# SPEEDING-UP FAST FOURIER TRANSFORM

By

Mohammed Ahmed Elmotaz Bellah Elsayed

A Thesis Submitted to the
Faculty of Engineering at Cairo University
in Partial Fulfilment of the
Requirements for the Degree of
MASTER OF SCIENCE
in
Electronics and Communications Engineering

FACULTY OF ENGINEERING, CAIRO UNIVERSITY
GIZA, EGYPT
2016

# SPEEDING-UP FAST FOURIER TRANSFORM

By

Mohammed Ahmed Elmotaz Bellah Elsayed

A Thesis Submitted to the
Faculty of Engineering at Cairo University
in Partial Fulfilment of the
Requirements for the Degree of
MASTER OF SCIENCE
in
Electronics and Communications Engineering

Under the Supervision of

Prof. Hossam Ali Fahmy
professor
Electronics and Communications Engineering Department
Faculty of Engineering, Cairo University

Asst Prof. Omar Ahmed Nasr
Assistant Professor
Electronics and Communications Engineering Department
Faculty of Engineering, Cairo University

FACULTY OF ENGINEERING, CAIRO UNIVERSITY
GIZA, EGYPT
2016

# SPEEDING-UP FAST FOURIER TRANSFORM

By

Mohammed Ahmed Elmotaz Bellah Elsayed

A Thesis Submitted to the
Faculty of Engineering at Cairo University
in Partial Fulfilment of the
Requirements for the Degree of
MASTER OF SCIENCE
in
Electronics and Communications Engineering

Approved by the Examining Committee:

---

Prof. Hossam Ali Fahmy, Thesis Main Advisor

---

Asst Prof. Omar Ahmed Nasr, Member

---

Prof. Mohamed Fathy Abu El-Yazeed, Internal Examiner

---

Prof. Hani Fikry Ragai, External Examiner
(Electronics and Communications Engineering department,
Faculty of Engineering, Ain Shams University)

FACULTY OF ENGINEERING, CAIRO UNIVERSITY
GIZA, EGYPT
2016

**Engineer's Name:** Mohammed Ahmed Elmotaz Bellah Elsayed
**Date of Birth:** 28/4/1990
**Nationality:** Egyptian
**E-mail:** maeb28490@gmail.com
**Phone:** 01142701705
**Address:** Electronics and Communications Engineering Department,
Cairo University,
Giza 12613, Egypt

**Registration Date:** 1/10/2012
**Awarding Date:** / /2016
**Degree:** Master of Science
**Department:** Electronics and Communications Engineering

**Supervisors:**

Prof. Hossam Ali Fahmy
Asst Prof. Omar Ahmed Nasr

**Examiners:**

| | |
|---|---|
| Prof. Hossam Ali Fahmy | (Thesis main advisor) |
| Asst Prof. Omar Ahmed Nasr | (Member) |
| Prof. Mohamed Fathy Abu El-Yazeed | (Internal examiner) |
| Prof. Hani Fikry Ragai | (External examiner) |

**Title of Thesis:**

SPEEDING-UP FAST FOURIER TRANSFORM

**Key Words:**
Fast Fourier Transform (FFT); CORDIC; SQNR; Single-path Delay Feedback

**Summary:**
This work proposes HardWare-Friendly FFT (HW-F FFT): a restructuring of radix-r FFT butterfly in order to achieve less area-time-power product compared with the conventional algorithm. Moreover, HW-F FFT allows the use of the unequal-gain CORDIC types without the need of any compensation after them. In one case study, Single-path Delay Feedback (SDF) pipeline architecture is used. Given the same hardware resources, HW-F FFT achieves a substantial increase in the Signal-to-Quantization Noise Ratio (SQNR) performance. The proposed algorithm offers up to 75 dB SQNR gain compared to the conventional FFT for different radix-r FFT sizes when different CORDICs and complex multiplier sizes are employed. In the same case study, if it is required to maintain a certain SQNR level, HW-F FFT achieves less computational area, up to 40% less, compared with the conventional FFT.

# Acknowledgments

Foremost, I take this opportunity to express my deep gratitude and appreciation to my advisors Prof. Hossam Fahmy and Dr. Omar Nasr for their continuous help, support, guidance, enthusiasm, patience, and encouragement. I learned a lot from their vision and dedication. I could not have imagine having better advisors.

I want to thank my parents for thier guidance and support. Ahmed, My Father and Huda, My Mother, truely the heaven is underneath your feet.

Finally, I would like to thank my friends who supported me , specially EECE department teaching assistants, through this venture and tried to help me to the best of their abilities.

# Dedication

To the beloved ones, who are not here anymore.

