$.

The binary tree representation is an effective way of understanding the difference in various FFT algorithms [28]. In binary tree representation, a node with a value n represents a 2^n-point DFT. Each node has at most two leaves which represent the inner and outer DFTs. An example of a binary tree is shown in Fig. 2.1 corresponding to the scheme of DFT. The left leaf corresponds to number of 2^q DFTs of 2^p-point whereas the right leaf represents set of 2^p DFTs of 2^q-point. This decomposition applies recursively until the leaf node matches the radix. When the binary tree is finalized, all leaf nodes correspond to the radix and internal nodes correspond to twiddle factor multiplication stages, which connect the two decomposed DFTs.

2.1.1 Classic FFT Algorithms

In this section, classic FFT algorithms are discussed. Firstly, the Cooley-Tukey algorithm is based on decomposing the N point DFT in $s = \log_r N$ stages,

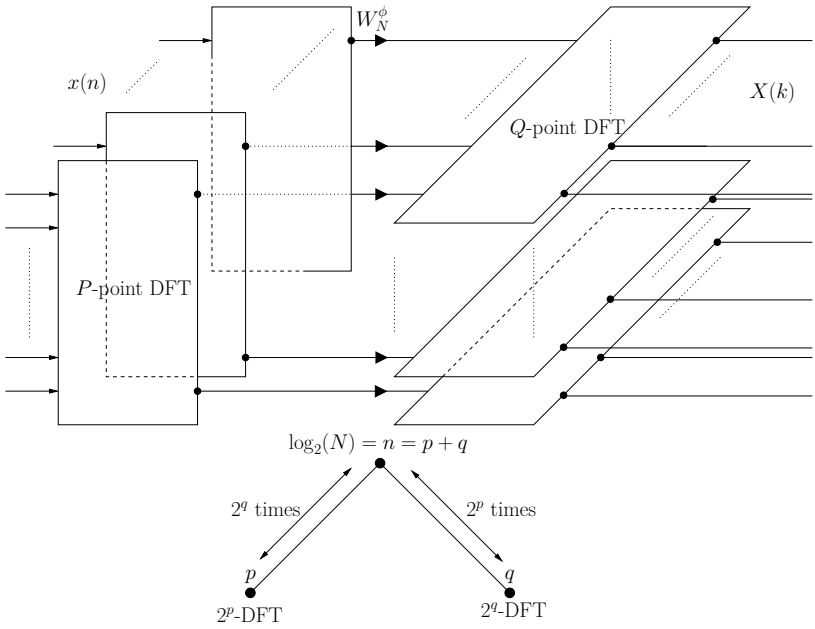

Figure 2.1: Decomposition Scheme and binary tree of Cooley-Tukey FFT algorithm.

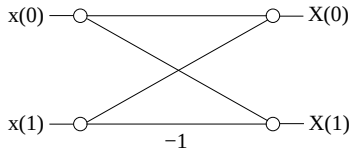

Figure 2.2: Flow graph of radix-2 butterfly.

where r is the radix of the FFT and N is a multiple of r. The decomposition splits an N point DFT into two arbitrary factors and is continued recursively until all the remaining transforms sizes are equal to r. Each radix-r FFT is called butterfly. It has r inputs and r outputs and calculates an r-point FFT. The most popular radices are 2 and 4, which are suited for FFT sizes that are multiple of 2 and 4, respectively. Figure 2.2 represents a radix-2 butterfly, which calculates:

$$\begin{aligned} X(0) &= x(0) + x(1), \\ X(1) &= x(0) - x(1). \end{aligned} \tag{2.8}$$

Figure 2.3 shows a radix-4 butterfly. Note that it requires a complex rotation by $e^{-j\pi/2}$. This complex rotation is considered a trivial rotation as it can be

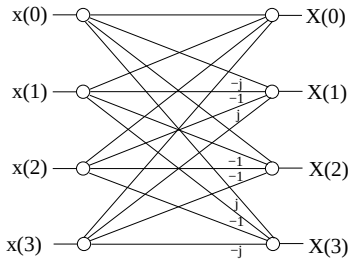

Figure 2.3: Flow graph of radix-4 butterfly.

realized by interchanging the real and imaginary parts and/or changing the sign of the input data. Rotations by $1, -1, j, -j$ are also trivial. Furthermore, higher radices generally reduces the number of stages at the cost of increasing the complexity of each stage because radices higher than 4 require butterflies with non trivial rotations. Non trivial rotation can be implemented using a general complex multiplier, the CORDIC algorithm or constant multiplications. As a result, the selection of the radix has a significant impact on the complexity of the selected FFT algorithm.

Figures 2.4(a), (b) and (c) show the binary trees of radix-2 DIF/DIT and radix-4 DIF FFT algorithms respectively. The node number represents the twiddle factor multiplication between the two decomposed stages and the index ϕ of twiddle factor can be generated from the counter in the range $[0, N-1]$ [19]. If compared, the radix-4 FFT algorithm has half the stages as compared to radix-2 FFT algorithm. Likewise, the number of stages can be reduced by using a higher-radix FFT algorithm. Finally, the most common decompositions, namely decimation in time (DIT) and decimation in frequency (DIF) are discussed. For an N-point DFT, a decomposition using decimation in time separates the input sequence into its even- and odd-indexed samples. This breaks down the N-point DFT into two $N/2$-point DFTs plus some additions and rotations. Then, the decomposition proceeds iteratively on the $N/2$-point DFTs until the whole algorithm has been simplified. Likewise, the decimation in frequency decomposes the DFT iteratively into DFTs of half size. Unlike the DIT, DIF starts from the output frequencies by separating them into even- and odd-indexed ones. Figures 2.4(a) and (b) represent the binary trees of a radix-2 DIF and DIT FFT, respectively. By comparing both figures, it can be observed that the DIF and DIT decompositions of the FFT only differ in the location of the twiddle factor multiplications.

2.1.2 Improved FFT Algorithms

Improved FFT algorithms are basically the modified versions of the classic FFT algorithms. As discussed in classical algorithms, a selection of radix has a large impact on the complexity of FFT algorithm. Higher radices help in reducing

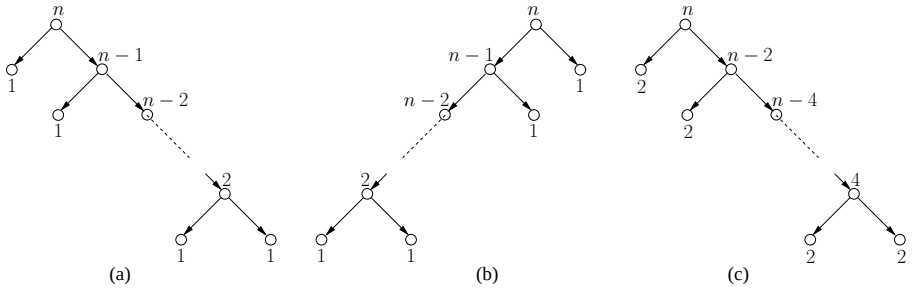

Figure 2.4: Binary trees of classical FFT algorithms: (a) Radix-2 DIF, (b) Radix-2 DIT, and (c) Radix-4 DIF.

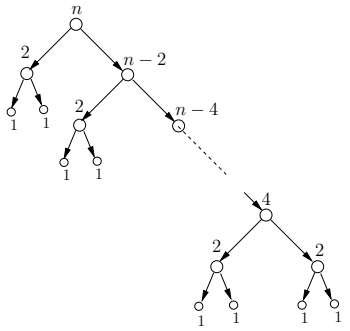

Figure 2.5: Binary tree of radix-2^2 FFT algorithm.

the number of stages. However, complexity of the butterflies increases [10]. The radix-2^2 FFT algorithm is one of most popular FFT algorithms. In radix-2^2 the number of non trivial multiplications is exactly the same as the radix-4. The binary tree diagram of radix-2^2 is depicted in Fig. 2.5, which shows that the alternate twiddle factor multiplication stages have the W_4 twiddle factor multiplication. The W_4 twiddle factor multiplication involves trivial multiplications.

Furthermore, radix-2^3 and radix-2^4 can be used to implement the radix-8 and radix-16 butterflies by radix-2 [23]. The radix-2^3 and radix-2^4 FFT algorithms have same twiddle factor stages as the radix-8 and radix-16 butterfly FFT algorithm. The binary tree of the radix-2^3 with radix-8 butterfly FFT algorithms and radix-2^4 with radix-16 FFT algorithms are shown in Fig. 2.6. The radix-2^4 can further be modified to implement the radix-16 by first splitting the radix-16 into two radix-4 and further decompose into radix-2. In this way, can remove the W_8 twiddle factor multiplication stage as shown in Fig. 2.7 (a) [23]. The twiddle factor multiplication stages of radix-2,radix-2^2, radix-2^3, radix-2^4, and modified radix-2^4 FFT algorithms are tabulated in Table 2.1. This table includes all the twiddle factor multiplication stages use to implement the FFT

Table 2.1: Multiplication at different stages for improved FFT algorithms.

	Stage number						
Radix	1	2	3	4	5	.....	s
2	W_N	$W_{N/2}$	$W_{N/4}$	$W_{N/8}$	$W_{N/16}$	.....	W_4
2^2	W_4	W_N	W_4	$W_{N/4}$	W_4	.....	W_4
2^3	W_4	W_8	W_N	W_4	W_8	.....	W_8
2^4	W_4	W_8	W_{16}	W_N	W_4	.....	W_{16}

algorithm. In addition, there are many other FFT algorithms, such as, based of balance binary tree [28], radix-2^k [16] and radix-r^k [17]. However, there are still hundreds on the FFT algorithms that can be generated. In paper D, authors proposed a method to generate the number of possible FFT algorithms. The difference among these FFT algorithms analyzed in terms of twiddle factor multiplication properties, such as switching activity, size of coefficient memories and round-off noise, which are related to the power consumption, area and accuracy of the circuit.

2.1.3 Split-Radix FFT Algorithm

Split-radix FFT algorithms for 2^n size DFT were introduced, which is famous for having the less number of multiplications and additions [2]. This algorithm provides optimum number of operations for power of two sized DFT. The main idea outlining split-radix FFT algorithm is that different decompositions can be used for different parts of the algorithm. Thus, independent parts of the algorithm should be computed independently and should use the best possible computational scheme, regardless of what scheme is used for other parts of the algorithm. Thus, reduction in computational complexity is achieved by using this approach [3].

As an example, let us consider a split radix FFT algorithm for 2^n size DFT. The radix-2 decomposition on the even indexed outputs of the DFT while using a radix-4 decomposition on the odd indexed parts. This approach thus combines the advantages of both radix-2 and radix-4 algorithm by minimizing the number of non trivial twiddle factors at any one stage while minimizing the number of such stages. When split radix algorithm is applied to radix-2 and radix-4 simultaneously on different parts of an N-point FFT, it results in an L-shaped butterfly, which is shown in Fig. 2.8 However when iterative decomposition is applied, unlike the uni-radix FFT algorithms, the split radix does not progress stage by stage thus resulting in a less regular FFT algorithm than the uni-radix [16] or the mixed-radix FFT algorithms [29].

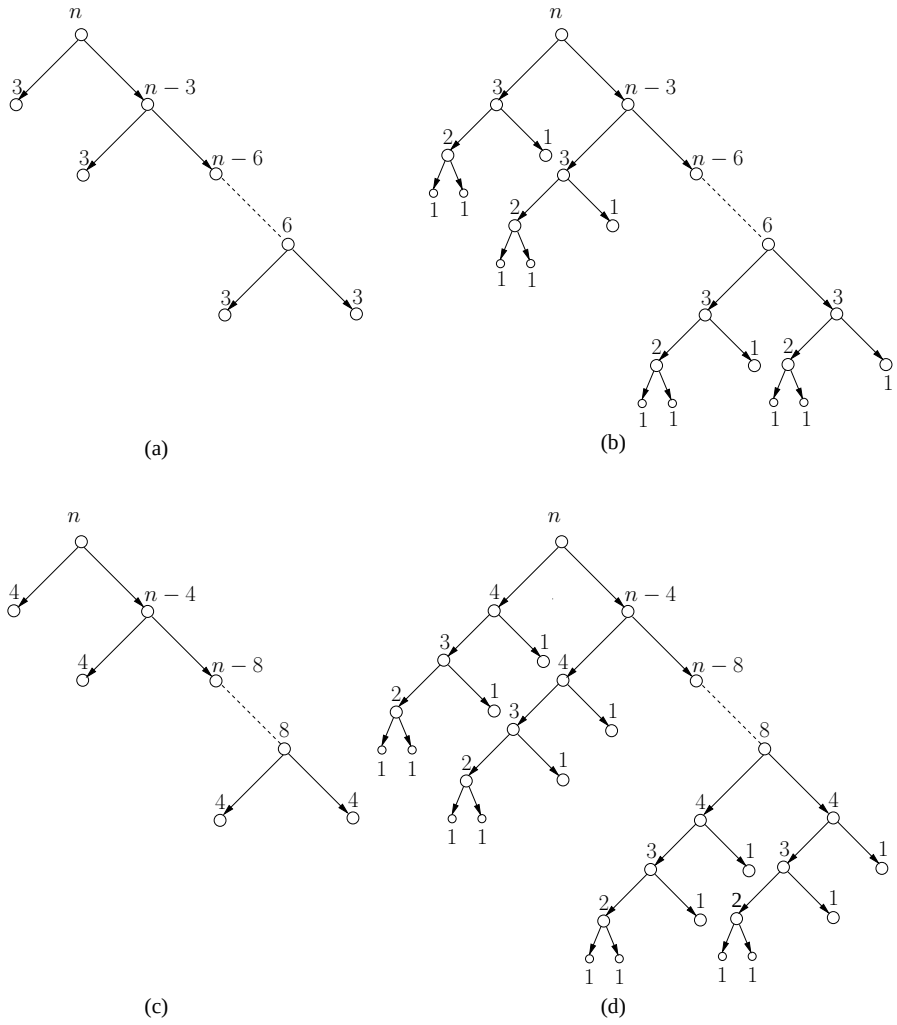

Figure 2.6: Binary tree of FFT algorithms: (a) Radix-8, (b) Radix-2^3, (c) Radix-16, and (d) Radix-2^4.

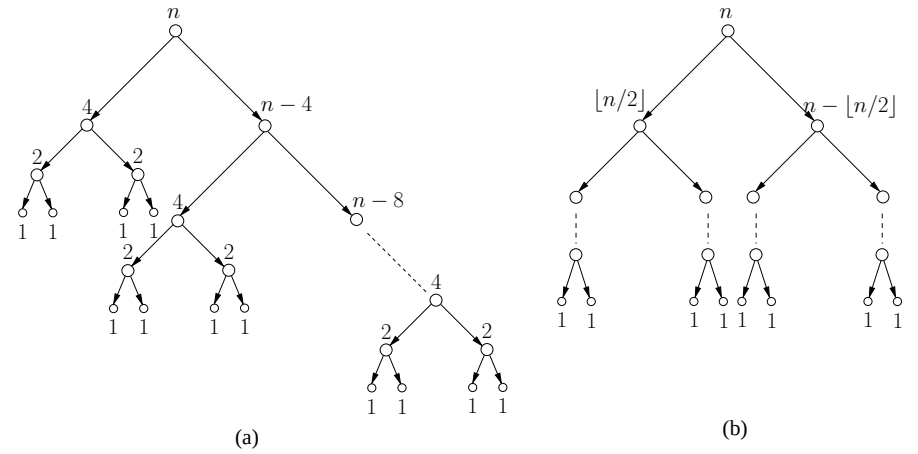

Figure 2.7: Binary tree of FFT algorithms: (a) Modified radix-2^4 and (b) Balance binary tree decomposition.

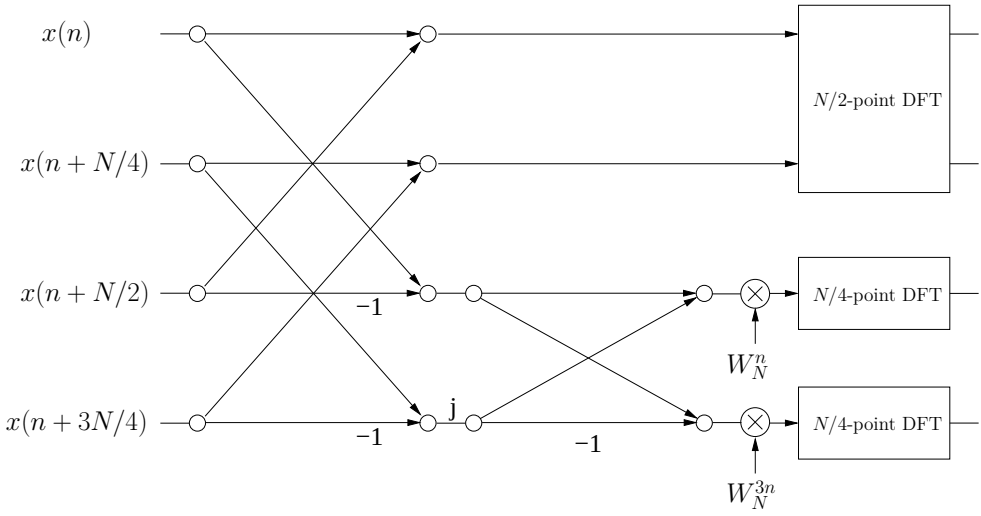

Figure 2.8: Split radix-2/4 butterfly.

2.1.4 Prime Factor Algorithm

The elimination of the twiddle factor multiplication is obtained by using the prime factor algorithm for computation of the FFT, also called Good Thomas algorithm [18]. The signal flow graph of prime factor FFT algorithm decomposition is shown in Fig. 2.9, where there is no multiplication between two decomposed sub-DFTs. This algorithm uses a similar divide and conquer approach, where the sizes of the decomposed DFTs are relative prime. Two numbers are relatively prime or coprime, if their common divisor is one. In such cases a special index mapping based on the Chinese remainder theorem is formed to connect these decomposed DFTs. In this case, the input $x(n)$ and data sequence $X(k)$ can be substituted by the following:

$$n = \langle An_1 + Bn_2 \rangle_N \begin{cases} n_1 = 0, 1, 2, ...Q-1, \\ n_2 = 0, 1, 2, ...P-1. \end{cases} \tag{2.9}$$

$$k = \langle Ck_1 + Dk_2 \rangle_N \begin{cases} k_1 = 0, 1, 2, ...Q-1, \\ k_2 = 0, 1, 2, ...P-1, \end{cases} \tag{2.10}$$

where $\langle f \rangle_N$ denotes the f modulo N. The coefficients A, B, C and D have to be chosen so that the following conditions are satisfied:

$$\langle A \rangle_N = Q, \qquad \langle BD \rangle_N = P \qquad \langle AD \rangle_N = \langle BC \rangle_N = 0, \tag{2.11}$$

and further development considering these conditions lead to:

$$W_N^{(\langle An_1+Bn_2\rangle_N \cdot \langle Ck_1+Dk_2\rangle_N)} = W_P^{k_1 n_1} W_Q^{k_2 n_2}, \tag{2.12}$$

thus twiddle factor multiplication is not required. The method to find the required values is not simple, it requires *Chinese remainder theorem*. One of the possible set of coefficients that satisfy these specific conditions is following:

$$A = Q, \qquad B = P, \qquad C = Q\langle Q^{-1} \rangle_P, \qquad C = P\langle P^{-1} \rangle_Q, \tag{2.13}$$

where $\langle Q^{-1} \rangle_P$ denotes the multiplicative inverse of Q modulo P, i.e., if $\langle Q^{-1} \rangle_P = \eta$ then $\langle Q\eta \rangle_P = 1$. Note that many solutions of these conditions may exist for specific value of P and Q.

By applying this new index mapping the DFT (2) becomes

$$X(k) = \underbrace{\sum_{n_1=0}^{P-1} \left[\underbrace{\sum_{n_2=0}^{Q-1} x\langle Qn_1 + Pn_2 \rangle_N W_Q^{n_2 k_2}}_{Q-point\ DFT} \right] W_P^{n_1 k_1}}_{P-point\ DFT}, \tag{2.14}$$

where there is no twiddle factor multiplication to interconnect two decomposed DFTs as shown in Fig. 2.9. Apparently, there is no considerable disadvantage

from (2.14). However, it requires complex index mapping of the input data [10]. This algorithm can also be applied recursively until the decomposed DFT can be factorized into coprime DFTs. Furthermore, it is even possible to compute DFT by using both the Cooley-Tukey FFT and prime factor FFT algorithm.