# Table of Contents

# List of Tables

# List of Figures

# List of Algorithms

# List of Symbols and Abbreviations

| Abbreviations | Description |
|---|---|
| **ASIC** | Application-Specific Integrated Circuit. |
| **BER** | Bit Error Rate. |
| **CORDIC** | COordinate Rotation DIgital Computer. |
| **DCT** | Discrete Cosine Transform. |
| **DFT** | Discrete Fourier Transform. |
| **DIF** | Decimation In Frequency. |
| **DIT** | Decimation In Time. |
| **DST** | Discrete Sine Transform. |
| **DVB-H** | Digital Video Broadcasting – Handheld. |
| **DVB-T** | Digital Video Broadcasting – Terrestrial. |
| **DVB-T2** | Digital Video Broadcasting – Second Generation Terrestrial. |
| **EEAS** | Extended Elementary Angle Set. |
| **FFT** | Fast Fourier Transform. |
| **GPS** | Global Positioning System. |
| **HW-F FFT** | HardWare-Friendly FFT. |
| **IFFT** | Inverse Fast Fourier Transform. |
| **iid** | Independent and Identically Distributed. |
| **JPEG** | Joint Photographic Experts Group. |
| **LTE** | Long Term Evolution. |
| **MDC** | Multiple-Path Delay Commutator. |
| **MPEG-4** | Moving Picture Experts Group Phase 4. |
| **MPEG-4 AVC** | Moving Picture Experts Group Phase 4, Advanced Video Coding. |
| **MRI** | Magnetic Resonance Imaging. |
| **MSR** | Mixed Scaling Rotation. |

| | |
|---|---|
| **MVR** | Modified Vector Rotational. |
| **OFDM** | Orthogonal Frequency-Division Multiplexing. |
| **OFDMA** | Orthogonal Frequency Division Multiple Access. |
| **ROM** | Read Only Memory. |
| **SDF** | Single-path Delay Feedback. |
| **SFG** | Signal Flow Graph. |
| **SIR** | Signal to Interference Ratio. |
| **SNR** | Signal to Noise Ratio. |
| **SQNR** | Signal to Quantization Noise Ratio. |
| **TWR** | Twiddle Factor Rotator. |

| Symbols | Description |
|---|---|
| **f** | Frequency in Hertz. |
| **t** | Time in second. |

# List of Publications

## Published:

[1] M. El-Motaz, O. Nasr, and K. Osama, "A cordic-friendly fft architecture," in *Wireless Communications and Mobile Computing Conference (IWCMC), 2014 International*, Aug. 2014, pp. 1087–1092.

# Abstract

Fast Fourier Transform (FFT) is the “indispensable operation” in signal processing realm. Being used in audio and image applications, spectrum and network analyzers, radar and communication systems, or even in astronomy and quantum mechanics fields made it earn that title Indisputably. Thus, enhancing FFT modules in terms of area, time, or power has a profound impact on various fields.

The butterfly-based structure of the FFT algorithm is the main reason responsible for reducing the arithmetic operations required to implement the transform. Yet, emerging systems are still hungry for more reduction in arithmetic operations. Among these operations, multiplication by the twiddle factors is the most expensive. From the implementation point of view, these multiplication operations can be synthesized using either complex multipliers are COordinate Rotation DIgital Computers (CORDIC).

This work proposes HardWare-Friendly FFT (HW-F FFT): a restructuring of radix-r FFT butterfly in order to achieve less area-time-power product compared with the conventional algorithm. Moreover, HW-F FFT allows the use of the unequal-gain CORDIC types without the need of any compensation after them.

In one case study, Single-path Delay Feedback (SDF) pipeline architecture is used. Given the same hardware resources, HW-F FFT achieves a substantial increase in the Signal-to-Quantization Noise Ratio (SQNR) performance. The proposed algorithm offers up to 75 dB SQNR gain compared to the conventional FFT for different radix-r FFT sizes when different CORDICs and complex multiplier sizes are employed. In the same case study, if it is required to maintain a certain SQNR level, HW-F FFT achieves less computational area, up to 40 % less, compared with the conventional FFT.

# Chapter 1

# Introduction

Fast Fourier Transform (FFT) is one of the most important and fundamental building blocks in signal processing and communications. Since 1965, when James Cooley of IBM and John Tukey of Princeton have introduced FFT to the world [1], the optimization in both algorithmic level and hardware implementation levels is an active area of research. Researchers try to increase its speed, decrease its area, or minimize its power consumption to enhance the performance, reduce the cost, and increase the battery life of different systems.

FFT module is vital for many applications. For instance, despite that the idea of Orthogonal Frequency-Division Multiplexing (OFDM) has been building for a very long time, using the FFT made it viable [2]. Most of today's wireless communications standards are based on OFDM (Orthogonal Frequency Division Multiplexing) and OFDMA (Orthogonal Frequency Division Multiple Access). Both technologies are based on making an inverse FFT (IFFT) at the transmitter side, and FFT at the receiver side. Long Term Evolution (LTE), WiFi and high speed WiFi (as 802.11ac), Digital Video Broadcasting standards (DVB-T, DVB-T2, DVB-H) and many other wireless standards are all based on OFDM or OFDMA. Even some GPS decoding algorithms use FFT.