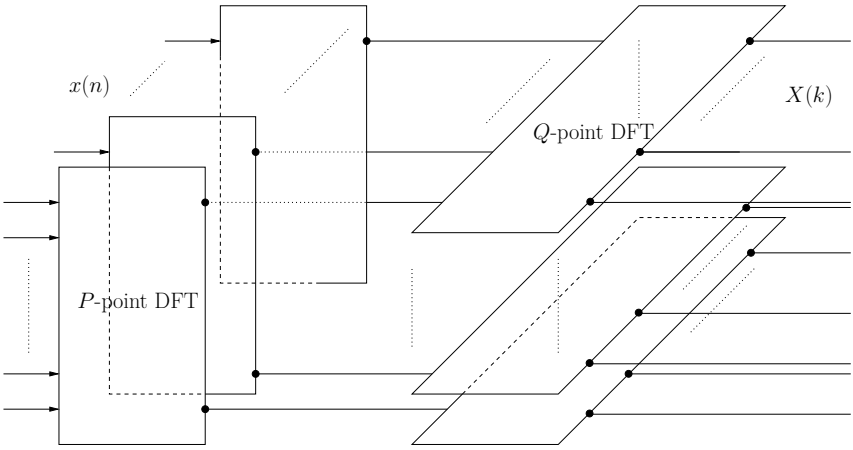

Figure 2.9: Decomposition based on prime factor FFT algorithm.

2.2 Winograd Fourier Transform Algorithm

The Winograd Fourier transform algorithm (WFTA), is an FFT algorithm which achieves a reduction on the number of multiplications from order $O(N^2)$ in the DFT to order N [4, 5, 30]. However, the hardware implementation gets complicated mainly due to complex index mapping. Furthermore, the number of additions is increased significantly as compared to other FFT algorithms. Thus, WFTA may only be efficient in implementing small size DFTs.

This FFT algorithm is also based on the decomposition of N-point DFT into sub-DFTs. Each sub-DFT is computed using Winograd algorithm, which can be defined as:

$$\begin{bmatrix} X(1) \\ . \\ . \\ . \\ . \\ X(N-1) \end{bmatrix} = IMO \times \begin{bmatrix} x(1) \\ . \\ . \\ . \\ . \\ x(N-1) \end{bmatrix}, \tag{2.15}$$

where I is a matrix, which corresponds to the addition between inputs, M is a diagonal multiplication matrix, and O is matrix, which corresponds the addition after multiplication. These operations, also called the preweave addition, the

multiplications, and the postweave additions, which are shown in Fig. 2. The blocks are carried out on the basis of Radar's permutation which converts a DFT into a cyclic convolution and then applies Winograd's cyclic convolution algorithm for computation of DFT.

Unlike the Cooley-Tukey FFT algorithm or the prime factor algorithm, the WFTA computation method does not process one sub-DFT at a time. The WFTA first computes the preweave additions of all sub-DFTs, then computes all multiplications of sub-DFTs and finally computes the postweave addition of all sub-DFTs.

Figure 2.10: Block diagram of Winograd algorithm.

2.3 Other Fourier Transforms Algorithms

More algorithms have been proposed, those are based on the circular convolution and recursive computation of DFT.

2.3.1 Bluestein Chirp Fourier Transforms

The Bluestein Chirp Fourier transform is based on the chirp z-transform algorithm [31]. In other words, it is a special case of chirp z-transform algorithm. The algorithm is more flexible and can be applied to the size of DFT. By applying the Blustein chirp algorithm, the DFT (2) becomes:

$$X(k) = W_N^{k^2/2} \sum_{n=0}^{N-1} x(n) W_N^{-\frac{(n-k)^2}{2}} W_N^{\frac{n^2}{2}}, \tag{2.16}$$

where $W_N^{k^2/2}$ and $W_N^{n^2/2}$ are the chirp signals. In order to better understand the algorithm its the block diagram is presented in Fig. 2.11. It consists of multiplication stages before and after the convolution block by chirp signals. The Bluestein Chirp Fourier transform algorithm requires $2N$ multiplications and a cyclic convolution of length N, so it is not efficient in reducing the number of operations. However, it can be suited to implement in a variety of applications [32]. Furthermore, it can be possible to use fast convolution algorithms for the cyclic convolution.

2.3.2 Rader Prime Algorithm

Rader algorithm is also based on the cyclic convolution and only used to compute the prime factor size DFTs [33]. It requires some indexing operation and

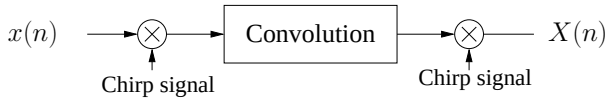

Figure 2.11: Bluestein Chirp algorithm.

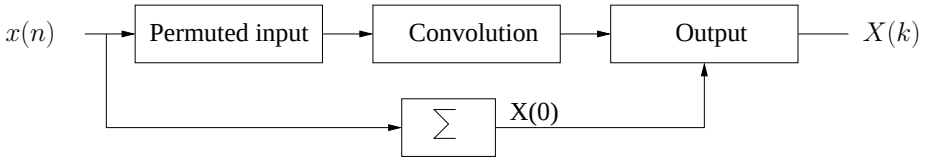

Figure 2.12: Rader prime algorithm.

the computation of a cyclic convolution. However, cyclic convolution requires $n-1$ length, which is not a prime. Thus, first covert the length from the prime length to non-prime for cyclic convolution length DFT by:

$$X(0) = \sum_{n=0}^{N-1} x(n). \tag{2.17}$$

Because the primitive element generator p generates all values of the output index k except $k(0)$. Now, replace n with p^n mod N and k by p^k mod N, and also substitutes (2.17), which gives the following equation:

$$X(\langle p^k \rangle_M) = x(0) + \sum_{n=0}^{N-1} x(\langle p^n \rangle_N) W_N^{\langle p^{n+k} \rangle_{N-1}}, \quad k = 1, ...N-1. \tag{2.18}$$

Now it is solved by applying cyclic convolution. The block diagram of the algorithm is depicted in Fig. 2.12, where the cyclic convolution block computes the output of DFT from 1 to $N-1$, $X(0)$ is calculate by simple summation, finally all outputs are combined of DFT.

2.3.3 Goertzel Algorithm

The Goertzel algorithm is a digital signal processing (DSP) technique for identifying frequency components of a signal. While the general FFT algorithm computes evenly across the bandwidth of the incoming signal, the Goertzel algorithm looks at specific, predetermined frequencies.

The Goertzel algorithm exploits the periodicity of the phase factor W_N^k and allows to express the computation of the DFT as a linear filtering operation. Since $W_N^{-kN} = 1$, this can multiply by the 2:

$$X(k) = \sum_{n=0}^{N-1} x(n) W_N^{nk} W_N^{-kN}, \tag{2.19}$$

$$= \sum_{n=0}^{N-1} x(n) W_N^{-k(N-n)}. \tag{2.20}$$

If $Y_k(n)$ is defined as,

$$Y_k(n) = \sum_{p=0}^{n} x_e(p) W_N^{-k(n-p)}. \tag{2.21}$$

Then it is clear that $y_k(n)$ is the convolution of the finite duration input sequence of $x(n)$ of length N that has an impulse response

$$h_k(n) = W_N^{-kn} u(n). \tag{2.22}$$

The system with impulse response $h_k(n)$ has the transfer function

$$H_k(z) = \frac{1}{1 - W_N^{-k} z^{-1}}. \tag{2.23}$$

Thus the entire DFT can be computed by passing the block of input data into a parallel bank of N single pole filters where each filter has a pole at the corresponding frequency of the DFT. Using the difference equation corresponding to the above equation to compute $y_k(n)$ recursively so

$$\begin{aligned} Y_k(n) &= W_N^{-k} Y_k(n-1) + x(n), \\ Y_k(-1) &= 0 \qquad\qquad . \end{aligned} \tag{2.24}$$

The complex multiplication and addition in the above equation can be avoided by combining the pairs of single pole filters possessing complex conjugate poles. This leads to two pole filter given by

$$H_k(z) = \frac{1 - W_N^k z^{-1}}{1 - 2\cos(2\pi k/N) z^{-1} + z^{-2}} \tag{2.25}$$

The flow graph of the system corresponding to a second-order Goertzel algorithm is shown in Fig. 2.13. The recursive part of this figure iterates N cycles and then pluses one complex multiply to obtain $X(k)$.

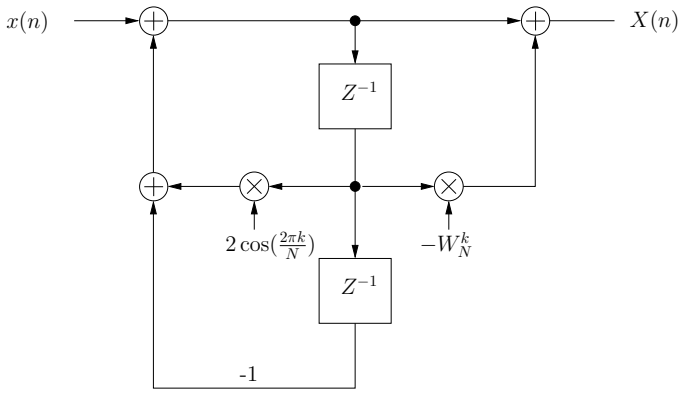

Figure 2.13: Flow graph of second-order Goertzel algorithm.

Chapter 3

FFT Architectures

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.

An FFT architecture consists of processing elements including butterflies and rotators with memory and data management circuits. Each butterfly is usually reused to compute several butterflies of the FFT algorithm. Similarly, rotators can be reused to perform several rotations. This reuse of the processing element reduces the area of circuit.

FFT algorithms can be implemented using different FFT architectures. Among them, the selection of the algorithm depends on the required specifications. Three basic specifications can be found in the literature: signal processing specifications such accuracy, application specifications such as latency and throughput, and hardware device specifications, includes area limitations and sometimes power consumption.

The most common FFT hardware architectures are explained in the next section, including direct implementation of the FFT algorithm, memory-based, and pipelined architectures.

3.1 Direct Implementation

The direct implementation is an isomorphic mapping of the signal flow graph of an FFT algorithms on to hardware. The direct implementation requires a number of processing elements equal to the number of operations. Likely, this may not be an efficient architecture for most applications, being very hardware intensive. Nevertheless, it can be suitable for small size FFTs and for extremely high

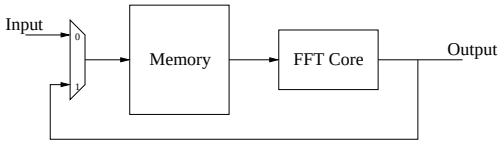

Figure 3.1: Memory based FFT.

throughputs [34]. Furthermore, the utilization of the butterflies and rotators is 100%. The hardware cost in terms of butterflies and rotators is proportional to $O(N \log N)$.