In the Image and Video processing domain, the widely used JPEG lossy image compression technique, MPEG-4 video compression and H.264/MPEG-4 AVC which are used in video communications are all based on the Discrete Cosine Transform (DCT), which is practically implemented using the blocks of FFTs.

FFT is also used in all applications that need Spectral Analysis such as spectrum sensing [3], light field reconstruction [4], magnetic resonance imaging (MRI) [5], noise filtration, remote sensing, image enhancement, machine learning and others.

In many of the previously mentioned applications, power consumption, area and speed of the FFT is of major consideration. For example, in LTE systems, the implementation of FFT block should be fast enough to meet the latency and throughput requirements of the system. Moreover, because it is a wireless system and the nodes are mobile, power

consumption is always a major issue. Reducing the power consumption of all receiver blocks, including the FFT, helps in extending the battery life of the devices. This is especially important for devices that need extended battery life such as body area sensor networks and sensors that rely on energy harvesting. Hence, reducing the complexity of the FFT implementation, which can be reflected on operating in a better power-speed-area performance point, will have a large impact on many domains.

## 1.1 History

Jean-Baptiste Joseph Fourier defined the frequency domain as a relation in the continuous time domain samples,

$$X(f) = \int_{-\infty}^{\infty} x(t)e^{-j2\pi ft}dt \tag{1.1}$$

where t is time and f is frequency in hertz. The continuous relation is a powerful mathematical tool and useful for signal processing analysis. However, for implementation purpose, we use the discrete form,

$$X(k) = \sum_{n=0}^{N-1} x(n)e^{-j\frac{2\pi nk}{N}} \tag{1.2}$$

where n is the discrete time index and k is discrete frequency index. This transformation called Discrete Fourier Transform (DFT).

Many other transformation has been derived from the DFT, such as Discrete Cosine Transform,

$$X(k) = \sum_{n=0}^{N-1} x(n)\cos(\frac{\pi}{N}(n+\frac{1}{2})k) \tag{1.3}$$

which is used in compression algorithms such as JPEG since it has a very attractive property called “energy compaction”. This property concentrate the energy of the signal in a few elements which is very useful in the compression algorithms.

Also another Fourier-related transform is Discrete Sine Transform (DST),

$$X(k) = \sum_{n=0}^{N-1} x(n)\sin(\frac{\pi}{N}(n+\frac{1}{2})(k+1)) \tag{1.4}$$

which is used to solve the partial differential equations.

DFT optimization has been the focus of many researches over years.

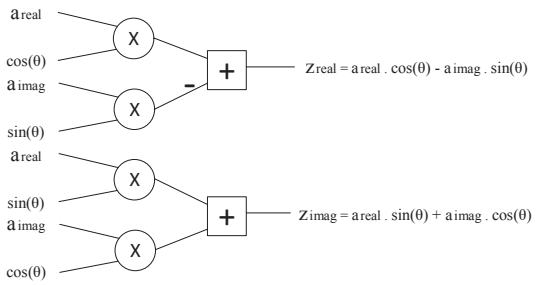


**Figure 1.1: The common complex multiplier architecture**

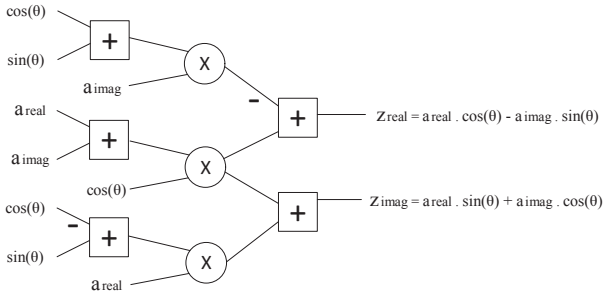


**Figure 1.2: Gauss's complex multiplier architecture**

## 1.2 Question Marks

### 1.2.1 From the hardware perspective, is the complex multiplier the most efficient solution to perform rotation?

In complex multiplier, there are 4 input words, $a_{real}$,$a_{imag}$,$b_{real}$ and$b_{imag}$. The output is presented as, $z_{real} = a_{real}b_{real} - a_{imag}b_{imag}$ and $z_{imag} = a_{real}b_{imag} + a_{imag}b_{real}$. Each one of them can take any combination of bits. When we use the complex multiplier for rotation, three of them can be assigned independently, however, the forth will depend completely on another argument; $b_{real} = \cos(\theta)$ and$b_{imag} = \sin(\theta)$ are dependent on each other. Therefore, there is redundancy when we use complex multiplier for rotation.

The common architecture of a complex multiplier using multipliers and adders is shown in Fig. 1.1. this architecture uses 4 multiplier and 2 adders. on the other hand, Gauss's architecture, shown in Fig. 1.2, uses only 3 multiplication and 5 adders. However, Both architecture uses considerably large hardware to perform a rotation.

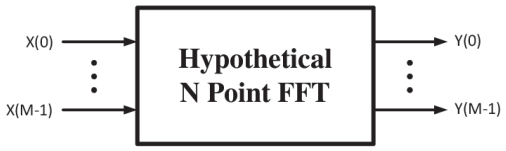


**Figure 1.3: Hypothetical N point FFT Block**

### 1.2.2 lower bound of the memory size inside the FFT block?

If the FFT block take M samples per time unit and give in average M samples per time unit then the minimum number of memory words in the FFT block are N-M where N is the FFT size. Since, If the FFT inputs are Independent and identically distributed (iid) normal random samples then the output are iid normal random samples with respect to each other and dependent in the all N input point of the corresponding FFT.

the first statement states that no way to generate L independent word from M word when M<L since the fft block is not stochastic block. If we assume that is true then if we take M from the L generated words then there equations must form independent system of equations. therefore the have unique inverse (if linear) or finite set of non linear. Therefore you will know the output and know the other L-M output, therefore they are not independent.

### 1.2.3 What is the fastest FFT?

Since FFT is essentially matrix multiplication, no one till now can tell what is the fastest FFT. Till now the researchers can not tell what is the fastest matrix multiplication algorithm.

### 1.2.4 Am I interested?

If you are going to implement FFT on hardware using CORDIC, this work might be interesting for you. If you are going to design robot manipulation systems or graphics and animation systems or speech/music synthesis and communication systems, this algorithm might be interesting for you. Since these systems by its nature use complicated algorithms that will need CORDIC anyway. Therefore, if the FFT is a part of your design and you want to share the CORDIC resources among these algorithms, this work will give you a way to share the CORDIC resource efficiently with FFT.

## 1.3 Organization of the thesis

The remainder of this thesis organized as follows. Chapter 2 previews a literature survey which this work is based on. It is about fast Fourier transforms and CORDIC algorithms and their hardware implementation. After that, chapter 3 illustrate the basic idea of the proposed algorithm through equations and signal flow graphs (SFG). Chapter 4 analyzes the error of the proposed algorithm in order to determine the right objective function that will maximize the SQNR. Chapter 5 previews the optimization methodology followed to solve the optimization problem. Chapter 6 carries out SQNR and hardware metrics evaluation. Chapter 7 draws the conclusion and lists some useful points worthy to be investigated in the future.

# Chapter 2

# Literature Review

Literature review for this thesis is concentrated on two areas: CORDIC and FFT algorithms and hardware architecture.

For CORDIC, we will preview the general concept of the CORDIC from algorithmic and hardware perspective. Then, we will dig more deeply into radix-2 unequal-gain CORDIC algorithms, being more efficient than the others.

CORDIC algorithm uses shift and add operations to perform complicated arithmetic operations that would otherwise require multiplications and extensive use of look-up tables [6]. Several algorithms have been introduced to modify the original CORDIC algorithms for systems involving pre-known angle rotations. The authors in [7] present optimization schemes and CORDIC circuits for fixed and known rotations. In [8], MVR-CORDIC was introduced to improve speed and SQNR. However it resulted in unequal CORDIC gain at every FFT stage, forcing gain compensation after each micro-rotation. In [9], EEAS-CORDIC was introduced to increase the angle space and simplify the scaling stage slightly. In [10], MSR-CORDIC integrates the scaling stage into the rotation stage, which resulted in an improvement in the speed of the algorithm. However, there was a significant decrease in the SQNR. The unequal CORDIC gain and large Read Only Memory (ROM) sizes required by the modified CORDICs, made the original CORDIC algorithm, or marginal modification version of it, the best choice for authors implementing CORDIC-based FFT as in [11–14]. Some authors have proposed a scaling-free CORDIC algorithm. This technique and its modified versions, as described in [15–18], are based on choosing elementary angle subset from the whole CORDIC angle space to synthesize the rotation angle. The condition, this subset is chosen based on, is that $\cos(\theta_{elementary}) = 1$ relative to operands' fixed point representation. Since the CORDIC scaling factor is related to the cosine of the angle, the algorithm is scaling-free. Although it seems that it has solved the unequal CORDIC gain problem, however, this algorithm has many drawbacks, as follows:

- it has extremely small convergence range. Therefore, its modified versions, [17], proposed to fix this problem by folding the angle domain by introducing, as an

option, 45 degree rotation. This solution needs $1/\sqrt{2}$ constant multiplier which increase the hardware area and decrease its regularity;

- its convergence range decreases as the operand size in bits increases. Conventional CORDIC may outperform it in terms of hardware requirements when the word-length reaches 20 bits [17];
- it does not take the advantage of the whole CORDIC angle domain; so it needs more iterations to reach the same SQNR as in the other CORDIC types.

All the modifications in[8–10] focus on the CORDIC algorithm. Recent researches focus on the optimization of the implementation of FFT architecture as in [11–14]. However, because of

For FFT, we are going to start with the basic idea of J. W. Cooley and J. W. Tukey to decompose the Fourier transform which made it worthy the title "Fast". Then we will walk through the implementation of this algorithm on hardware.