3.2 Memory-Based Architectures

Memory-based architectures are also called in-place architectures or iterative architectures. They consist of one or more processing elements that calculate all the butterflies and twiddle factor multiplication of the algorithm. The memory-based FFT architecture uses one or more processing elements so one or more memories are required to store the value. The FFT computation is done by fetching the input data from memory, processing it in the processing element and storing the result again in the memory. This process is iterated until all FFT computations are completed. This is an efficient architecture to meet stringent low area requirements for large size FFTs. [29, 35–38]

A simple memory-based architecture is shown in Fig. 3.1. It consists of a memory and an FFT core, which computes the butterfly and rotation. As seen in Fig. 3.1, after every iteration data are stored in memory so it is necessary to compute whole FFT before it receives new samples. Thus, the memory-based architecture is unable to compute the FFT when data arrives continuously at the input. Indeed, this can be solved by adding extra memory for storing the incoming data while the FFT is being calculated.

In memory-based architectures the throughput depends on the latency and the size of the FFT. The higher the computation time of the FFT core, results in higher the latency and the lower throughput. Likewise, large size FFTs require large number of iterations, which lowers the throughput. As a result, it may be unsuitable for FFT computation in real time applications.

To improve the throughput and to decrease the latency in memory-based architectures, high-radix processing elements are used. However, when memory-based and higher-radix processing element are used together, it causes memory conflict problems due to improper memory accesses. Thus multi bank memory is required in order to solve these conflicts [39]. Parallel processing is another approach to increase the throughput and decrease the latency. The column FFT architecture is an example of highest parallelism [37, 38, 40–42], where an entire stage of the FFT signal flow graph is computed in parallel, as shown in Fig. 3.2. These architectures have high throughput, but hardware cost is increased as

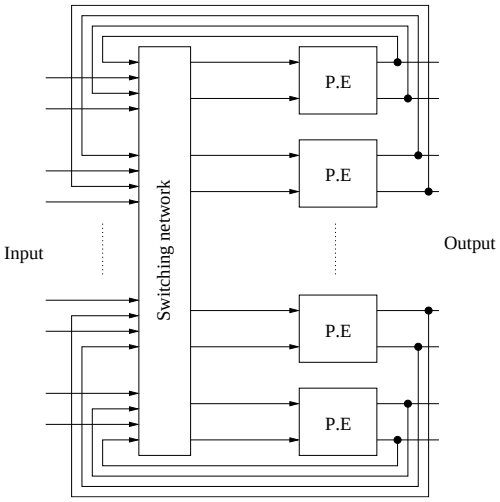

Figure 3.2: Memory based FFT.

compared to single processing element architectures.

3.3 Pipelined

Pipelined architectures [3, 14, 16, 23, 28, 43–53] are also called streaming architectures. These architectures have a regular structure, moderated area, relatively simple control and high throughput. In addition, a major advantage of pipelined architectures is that they can process a continuous flow of data without any additional circuitry as compared to memory-based architectures. Furthermore, it is possible to increase the clock frequency by adding registers in order to divide the critical path, which increases the throughput. As a result, pipelined architectures have high throughput rates and are suitable for real-time applications.

In pipelined FFT architectures, a single stage of the FFT algorithm is computed by a single stage of the architecture. Figure 3.3 shows a 16-point radix-2 FFT algorithm and its pipelined architecture. In order to start the computations, $x(n)$ and $x(n+N/2)$ should be available. For the continuous flow of data, the first half samples have to be stored in memory until the second half sample arrives. Based on memory storage, there are two main categories: delay commutator and delay feedback. In the next section, the most common pipelined FFT architectures based on these categories are discussed, including single-path delay feedback (SDF), multi-path delay commutator (MDC) and multi-path delay feedback (MDF).

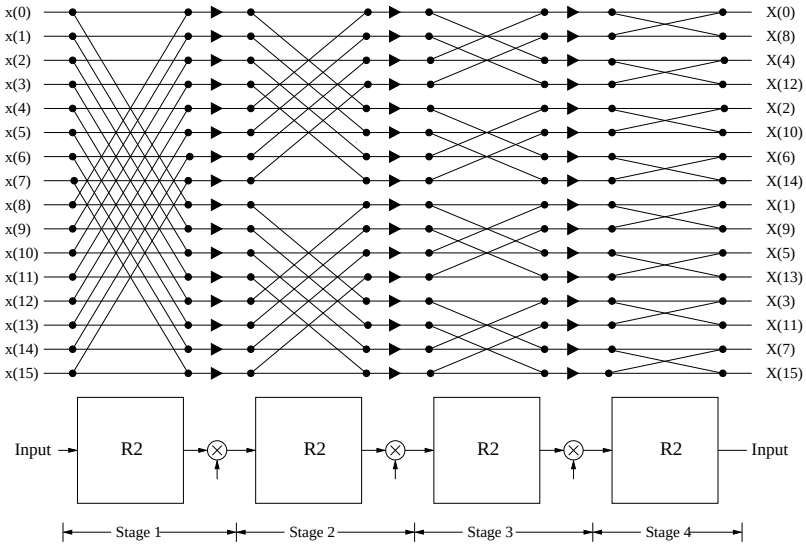

Figure 3.3: Transform from signal flow graph to pipeline FFT architecture .

3.3.1 Single-Path Delay Feedback Pipelined FFT Architecture

The single-path delay feedback architecture has a feedback loop at every stage [14, 23, 28]. The butterfly processing element stores the input samples in feedback memory, until they are required for butterfly computation. These butterflies have 50% utilization ratio. This architecture has one continuous data stream of one sample per clock cycle, which means that at every clock cycle one sample is fed and one sample is received as output. However, higher throughput can be achieved by increasing the clock frequency.

Figure 3.4 shows the radix-r butterfly SDF pipelined FFT architecture. In this architecture, the number of stages is $s = \log_r(N)$ and each stage contains a radix-r butterfly, a rotator and memory or buffers. The radix-r butterfly consist of an adder, a substractor and multiplexers. However, in case of radix higher than 4, rotators are also required for computation of non-trivial rotations. Furthermore, multiplexers are used to control the inputs of the memory whether fed directly without any computation or after the butterfly computation is performed. These multiplexers are usually controlled by a clock signal. In case of radix-2, multiplexers of first stage switch their position after $N/2$ clock cycles and every $N/4$ clock cycles in the consequent stages. Likewise, the memory requirement is only $N - 1$ because it requires N cycles to provide first output. As far as the rotators are concerned, they can be implemented using different architectures, which will be discussed in Chapter 4. In general, a single-path delay feedback architecture for FFT is considered as an optimal choice in terms of the hardware cost and performance for many applications.

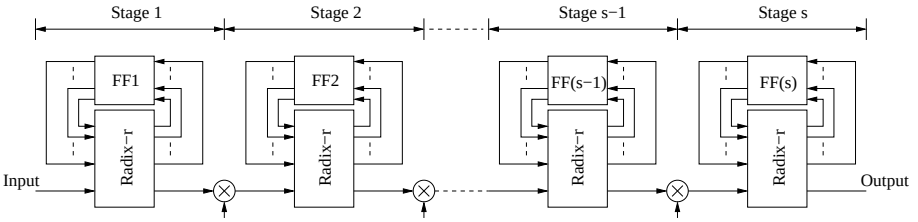

Figure 3.4: Radix-r single-path delay feedback architecture.

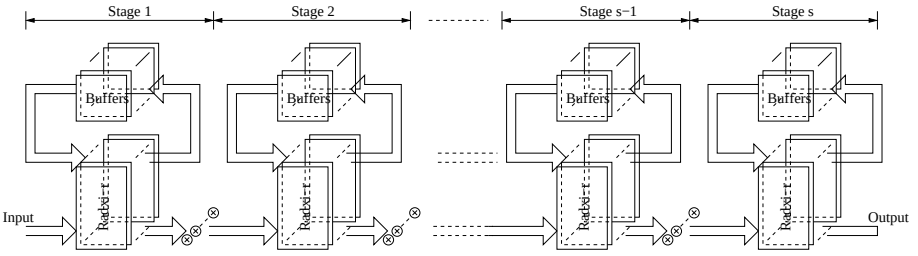

Figure 3.5: Radix-r multi-path delay feedback pipelined FFT architecture.

3.3.2 Multi-Path Delay Feedback Pipelined FFT Architecture

Multi-path delay feedback pipelined FFT architectures are also known as parallel feedback architectures. These architectures consist of parallel SDF pipelined FFT architectures for processing several samples [44, 54–56]. Like, the SDF pipelined FFT, the MDC architecture has 50% butterfly utilization ratio. Thus, the parallelism does not improve the utilization ratio. MDF architectures can process samples in parallel, so it has high throughput at the expense of hardware cost.

MDF pipelined FFT architectures can use different radices, such as radix-2 [54], radix-2^2 [56], and radix-2^4 [44]. Figure 3.5 shows the radix-r multi-path delay feedback architecture, where each path has single input and single output. Similar to other architectures, a radix-r butterfly is used to compute r-point DFTs. This figure shows the rotators in each path for twiddle factor multiplication. However, this can be minimized by using the scheduling technique and the specified constant multipliers, which are based on sharing the same rotator for multiple streams. Furthermore, a memory is used in the feedback loop, which feeds the samples to butterfly stages according to data flow requirements of the circuit.

3.3.3 Multi-Path Delay Commutator Pipelined FFT Architecture

The Multi-path delay commutator pipelined FFT architecture is the most straight forward implementation of FFT algorithms, also referred to as the feedforward

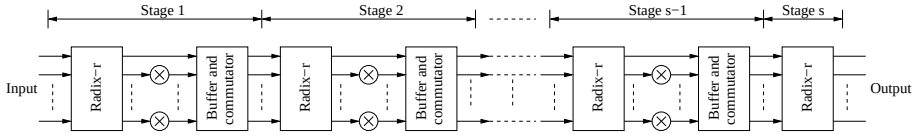

Figure 3.6: Radix-r multi-delay commutator pipelined FFT architecture.

FFT architecture [16, 57–59]. This architecture does not have any feedback loop so data is processed by the butterflies and the rotators then fed to the next stage. As a result, they have 100% utilization ratio of butterflies [16]. Generally, it can compute several samples in parallel, so they can provide higher throughput than SDF pipelined architecture at the cost of hardware.

The MDC pipelined architecture can be used for several butterflies, such as radix-2 [47], radix-4 [45] and radix-8 [45]. These architectures can be improved by using radix-2^2, radix-2^3, radix-2^4 and radix-2^k [16]. Figure 3.6 shows the radix-r MDC pipelined FFT architecture. This consists of radix-r butterflies, rotators, buffers and commutators. After the radix-r butterfly computes the r-point DFTs, the result is then fed to the buffer and commutator after twiddle factor multiplication. The twiddle factor multiplication is performed by the rotator and its implementation techniques will be discussed in next Chapter. The purpose of the buffer and commutator block is to store the samples coming from one stage and reorder them in the right order for the next stage. Hence, the buffer and commutator block is used after twiddle factor multiplication. Each buffer and commutator block has a different size of buffer determined by the stage number.