In the conventional FFT architecture, equal CORDIC gains is required for all the synthesized rotations. This constraint is enforced in order to avoid gain compensation after each FFT stage. Equal CORDIC gains for different rotations can be obtained by the original CORDIC algorithm in Volder and MSR. However, MVR and EEAS CORDIC algorithms introduce unequal CORDIC gains for different rotations. Therefore, the original CORDIC algorithm and MSR CORDIC are the best choices for authors implementing CORDIC-based FFT [19, 20].

Different applications impose wide-range of specifications on FFT implementation, such as high data throughput or short processing time with moderate silicon area and power consumption. For this type of applications In high throughput applications, the pipeline-based architectures are the most suitable [21]. Pipeline-based architectures are divided into two types, parallel pipeline and serial pipeline architectures [22, 23]. In serial pipeline architecture, the hardware cost is minimized as well as the I/O requirements. However, parallel-pipeline architecture achieves higher throughput and lower latency at the cost of substantial increase in the number of memory elements, computational units and I/O requirements.

In communication systems, the massive throughput offered by parallel-pipeline architecture is not needed. On the other hand, the silicon area and power consumption are required to be minimized. Therefore when it is related to communication systems, serial pipeline is a suitable design choice [21, 24, 25].

## 2.1 Review of related CORDIC techniques

The necessity for a replacement of the analog navigation computer of the B-58 aircraft by a high-accuracy, high-performance digital computer was the mother of the CORDIC invention [26]. In 1959, Jack E. Volder introduced the CORDIC algorithm [27] as an iterative method used to implement many elementary functions, including but not limited to,

- trigonometric functions, such as sin(.), cos(.), tan(.), and $\tan^{-1}(.)$;
- hyperbolic functions, such as sinh(.), cosh(.), tanh(.), and $\tanh^{-1}(.)$;
- logarithm and exponential functions, such as ln(.), exp(.);
- and square-root function, $\sqrt{(.)}$ .

Some of these functions are used intensively in navigation equations calculation, e.g. rotating a vector $\begin{bmatrix} x \\ y \end{bmatrix}$, which belongs to two-dimensional euclidean plane $\mathbb{R}^2$, by an arbitrary angle $\theta$.

### 2.1.1 General Concept

The basic form of vector rotation, demonstrated in Fig. 2.1 (a), considers the problem of rotating a complex number $x_0 + jy_0$ to a new point $x_\infty + jy_\infty$ by an arbitrary angle $\theta$. The rotation can be computed as $x_\infty + jy_\infty = (x_0 + jy_0)e^{j\theta}$ or in matrix form as,

$$\begin{bmatrix} x_\infty \\ y_\infty \end{bmatrix} = \begin{bmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{bmatrix} \begin{bmatrix} x_0 \\ y_0 \end{bmatrix}. \tag{2.1}$$

CORDIC algorithm decomposes the angle $\theta$ to elementary angle set as, $\theta = \theta_0 + \theta_1 + \cdots + \theta_{N-1}$, in order to simplify the hardware. Hence, the output point equation can be written as $x_\infty + jy_\infty = (x_0 + jy_0) \prod_{m=0}^{m=N-1} e^{j\theta_m}$ or in matrix form as,

$$\begin{aligned}
\begin{bmatrix} x_1 \\ y_1 \end{bmatrix} &= \begin{bmatrix} \cos(\theta_0) & -\sin(\theta_0) \\ \sin(\theta_0) & \cos(\theta_0) \end{bmatrix} \begin{bmatrix} x_0 \\ y_0 \end{bmatrix} \\
\begin{bmatrix} x_2 \\ y_2 \end{bmatrix} &= \begin{bmatrix} \cos(\theta_1) & -\sin(\theta_1) \\ \sin(\theta_1) & \cos(\theta_1) \end{bmatrix} \begin{bmatrix} x_1 \\ y_1 \end{bmatrix} \\
\vdots \quad & \vdots \qquad\qquad \vdots \\
\begin{bmatrix} x_N \\ y_N \end{bmatrix} &= \begin{bmatrix} \cos(\theta_{N-1}) & -\sin(\theta_{N-1}) \\ \sin(\theta_{N-1}) & \cos(\theta_{N-1}) \end{bmatrix} \begin{bmatrix} x_{N-1} \\ y_{N-1} \end{bmatrix} \qquad (2.2) \\
\begin{bmatrix} x_\infty \\ y_\infty \end{bmatrix} &= \begin{bmatrix} x_N \\ y_N \end{bmatrix}. \qquad (2.3)
\end{aligned}$$