.

3.4 Bit-Reversal / I/O Order

Bit reversal is an algorithm that generates a set of indexed data according to a reversing of the bits [60]. This algorithm is used many times to sort out the output frequencies of the FFT, which are usually provided in bit-reversed order. This algorithm permutes a set of indexed data according to a reversing of the bits of the index.

The bit reversal of $N = 2^n$ indexed data is an algorithm that reorders the data according to a reversing of the bits of the index. This means that any sample with index $I = b_{n-1}, ..., b_1, b_0$ moves to the place $BR(I) = b_0, b_1, ..., b_{n-1}$. In other words bit reversal is an inversion operation, i.e., $\mathrm{BR}(x) = \mathrm{BR}^{-1}(x)$ therefore, if data are in natural order, the bit reversal algorithm obtains them in bit-reversed order and vice versa.

There are different approaches to implement bit reversal algorithms. The most common approach is to store the set of data in memory and read the data from the memory afterwords [61, 62]. Other approach is based on the circuits

consists of buffers and multiplexers in series. This architecture also called the optimum bit reversal circuit because it has a minimum number of registers and minimum latency [63].

Chapter 4

Rotation and its Hardware

In this chapter, the different possibilities to implement rotations are discussed. The rotations can be implemented using different algorithms, like general complex multiplier, CORDIC, and complex constant multiplier. The CORDIC and complex constant multiplier are multiplier-less implementations, where shift and adds are used to implement the rotation.

4.1 Rotation

A rotation is the multiplication by a complex number whose magnitude is equal to one, i.e. only the phase of the data is affected. The rotation of a complex number $x + iy$ by an angle α is calculated by the following equations:

$$\begin{aligned} X &= x \cdot \cos\alpha - y \cdot \sin\alpha, \\ Y &= y \cdot \cos\alpha + x \cdot \sin\alpha, \end{aligned} \tag{4.1}$$

where X and Y are the real and imaginary parts of the result. Thus, the rotation can be written as:

$$X + jY = (x \cdot \cos\alpha - y \cdot \sin\alpha) + j\ (y \cdot \cos\alpha + x \cdot \sin\alpha)\,. \tag{4.2}$$

The rotation in (4.1) is shown in Fig. 2. This rotation can be computed in several different ways, including general complex multiplication [20], and the CORDIC algorithm, [21, 22] and algorithms based on constant multiplication [23–26].

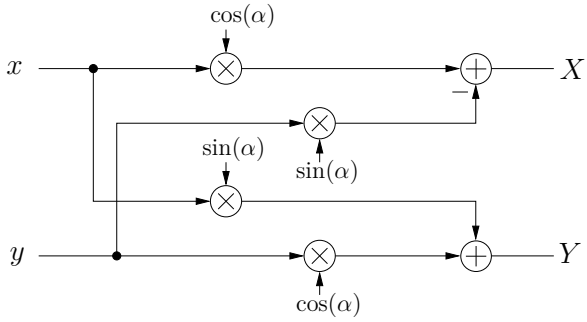

Figure 4.1: Operational diagram of rotation.

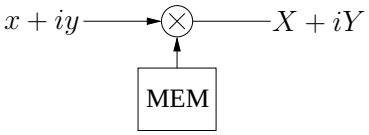

Figure 4.2: General complex multiplier.

4.2 General Complex Multiplier

The most common way of implementing the rotation is by using four multipliers and two adders. When complex multipliers are used, the sine and cosine components of the rotation angle are usually stored in a memory. The architecture of the multiplier is shown in Fig. 4.2, where the multiplier symbol represents the multiplication and addition of (4.1) and the memory is used to store the sine and cosine component of the rotation angles. In pipelined FFT architecture, each stage performs N number of rotations so need to store N number of twiddle factor coefficients in the memory. In order to reduce the size of the coefficient memory, there are a number of approaches, which have been applied [28, 64]. The coefficients in memory are the input to the complex multiplier in order to calculate the rotation of the input sample. Furthermore, the number of $0 \rightarrow 1$ bit transitions between successive coefficient that are read from the coefficient memory is defined as a switching activity (SA), which is related to the power consumption of the circuit [65–67].

The complex multiplier consists of four real multiplications and two additions. Regardless of the application of rotation, the complex multiplier can be realized by different approaches. These approaches generally reduce the number of real multiplications from four to three at additional cost.

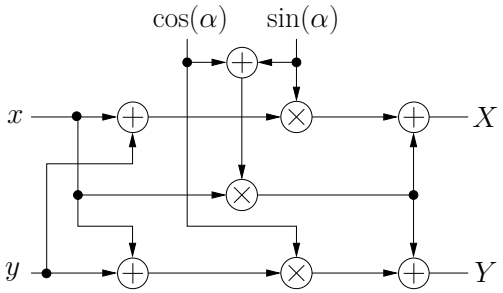

Figure 4.3: Algorithm with three multiplier and five adders.

4.2.1 Approach I

The general rotation (4.1) can also be written in the following approach:

$$\begin{aligned} X &= x \cdot (\cos\alpha + \sin\alpha) - (x+y) \cdot \sin\alpha, \\ Y &= x \cdot (\cos\alpha + \sin\alpha) + (y-x) \cdot \cos\alpha. \end{aligned} \tag{4.3}$$

This consists of only three multiplications and five additions. In this approach one multiplication is replaced by three additions. It is appropriate solution when no multiplier is available. Based on (4.3), the structure is depicted in Fig. 4.3.

4.2.2 Approach II

Another approach to rewrite the rotation (4.1):

$$\begin{aligned} X &= (x+y) \cdot \cos\alpha - x \cdot (\sin\alpha + \cos\alpha), \\ Y &= (x+y) \cdot \cos\alpha + y \cdot (\sin\alpha - \cos\alpha). \end{aligned} \tag{4.4}$$

It requires three real multiplications and three additions. This approach replaces one multiplication by addition. However, it needs to store three coefficients. Since $(\sin\alpha + \cos\alpha)$ and $(\sin\alpha - \cos\alpha)$ can also be precomputed so one replaces the coefficient $\sin\alpha$ and other stores at new location in memory. The structure based on (4.4) is shown in Fig. 4.4.

4.2.3 Approach III

This approach can be applied only in the case of rotation, which are based on lifting [68]. The rotation (4.1) can be written in matrix form as follows:

$$\begin{bmatrix} X \\ Y \end{bmatrix} = \begin{bmatrix} \cos\alpha & \sin\alpha \\ -\sin\alpha & \cos\alpha \end{bmatrix} \times \begin{bmatrix} x \\ y \end{bmatrix} \tag{4.5}$$

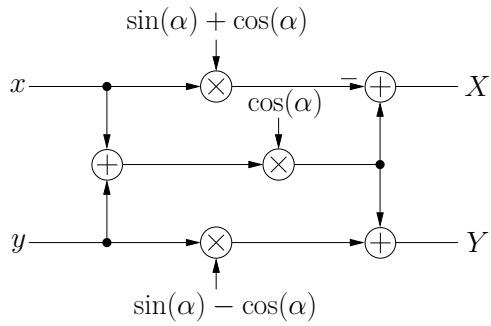

Figure 4.4: Algorithm with three multiplier and three adders.

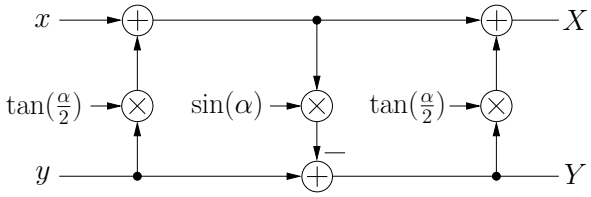

Figure 4.5: Algorithm with three multiplier and three adders.

When the lifting approach is applied to the (4.5). The following equation is obtained:

$$\begin{bmatrix} X \\ Y \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} \cdot \begin{bmatrix} 1 & -\tan\frac{\alpha}{2} \\ 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 \\ \sin\alpha & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & \tan\frac{\alpha}{2} \\ 0 & -1 \end{bmatrix} \times \begin{bmatrix} x \\ y \end{bmatrix} \tag{4.6}$$

By further computing the matrices, the following equations are obtained:

$$\begin{aligned} X &= x - x \cdot \sin\alpha \cdot \tan\tfrac{\alpha}{2} + 2y \cdot \tan\tfrac{\alpha}{2} - y \cdot \tan^2\tfrac{\alpha}{2} \cdot \sin\alpha, \\ Y &= y - x \cdot \sin\alpha - y \cdot \sin\alpha \tan\tfrac{\alpha}{2}. \end{aligned} \tag{4.7}$$

Based on these equations, the structure is depicted in Fig. 4.5. This approach also requires three real multiplications and three additions. However, there is no need for an additional coefficient as it just replaces the $\cos\alpha$ with $\tan\frac{\alpha}{2}$ in memory.

4.3 CORDIC

CORDIC (COordinate Rotation DIgital Computer) is one popular algorithm for the implementation of multiplier-less rotations [21, 22, 69]. It realizes rotation by means of a series of shifts and additions, which reduces the amount of hardware. It is also suitable where multipliers are not available. However, it may affect the accuracy since it is based on approximation.

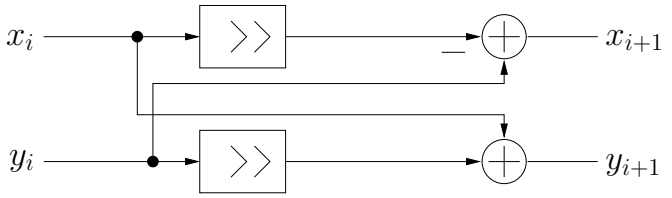

Figure 4.6: CORDIC micro-rotation.

The CORDIC algorithm decomposes the angle that has to be rotated, θ, into a sum of M predefined angles, α_i, according to:

$$\theta = \sum_{i=0}^{M-1} \delta_i \alpha_i + \epsilon, \tag{4.8}$$

where ϵ is the error of the approximation, δ_i indicates the direction of the so called micro-rotation and:

$$\alpha_i = \tan^{-1}(2^{-i}). \tag{4.9}$$

These angles that define the micro-rotations have the property that they can be rotated by shifts and additions, which reduces significantly the hardware resource. These micro-rotations are carried out as follows:

$$\begin{aligned} x_{i+1} &= x_i - y_i\ \delta_i\ 2^{-i}, \\ y_{i+1} &= y_i + x_i\ \delta_i\ 2^{-i}. \end{aligned} \tag{4.10}$$

The hardware circuit for calculating the case of $\delta_i = 1$ is depicted in Fig. 4.6. In Fig. 4.6, the angle α_i that the input datum is rotated is chosen by setting the number of bits that are shifted before the additions and subtractions are carried out.

Usually $\delta \in \{-1, 1\}$. This forces to calculate all the micro-rotations either clockwise or counter clockwise and assures a constant gain of the CORDIC, which can be compensated by multiplying the outputs by:

$$K = \prod_{i=0}^{M} \cos(\alpha_i) = \prod_{i=0}^{M} \cos(\tan^{-1}(2^{-i})). \tag{4.11}$$

This option is preferable when the circuit is used for rotating several different angles, and a constant gain for all of them is required, as happens in the rotators for the FFT. However, in a constant rotator only a single angle θ must be rotated. In this case it is better to consider $\delta_i \in \{-1, 0, 1\}$. This approach is called redundant CORDIC [70] and allows to remove certain micro-rotations, reducing the number of adders.

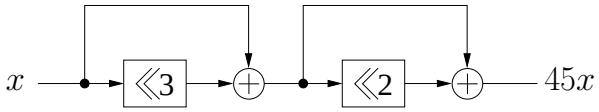

Figure 4.7: Simplified constant multiplication.

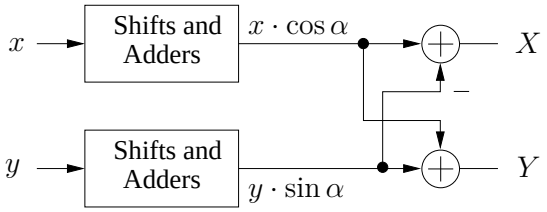

Figure 4.8: Rotation based on constant multiplication.

4.4 Constant Multiplication

For constant multiplication, it is possible to replace general multiplier by shifts, adders, and substractors. Adders and substractors have same complexity, so both are referred as adders. When an input signal is multiplied by more than one constants, a simple method exit to realize each multiplier individually. This can be done optimally for up to 19-bits coefficients [71]. However, it is also possible to utilize the redundancies between the constants in order to reduce the complexity of the hardware. Furthermore, the architecture of a constant multiplier is shown in Fig. 4.7, where the input data x is multiplied with a coefficient 45, being the output $45x$.

In terms of complexity, shift operations are free as they only reduce the number of adders in the implementation of the constant multiplications. For complex multiplications in particular each input is multiplied by two constant coefficients. A dedicated algorithm for realizing multiple constant multiplications (MCM) with two constants have been proposed in [72]. MCM is constant multiplication algorithm, where the input is multiplied with several constant coefficients. In fact, any rotation angle α can be realized with constant multiplications, so it is possible to implement the rotation (4.1) using a constant multiplication algorithm. It is based on the implementation on constant value of the sine and cosine functions of rotation. A general block diagram of the rotator is shown in Fig. 4.8. This consists of two constant multiplication blocks, where constant value of sin and cosine is implemented by shifts and adders with two additional adders, which are used to complete the rotation. In addition, the complexity of the rotation based on constant multiplication depends on the precision requirement of the rotator

The author proposes an architecture to implement the twiddle factor multiplication based on constant multiplication. The different approaches have been used to reduce the complexity and improve the accuracy. In paper F, the ar-

chitecture is proposed, which makes use of the trigonometric identities of the angles. This allows finding expressions that can be reused for different angles. As a result, a simplified architecture is obtained for implementation of the twiddle factor multiplication. Another architecture proposed in paper H, uses the coefficients of twiddle factor , which have low adder cost and more accuracy. This can be obtained using the scaling of the coefficient. The number of adders are further optimized using multiple constant multiplication (MCM) approach.

Chapter 5

Finite Word Length Effects

FFT algorithms are implemented in hardware with finite word length data and arithmetic. As a result, the coefficients and internal signals are quantized (finite word length) in FFT architectures, which determines the input dynamic range, precision, and hardware resources. This generally leads to phenomena known as finite word length effects.

This chapter presents these effects in FFTs, such as coefficient quantization error, data quantization error and over flow. Here data quantization error is explained in terms of round-off error.

5.1 Coefficient Quantization

It is usually required to quantize all the coefficients to a fix number of bits, because it is not possible to keep the unlimited word length of the coefficients. If the coefficients of rotation is perfectly represented by the required number of bits, there would be no error in implementation, however it increases the hardware cost. This is usually a trade off between hardware cost and accuracy. The coefficient quantization causes an error in the computed result of arithmetic operations. Although the nature of coefficient quantization is inherently non statical, useful results may be obtained by means of a statistical analysis.

The quantized coefficient is defined as the sum of coefficient and the error introduced by the coefficient quantization. Thus, the quantized coefficient is defined as:

$$c_q = c + \Delta_c, \tag{5.1}$$

where c is coefficient and Δ_c is the coefficient quantization error. Now, if it uses rounding with b fractional bits, we know that $|\Delta_c| \leq 2^{-(b+1)} = 0.5$ ulp[1].

[1]Unit of least position, i.e., the weight of the least significant bit of the representation. When using b fractional bits, ulp $= 2^{-b}$.

In FFT, rotations are the one of the source of coefficient quantization error. The rotation consists of two coefficients, $\cos\alpha$ and $\sin\alpha$, which have always a value in a range $[-1, 1]$. If $\cos\alpha$ and $\sin\alpha$ are represented by a finite number of bits, a rotation with the coefficient quantization error can be described as,

$$\begin{aligned}(\cos\alpha)_c &= \cos\alpha + \Delta_{\cos\alpha},\\ (\sin\alpha)_c &= \sin\alpha + \Delta_{\sin\alpha},\end{aligned} \tag{5.2}$$

where $\Delta_{\cos\alpha}$ and $\Delta_{\sin\alpha}$ are the coefficient quantization errors of cosine and sine coefficients, respectively. As a result, the overall expression of the rotation with coefficient quantization is represented by:

$$\begin{aligned}X_c &= x\cdot(\cos\alpha + \Delta_{\cos\alpha}) - y\cdot(\sin\alpha + \Delta_{\sin\alpha}),\\ Y_c &= y\cdot(\cos\alpha + \Delta_{\cos\alpha}) + x\cdot(\sin\alpha + \Delta_{\sin\alpha}).\end{aligned} \tag{5.3}$$

Taking all above into account, the rotation error due to quantization of the coefficients is calculated by subtracting the (5.3) form (4.1), we get:

$$\begin{aligned}\epsilon_{\cos\alpha} &= x\cdot\Delta_{\cos\alpha} - y\cdot\Delta_{\sin\alpha},\\ \epsilon_{\sin\alpha} &= y\cdot\Delta_{\cos\alpha} + x\cdot\Delta_{\sin\alpha}.\end{aligned} \tag{5.4}$$

The absolute value of the error becomes:

$$\|E\| = \sqrt{(x^2+y^2)\cdot(\Delta^2_{\cos\alpha} + \Delta^2_{\sin\alpha})}. \tag{5.5}$$

The error depends on the magnitude of the input and the coefficient quantization error. However, the coefficient quantization error can be considered to reduce the error [27].

Each output of the FFT computation passes number of cascaded rotation. Thus, the error is accumulated at the output of FFT. The effect of the coefficient quantization can be computed by using Forbenius norm, which is based on comparing the quantized transfer function to the ideal transfer function [73]. This can be defined as:

$$FN = \sum_{i=0}^{N-1}\sum_{j=0}^{N-1}\left\|Z_{i,j} - \hat{Z}_{i,j}\right\|^2 \quad (dB), \tag{5.6}$$

where Z is the ideal matrix of rotations and $\hat{Z}$ represents quantized coefficients. As a result, the impact of coefficient quantization in FFT could be analyzed.

5.2 Round-Off Error

In fixed point representation, it is inefficient to increase the word length after multiplication. However, the products must be quantized using rounding or truncation. The quantized products can be expressed as the sum of unquantized

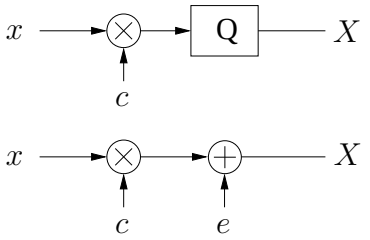

Figure 5.1: Linear noise model of the quantization.

product and a random noise source as shown in Fig. 5.1 [74–76]. Considering the multiplication of the quantized numbers x and c, the product X itself will be quantized to a b-bit that is quantized. The quantization product $X_Q = [cx]_Q$ is modelled by an additive error (e). Thus the quantized product is expressed as:

$$X_Q = [cx]_Q + e. \tag{5.7}$$

Round-off error sources in the design are often modelled as random noise source with statistical properties, variance σ^2 and mean value m, determined by the quantization model and word length [74]. Assume that the variance and mean value for quantization at node i are σ_i^2 and m_i, respectively. Then, the total mean and variance of the round-off noise for K sources is given as [74]

$$m_{\text{tot}} = \sum_{i=1}^{K} \left(m_i \sum_{n=0}^{\infty} h_i(n) \right), \tag{5.8}$$

$$\begin{aligned} \sigma_{\text{tot}}^2 &= \sum_{i=1}^{K} \left(\sigma_i^2 \sum_{n=0}^{\infty} h_i^2(n) \right), \\ &= \sum_{i=1}^{K} \left(\frac{\sigma_i^2}{2\pi} \int_{-\pi}^{\pi} \left| H_i \left(e^{j\omega T} \right) \right|^2 \mathrm{d}\omega T \right), \end{aligned} \tag{5.9}$$

where $h_i(n)$ is the impulse response from the noise source to the output and $H_i\left(e^{j\omega T}\right)$ is the corresponding transfer function. As a result, the variance of the round-off error is equal to sum of the all noise sources.

In the FFT, a multiplication of the data by a rotation introduces the round-off noise, which means that a noise source is added after each multiplication. The total round-off noise is calculated by adding the effect of all round-off noises at the output [76].

5.3 Overflow

The possibility of overflow occurs when the available data range exceeds the available finite length [74]. This happens in addition/subtraction operation of

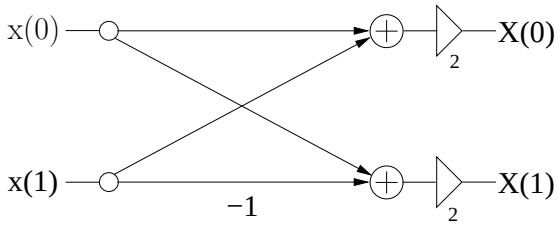

Figure 5.2: Butterfly with scaling.

the FFT. Generally, it is assumed that input data are numbers in the range $[-1, 1]$, with a certain number of bits. According to this typical interpretation, the output value of addition/subtraction operations not in the range $[-1, 1]$, since the result causes overflows.