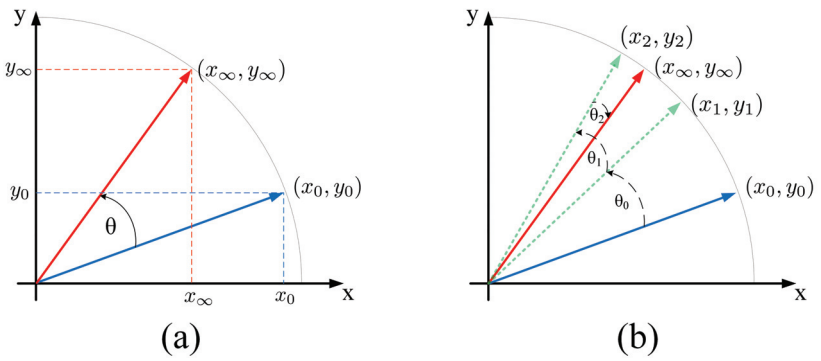


**Figure 2.1: Vector rotation in two-dimensional euclidean plane $\mathbb{R}^2$, (a) using one iteration, (b) using three iterations.**

Fig. 2.1 (b) illustrates an example of a vector rotation by an angle $\theta$ decomposed into three elementary angles as, $\theta = \theta_0 + \theta_1 + \theta_2$. Equation 2.2 can be written as follows,

$$\begin{bmatrix} x_N \\ y_N \end{bmatrix} = \left( \prod_{m=0}^{m=N-1} \begin{bmatrix} \cos(\theta_m) & -\sin(\theta_m) \\ \sin(\theta_m) & \cos(\theta_m) \end{bmatrix} \right) \begin{bmatrix} x_0 \\ y_0 \end{bmatrix} \tag{2.4}$$

$$\begin{bmatrix} x_N \\ y_N \end{bmatrix} = \left( \prod_{m=0}^{m=N-1} \cos(\theta_m) \begin{bmatrix} 1 & -\tan(\theta_m) \\ \tan(\theta_m) & 1 \end{bmatrix} \right) \begin{bmatrix} x_0 \\ y_0 \end{bmatrix} \tag{2.5}$$

The counter $m$ represents the iteration index, and hence the equation of the $m^{th}$ CORDIC iteration is given by,

$$\begin{bmatrix} x_{m+1} \\ y_{m+1} \end{bmatrix} = \cos(\theta_m) \begin{bmatrix} 1 & -\tan(\theta_m) \\ \tan(\theta_m) & 1 \end{bmatrix} \begin{bmatrix} x_m \\ y_m \end{bmatrix}. \tag{2.6}$$

All different CORDIC algorithms agreed on the previous way to decompose the rotation operation, however, they differ on how to create the elementary angle set that will form the angle $\theta$.

### 2.1.2 Conventional CORDIC

Conventional CORDIC refers to the original CORDIC algorithm introduced by Volder [27]. He suggested to decompose $\theta$ to discrete set of angles as, $\{\theta_0, \theta_1, \ldots, \theta_m, \ldots, \theta_{N-1}\}$, where $\theta_m = \tan^{-1}(d(m)2^{-m})$ and $d(m) = \pm 1$. Therefore, the $m^{th}$ conventional CORDIC iteration is given by,

$$\begin{bmatrix} x_{m+1} \\ y_{m+1} \end{bmatrix} = \mathrm{CG}(m)\begin{bmatrix} 1 & -d(m)2^{-m} \\ d(m)2^{-m} & 1 \end{bmatrix}\begin{bmatrix} x_m \\ y_m \end{bmatrix} \tag{2.7}$$

$$\mathrm{CG}(m) = \cos(\tan^{-1}(d(m)2^{-m})) = \cos(d(m)\tan^{-1}(2^{-m}))$$

$$\mathrm{CG}(m) = \cos(\tan^{-1}(2^{-m})) = \frac{1}{\sqrt{1+2^{-2m}}} \tag{2.8}$$

where CG refers to "CORDIC Gain". Due to the cosine insensitivity to the angle's sign, $d(m)$ does not contribute in CG($m$) value.

Thus, for example, if the total number of iteration $N$ equals to 3, the elementary angle set are $\{\pm\tan^{-1}(2^{-0}), \pm\tan^{-1}(2^{-1}), \pm\tan^{-1}(2^{-2})\} = \{\pm 45°, \pm 26.565°, \pm 14{,}036°\}$.

According to equation 2.7, each iteration is constrained to one of two elementary angles, i.e. $\theta_m \in \{\pm\tan^{-1}(2^{-m})\}$.Therefore, going back to the example, the $0^{th}$ iteration may rotate the input vector with $+45°$ or $-45°$, the $1^{st}$ iteration may rotate its input vector with $+26.565°$ or $-26.565°$, and finally the $2^{nd}$ iteration may rotate with $+14{,}036°$ or $-14{,}036°$. Hence the number of all reachable angle set $\phi$ is $2^N$, and in the example $\phi \in \{\pm 85.6013°, \pm 57.5288°, \pm 32.4712°, \pm 4.3987°\}$. As the number of iterations increased, the reachable angle set $\phi$ got richer. Fig. 2.2 shows the conventional CORDIC's reachable angles using 3 iterations and 8 iterations.