5.4 Scaling

Scaling is used to prevent the overflows in fixed point realization while at the same time keeping the signal level as high as possible to reduce the round-off noise [10]. It adjusts the range of data by inserting scaling multipliers. However, the scaling should not disturb the transfer function.

5.4.1 Static Data Scaling

In order to avoid overflows in FFT architectures, it is common to introduce a scaling factor after each butterfly stage. To completely guarantee that overflows will not happen, the outputs can be divided by two after a radix-2 butterfly as shown in Fig. 5.2. The factor two comes from the assumption that the inputs have a given value, which can at most be double by adding or subtracting the two inputs. With a bit more knowledge about the statistical properties of the input signal, it is possible to find a bit less restrictive scaling factors using L_P-scaling [74].

5.4.2 Dynamic Data Scaling

The static scaling is often overly prohibiting, as it is statistically rather unlikely that two signals will in fact add up to a large value, and if they do so, the other butterfly output will be small. Therefore, it is of interest to find ways to dynamically adapt the scaling factors to the actual signal processed.

Several schemes have been proposed to perform this dynamic data scaling for FFT processors [10, 77, 78]. The basic idea is to scale down the data when required and scale up the data when possible. In other words, if the butterfly results are large, then scale to make them smaller, if they are in the proper range leave them, and if they are small make them larger, keeping more bits of

significance. If the same scaling factor should be used for all data, it is often not required to be stored. However, all data must be processed in a stage before the scaling decision can be made. In memory-based architectures this can be rather straightforwardly supported by having a memory with a slightly longer word length compared to that of the arithmetic operator inputs and some additional logic keeping track of the stored values. For pipelined architectures, it increases the memory requirements, as most pipelined architectures do not have access to all intermediate data at the same time. These requirements can be reduced by storing the amount of scaling applied to separate blocks of data.

There are basically two trade-offs in the design of such scaling schemes. First, the larger blocks, the more additional memories are required. Second, the smaller blocks, the more additional memory is required to store the scaling information. As the scaling information is basically the exponent in floating-point arithmetic, these schemes are often referred to as block floating-point. Indeed, using a block size of one half sample (separate blocks for real and imaginary parts) would lead to floating-point arithmetic. The use of these techniques also require that data from different blocks are aligned before the butterfly operations.

Chapter 6

Conclusions and Future Work

6.1 Conclusions

In this thesis, the possibilities to improve the complexity and the performance of the rotations in FFTs at algorithmic and arithmetic level were investigated. A framework for the generation of a large number of possible FFT algorithms has been proposed. Based on this framework, the generated FFT algorithms will have the same butterfly operations and data flow, but differ in the twiddle factor multiplication. The difference among these FFT algorithms lies in terms of switching activity of the twiddle factor multiplication, the number of non-trivial multiplications, the coefficient memory complexity, and the round-off noise, which are related to the power consumption, area and performance of the circuit.

The thesis also proposed two classes of architectures to implement rotations based on constant multiplication. The first was based on trigonometric identities, resulting in architectures competitive and often better than previously published rotators. The second is based on coefficient scaling and shows even better results.

6.2 Future Work

The following ideas can be done as a future work:

- ⋆ It would be beneficial to analyze the FFT algorithms generated by the binary tree decomposition for parallel pipelined FFT architectures. The parallelization would lead to that certain bits of the sequential control are fixed for a given stage. Hence, this fact can be taken into account to select algorithms optimized for a certain degree of parallelism and co-optimizing with the architectures.

- ⋆ For twiddle factor multiplication architectures based on trigonometric identities, the accuracy and/or complexity could be improved by using scaling of the coefficients.
- ⋆ With better relative cost measures for the low-adder rotators, CORDIC implementations, and general complex multipliers and coefficient memories, the generated algorithms would be even more useful to determine algorithms suitable for low-complexity FFT algorithms.

Chapter 7

References

References

[1] J. Cooley and J. Tukey, "An algorithm for the machine calculation of complex Fourier series," *Math. Comput.*, vol. 19, pp. 297–301, 1965.

[2] P. Duhamel and H. Hollmann, "'Split radix' FFT algorithm," *Electron. Lett.*, vol. 20, no. 1, pp. 14–16, Jan. 1984.

[3] W.-C. Yeh and C.-W. Jen, "High-speed and low-power split-radix FFT," *IEEE Trans. Signal Process.*, vol. 51, no. 3, pp. 864–874, Mar. 2003.

[4] S. Winograd, "On computing the discrete Fourier transform," *Nat. Acad. Sci. USA,*, vol. 73, no. 4, pp. 1005–1006, Apr. 1976.

[5] ——, "On computing the DFT," *Math. of Comp.,*, vol. 32, no. 1, pp. 175–199, Jan. 1978.

[6] M. T. Heideman, D. H. Johnson, and C. S. Burrus, "Gauss and the history of the fast Fourier transform," *Archive for History of Exact Sciences*, vol. 34, pp. 265–277, 1985.

[7] J. A. C. Bingham, "Multicarrier modulation for data transmission: an idea whose time has come," *IEEE Commun. Mag.*, vol. 28, no. 5, pp. 5–14, May 1990.

[8] J.-J. van de Beek, O. Edfors, M. Sandell, S. K. Wilson, and P. O. Borjesson, "On channel estimation in OFDM systems," in *Proc. IEEE Vehicular Tech. Conf.*, vol. 2, 1995, pp. 815–819.

[9] X. e Sun and H. Wang, "The high effective application of FFT and IFFT of real signal for seismic data processing," in *Proc. Int. Conf. Electrical and Control Engineering*, 2011, pp. 447–450.

[10] A. V. Oppenheim and R. W. Schafer, *Dicrete-Time Signal Processing*. Prentice Hall, 1989.

[11] C. Loeffler, A. Ligtenberg, and G. S. Moschytz, "Practical fast 1-D DCT algorithms with 11 multiplications," in *Proc. IEEE Int. Conf. Acoust. Speech Signal Process.*, 1989, pp. 988–991.

[12] J. Gray, A. and J. Markel, "Digital lattice and ladder filter synthesis," *IEEE Trans. Audio Electroacoust.*, vol. 21, no. 6, pp. 491–500, Jun. 1973.

[13] P. Vaidyanathan, "Passive cascaded-lattice structures for low-sensitivity FIR filter design, with applications to filter banks," *IEEE Trans. Circuits Syst.*, vol. 33, no. 11, pp. 1045–1064, Nov. 1986.

[14] S. He and M. Torkelson, "A new approach to pipeline FFT processor," in *Proc. Int. Parallel Processing Symp.*, 1996, pp. 766–770.

[15] ——, "Designing pipeline FFT processor for OFDM (de)modulation," in *Proc. URSI Int. Symp. Signals Syst. Elect.*, 1998, pp. 257–262.

[16] M. Garrido, J. Grajal, M. A. Sanchez, and O. Gustafsson, "Pipelined radix-2^k feedforward FFT architectures," *IEEE Trans. VLSI Syst.*, 2012, accepted.

[17] A. Cortes, I. Velez, and J. F. Sevillano, "Radix r^k FFTs: Matricial representation and SDC/SDF pipeline implementation," *IEEE Trans. Signal Process.*, vol. 57, no. 7, pp. 2824–2839, Jul. 2009.

[18] C. S. Burrus and P. Eschenbacher, "An in-place, in-order prime factor FFT algorithm," *IEEE Trans. Acoust., Speech, Signal Process.*, vol. 29, no. 4, pp. 806–817, Apr. 1981.

[19] F. Qureshi and O. Gustafsson, "Generation of all radix-2 fast Fourier transform algorithms using binary trees," in *Proc. Europ. Conf. Circuit Theory Design*, 2011, pp. 677–680.

[20] K. K. Parhi, *VLSI Digital Signal Processing Systems, Design and Implementation*. Wiley-Interscience, 1999.

[21] M. Garrido and J. Grajal, "Efficient memoryless CORDIC for FFT computation," in *Proc. IEEE Int. Conf. Acoust. Speech Signal Process.*, vol. 2, 2007, pp. 113–116.

[22] J. E. Volder, "The CORDIC trigonometric computing technique," *IRE Transactions on Electronic Computers*, vol. EC-8, no. 3, pp. 330–334, Sep. 1959.

[23] J.-Y. Oh and M.-S. Lim, "New radix-2 to the 4th power pipeline FFT processor," *IEICE Trans. Electron.*, vol. E88-C, no. 8, pp. 1740–1746, Aug. 2005.

[24] W. Han, A. T. Erdogan, T. Arslan, and M. Hasan, "High-performance low-power FFT cores," *ETRI J.*, vol. 30, no. 3, pp. 451–460, Jun. 2008.

[25] K. Maharatna, E. Grass, and U. Jagdhold, "A 64-point Fourier transform chip for high-speed wireless LAN application using OFDM," *IEEE J. Solid-State Circuits*, vol. 39, no. 3, pp. 484–493, Mar. 2004.

[26] F. Qureshi and O. Gustafsson, "Low-complexity constant multiplication based on trigonometric identities with applications to FFTs," *IEICE Trans. Fundamentals*, vol. E94-A, no. 11, pp. 2361 – 2368, Nov. 2011.

[27] M. Garrido, O. Gustafsson, and J. Grajal, "Accurate rotations based on coefficient scaling," *IEEE Trans. Circuits Syst. II*, vol. 58, no. 10, pp. 662–666, Oct. 2011.

[28] H.-Y. Lee and I.-C. Park, "Balanced binary-tree decomposition for area-efficient pipelined FFT processing," *IEEE Trans. Circuits Syst. I*, vol. 54, no. 4, pp. 889–900, Apr. 2007.

[29] C.-F. Hsiao, Y. Chen, and C.-Y. Lee, "A generalized mixed-radix algorithm for memory-based FFT processors," *IEEE Trans. Circuits Syst. II*, vol. 57, no. 1, pp. 26–30, Jan. 2010.

[30] H. Silverman, "An introduction to programming the Winograd Fourier transform algorithm (WFTA)," *IEEE Trans. Acoust., Speech, Signal Process.*, vol. 25, no. 2, pp. 152–165, Apr. 1977.

[31] M. Narasimha, K. Shenoi, and A. Peterson, "Quadratic residues: Application to chirp filters and discrete Fourier transforms," in *Proc. IEEE Int. Conf. Acoust. Speech Signal Process.*, vol. 1, 1976, pp. 376–378.