As a result to equation 2.7, the output point equation can be written as,

$$\begin{bmatrix} x_N \\ y_N \end{bmatrix} = \mathrm{CG}_{tot}\left(\prod_{m=0}^{m=N-1}\begin{bmatrix} 1 & -d(m)2^{-m} \\ d(m)2^{-m} & 1 \end{bmatrix}\right)\begin{bmatrix} x_0 \\ y_0 \end{bmatrix} \tag{2.9}$$

$$\mathrm{CG}_{tot} = \prod_{m=0}^{m=N-1} \mathrm{CG}(m). \tag{2.10}$$

For fixed number of iteration N, the total CORDIC gain $\mathrm{CG}_{tot}$ is constant for all the reachable angles $\phi$. For example $\mathrm{CG}_{tot} = 0.6136$ and 0.6073 at N = 3 and 8 iterations, respectively, and $\lim_{N\to\infty} \prod_{n=0}^{N-1} CG(n) = 0.6073$. Since $\mathrm{CG}_{tot}$ can be calculated and multiplied once after the matrix multiplication, conventional CORDIC iteration can be redefined as,

$$\begin{bmatrix} x'_{m+1} \\ y'_{m+1} \end{bmatrix} = \begin{bmatrix} 1 & -d(m)2^{-m} \\ d(m)2^{-m} & 1 \end{bmatrix}\begin{bmatrix} x'_m \\ y'_m \end{bmatrix}. \tag{2.11}$$

Hence, the output point equation can be written as,

$$\begin{bmatrix} x'_N \\ y'_N \end{bmatrix} = \left(\prod_{m=0}^{m=N-1}\begin{bmatrix} 1 & -d(m)2^{-m} \\ d(m)2^{-m} & 1 \end{bmatrix}\right)\begin{bmatrix} x_0 \\ y_0 \end{bmatrix} \tag{2.12}$$

$$\begin{bmatrix} x_N \\ y_N \end{bmatrix} = \mathrm{CG}_{tot}\begin{bmatrix} x'_N \\ y'_N \end{bmatrix}. \tag{2.13}$$

#### 2.1.2.1 Conventional CORDIC parameters selection criteria

**Number of iterations**

The number of iterations N is selected based on the required SQNR level. Since the reachable angle set $\phi$ is finite, it is impossible to represent the continuous range of the target angle $\theta$. Hence, the error vector $\boldsymbol{e}$ is defined as,

$$\boldsymbol{e} = \begin{bmatrix} x_e \\ y_e \end{bmatrix} = \begin{bmatrix} x_\infty \\ y_\infty \end{bmatrix} - \begin{bmatrix} x_N \\ y_N \end{bmatrix}.$$

A rule of thumb is to choose N = fixed-point length+2. Fig. 2.3 illustrates the error vector in conventional CORDIC at N = 3 and 8. Given that the target angle = 15°, using N = 3, the closest point to the target was 4.4°. However, when employing CORDIC wit N = 8, the error almost vanished. It is obvious that the error vector is getting smaller as N increases.

#### 2.1.2.2 Conventional CORDIC implementation

A typical hardware implementation of the conventional cordic iteration is shown in Fig. 2.4.

Thus, the conventional CORDIC algorithm reduces to a number of shift-add operations which is very attractive to hardware designers. The sign in each iteration is determined by $d(n)$ and it can be stored in a ROM for known rotations, as in FFT case. This simple form of CORDIC is valid for any angle, but it always requires N iterations, regardless of the angle of rotation. Although fixing the number of rotations simplify the design of the CORDIC, it results in a large overhead because the choice of the number of iterations is usually based on the worst case of all possible angles supported by the CORDIC.

### 2.1.3 MVR-CORDIC

In [8], the authors proposed that in some applications, where there is some information about the angle of rotation known beforehand, the number of iterations can be reduced. So we can perform the rotation using only M iteration, a subset of the full N iterations of Volder CORDIC. The following equations describe the algorithm,

$$\begin{bmatrix} x'_{m+1} \\ y'_{m+1} \end{bmatrix} = \mathrm{CG}(m) \begin{bmatrix} 1 & -d(m)2^{-t(m)} \\ d(m)2^{-t(m)} & 1 \end{bmatrix} \begin{bmatrix} x'_m \\ y'_m \end{bmatrix} \tag{2.14}$$