[32] L. Bluestein, "A linear filtering approach to the computation of discrete Fourier transform," *IEEE Trans. Audio Electroacoust.*, vol. 18, no. 4, pp. 451–455, 1970.

[33] C. M. Rader, "Discrete Fourier transforms when the number of data samples is prime," *Proc. IEEE*, vol. 56, no. 6, pp. 1107–1108, Jun. 1968.

[34] M. Garrido, "Efficient hardware architectures for the computation of the FFT and other related signal processing algorithms in real time," Ph.D. dissertation, Universidad Politécnica de Madrid, 2009.

[35] S.-C. Moon and I.-C. Park, "Area-efficient memory-based architecture for FFT processing," in *Proc. IEEE Int. Symp. Circuits Syst.*, vol. 5, V-101 - V-104 2003.

[36] P.-Y. Tsai, T.-H. Lee, and T.-D. Chiueh, "Power-efficient continuous-flow memory-based FFT processor for WiMax OFDM mode," in *Proc. Int. Symp. Intelligent Signal Process. Comm.*, 2006, pp. 622–625.

[37] H. S. Stone, "Parallel processing with the perfect shuffle," *IEEE Trans. Comput.*, no. 2, pp. 153–161, Feb. 1971.

[38] J. A. Hidalgo, J. Lopez, F. Arguello, and E. L. Zapata, "Area-efficient architecture for fast Fourier transform," *IEEE Trans. Circuits Syst. II*, vol. 46, no. 2, pp. 187–193, Feb. 1999.

[39] B. G. Jo and M. H. Sunwoo, "New continuous-flow mixed-radix (CFMR) FFT processor using novel in-place strategy," *IEEE Trans. Circuits Syst. I*, vol. 52, no. 5, pp. 911–919, May 2005.

[40] P. Philipov, V. Lazarov, Z. Zlatev, and M. Ivanova, "A parallel architecture for radix-2 fast Fourier transform," in *Proc. IEEE John Vincent Atanasoff Int. Symp. Modern Computing*, 2006, pp. 229–234.

[41] S. F. Gorman and J. M. Wills, "Partial column FFT pipelines," *IEEE Trans. Circuits Syst. II*, vol. 42, no. 6, pp. 414–423, Jun. 1995.

[42] F. Arguello, J. D. Bruguera, R. Doallo, and E. L. Zapata, "Parallel architecture for fast transforms with trigonometric kernel," *IEEE Trans. Parallel Distrib. Syst.*, vol. 5, no. 10, pp. 1091–1099, Oct. 1994.

[43] S. Lee and S.-C. Park, "Modified SDF architecture for mixed DIF/DIT FFT," in *Proc. IEEE Int. Symp. Circuits Syst.*, 2007, pp. 2590–2593.

[44] S.-N. Tang, J.-W. Tsai, and T.-Y. Chang, "A 2.4-Gs/s FFT processor for OFDM-based WPAN applications," *IEEE Trans. Circuits Syst. II*, vol. 57, no. 6, pp. 451–455, Jun. 2010.

[45] M. A. Sanchez, M. Garrido, M. Lopez-Vallejo, and J. Grajal, "Implementing FFT-based digital channelized receivers on FPGA platforms," *IEEE Trans. Aerosp. Electron. Syst.*, vol. 44, no. 4, pp. 1567–1585, Oct. 2008.

[46] M. A. Sanchez, M. Garrido, M. Lopez-Vallejo, J. Grajal, and C. Lopez-Barrio, "Digital channelised receivers on FPGAs platforms," in *Proc. IEEE Int. Radar Conf*, 2005, pp. 816–821.

[47] S. He and M. Torkelson, "Design and implementation of a 1024-point pipeline FFT processor," in *Proc. IEEE Custom Integrated Circuits Conf.*, 1998, pp. 131–134.

[48] M. E. Grandmaison, J. Belzile, C. Thibeault, and F. Gagnon, "Reconfigurable and efficient FFT/IFFT architecture," in *Proc. Canadian Conf. Electrical Comp. Engg.*, vol. 2, 2004, pp. 1115–1118.

[49] A. Cortes, J. F. Sevillano, I. Velez, and A. Irizar, "An FFT core for DVB-T/DVB-H receivers," in *Proc. IEEE Int. Conf. Electron. Circuits Syst.*, 2006, pp. 102–105.

[50] M. Shin and H. Lee, "A high-speed four-parallel radix-2^4 FFT/IFFT processor for UWB applications," in *Proc. IEEE Int. Symp. Circuits Syst.*, 2008, pp. 960–963.

[51] L. Yang, K. Zhang, H. Liu, J. Huang, and S. Huang, "An efficient locally pipelined FFT processor," *IEEE Trans. Circuits Syst. II*, vol. 53, no. 7, pp. 585–589, Jul. 2006.

[52] W. Han, T. Arslan, A. T. Erdogan, and M. Hasan, "Low power commutator for pipelined FFT processors," in *Proc. IEEE Int. Symp. Circuits Syst.*, 2005, pp. 5274–5277.

[53] S. Athar, O. Gustafsson, F. Qureshi, and I. Kale, "On the efficient computation of single-bit input word length pipelined FFTs," *IEICE Electron. Express*, vol. 8, no. 17, pp. 1437–1443, 2011.

[54] E. H. Wold and A. M. Despain, "Pipeline and parallel-pipeline FFT processors for VLSI implementations," *IEEE Trans. Comput.*, no. 5, pp. 414–426, May 1984.

[55] H. Liu and H. Lee, "A high performance four-parallel 128/64-point radix-24 FFT/IFFT processor for MIMO-OFDM systems," in *Proc. IEEE Asia-Pacific Conf. Circuits Syst.*, 2008, pp. 834–837.

[56] N. Li and N. P. van der Meijs, "A radix 2^2 based parallel pipeline FFT processor for MB-OFDM UWB system," in *Proc. IEEE Int. SOC Conf.*, 2009, pp. 383–386.

[57] R. Rabiner and B. Gold, *Theory and application of Digital Signal Processing.* Prentice Hall, Inc, 1975.

[58] E. E. Swartzlander, W. K. W. Young, and S. J. Joseph, "A radix 4 delay commutator for fast Fourier transform processor implementation," *IEEE J. Solid-State Circuits*, vol. 19, no. 5, pp. 702–709, Oct. 1984.

[59] E. E. Swartzlander, V. K. Jain, and H. Hikawa, "A radix-8 wafer scale FFT processor," *J. VLSI Signal Processing*, vol. 4, pp. 165–176, May 1992.

[60] B. Gold and C. M. Rader, *Digital Processing of Signals.* McGraw-Hill, 1969.

[61] D. Sundararajan, M. Omair Ahmad, and M. N. S. Swamy, "A fast FFT bit-reversal algorithm," *IEEE Trans. Circuits Syst. II*, vol. 41, no. 10, pp. 701–703, Oct. 1994.

[62] J. M. Rius and R. De Porrata-Doria, "New FFT bit-reversal algorithm," *IEEE Trans. Signal Process.*, vol. 43, no. 4, pp. 991–994, Apr. 1995.

[63] M. Garrido, J. Grajal, and O. Gustafsson, "Optimum circuits for bit reversal," *IEEE Trans. Circuits Syst. II*, vol. 58, no. 10, pp. 657–661, Oct. 2011.

[64] F. Qureshi and O. Gustafsson, "Analysis of twiddle factor memory complexity of radix-2^i pipelined FFTs," in *Proc. Asilomar Conf. Signals Syst. Comput.*, 2009, pp. 217–220.

[65] J.-M. Wu and Y.-C. Fan, "Coefficient ordering based pipelined FFT/IFFT with minimum switching activity for low power WiMAX communication system," in *Proc. IEEE Int. Symp. Consumer Electronics*, 2006, pp. 1–4.

[66] M. Hasan, T. Arslan, and J. S. Thompson, "A novel coefficient ordering based low power pipelined radix-4 FFT processor for wireless LAN applications," *IEEE Trans. Consum. Electron.*, vol. 49, no. 1, pp. 128–134, Feb. 2003.

[67] F. N. Najm, "A survey of power estimation techniques in VLSI circuits," *IEEE Trans. VLSI Syst.*, vol. 2, no. 4, pp. 446–455, Dec. 1994.

[68] S. Chan and P. Yiu, "An efficient multiplierless approximation of the fast Fourier transform using sum-of-powers-of-two (SOPOT) coefficients," *IEEE Signal Process. Lett.*, vol. 9, no. 10, pp. 322–325, Oct. 2002.

[69] R. Andraka, "A survey of CORDIC algorithms for FPGA based computers," in *Proc. Int. Symp. Field Programmable*, 1998, pp. 191–200.

[70] J.-A. Lee and T. Lang, "Constant-factor redundant CORDIC for angle calculation and rotation," *IEEE Trans. Comput.*, vol. 41, no. 8, pp. 1016–1025, Aug. 1992.

[71] O. Gustafsson, A. G. Dempster, K. Johansson, M. D. Macleod, and L. Wanhammar, "Simplified design of constant coefficient multipliers," *Circuits Syst. Signal Process.*, vol. 25, pp. 225–251, Apr. 2006.

[72] A. G. Dempster and M. D. Macleod, "Multiplication by two integers using the minimum number of adders," in *Proc. IEEE Int. Symp. Circuits Syst.*, vol. 2, 2005, pp. 1814–1817.

[73] S. C. Chan and P. M. Yiu, "An efficient multiplierless approximation of the fast Fourier transform using sum-of-powers-of-two (SOPOT) coefficients," *IEEE Signal Process. Lett.*, vol. 9, no. 10, pp. 322–325, Oct. 2002.

[74] L. Wanhammar, *DSP Integrated Circuits.* Linköping University: Academic Press, 1999.

[75] W.-H. Chang and T. Q. Nguyen, "On the fixed-point accuracy analysis of FFT algorithms," *IEEE Trans. Signal Process.*, vol. 56, no. 10, pp. 4673–4682, Oct. 2008.

[76] A. V. Oppenheim and C. J. Weinstein, "Effects of finite register length in digital filtering and the fast Fourier transform," *Proc. IEEE*, vol. 60, no. 8, pp. 957–976, Aug. 1972.

[77] K. Kalliojärvi and J. Astola, "Roundoff errors in block-floating-point systems," *IEEE Trans. Signal Process.*, vol. 44, no. 4, pp. 783–790, Apr. 1996.

[78] T. Lenart and V. Öwall, "Architectures for dynamic data scaling in 2/4/8K pipeline FFT cores," *IEEE Trans. VLSI Syst.*, vol. 14, no. 11, pp. 1286–1290, Nov. 2006.