$$\mathrm{CG}(m) = \frac{1}{\sqrt{1+2^{-2t(m)}}} \tag{2.15}$$

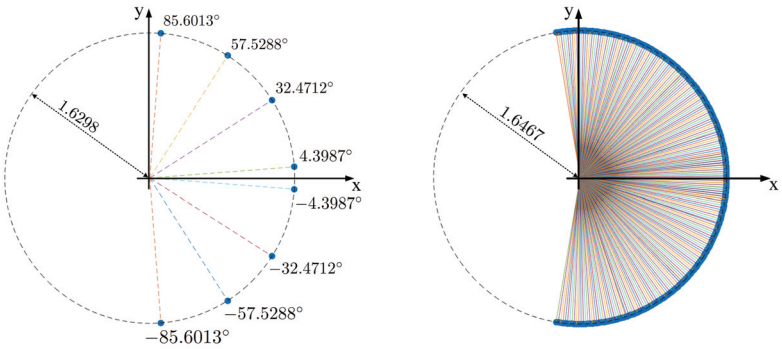


**Figure 2.2: Conventional CORDIC's reachable angles with, (a) 3 iterations, (b) 8 iterations.**

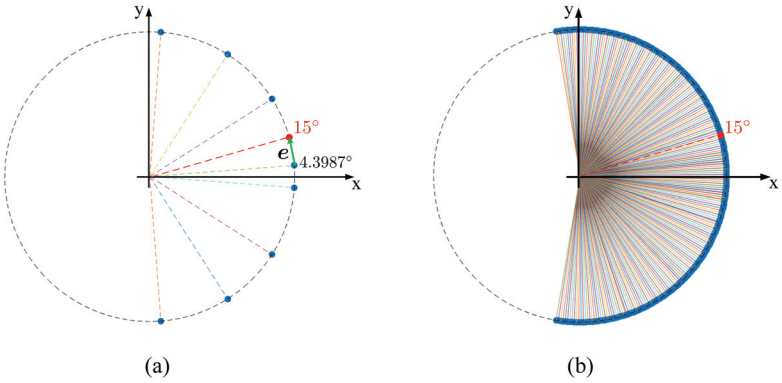


**Figure 2.3: Error vector *e* in conventional CORDIC with, (a) 3 iterations, (b) 8 iterations.**

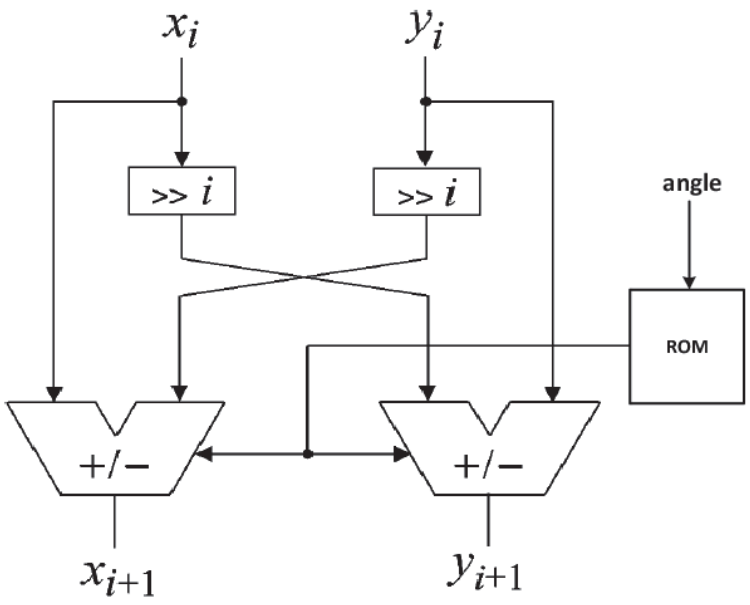


**Figure 2.4: implementation of Volder CORDIC stage, where ">> i" denotes right shift operation by i bits.**

where $t(m) \in [0, N-1]$. So $x' + jy'$ can be evaluated in M iterations by,

$$\begin{bmatrix} x'_M \\ y'_M \end{bmatrix} = \prod_{m=0}^{M-1} \begin{bmatrix} 1 & -d(m)2^{-t(m)} \\ d(m)2^{-t(m)} & 1 \end{bmatrix} \begin{bmatrix} x_0 \\ y_0 \end{bmatrix} \tag{2.16}$$

$$\begin{bmatrix} x_M \\ y_M \end{bmatrix} = \left( \prod_{m=0}^{M-1} \mathrm{CG}(m) \right) \begin{bmatrix} x'_M \\ y'_M \end{bmatrix} \tag{2.17}$$

where $M < N$. The parameters of MVR-CORDIC, $d(m)2^{-t(m)}$, are chosen to minimize the rotation residual error. The same SQNR, as conventional CORDIC, can be achieved using a smaller number of iterations because it is allowed to skip some micro-rotations and repeating others. But while performing the rotation in fewer iterations, this method introduces unequal CORDIC gain, which depends on the angle of rotation. Hence, this gain must be compensated with a factor that reverses the effect of this gain. This compensation was not necessary in the conventional CORDIC. A typical hardware implementation of MVR cordic stage is shown in Fig. 2.5.

Fig. 2.6 shows all the reachable points using MVR CORDIC algorithm using 4 iterations. In contrast to Volder CORDIC, not all the points are located on the same circle. We can compute an upper bound of the number of the reachable points, S, by this formula, $S = \begin{pmatrix} M+N-1 \\ N \end{pmatrix}$, where M is the number of reachable point in one iteration and can be computed as $M = 2 \times Number\,of\ shifts\,barrel\,shifter\,able\,to\,do + 1$.

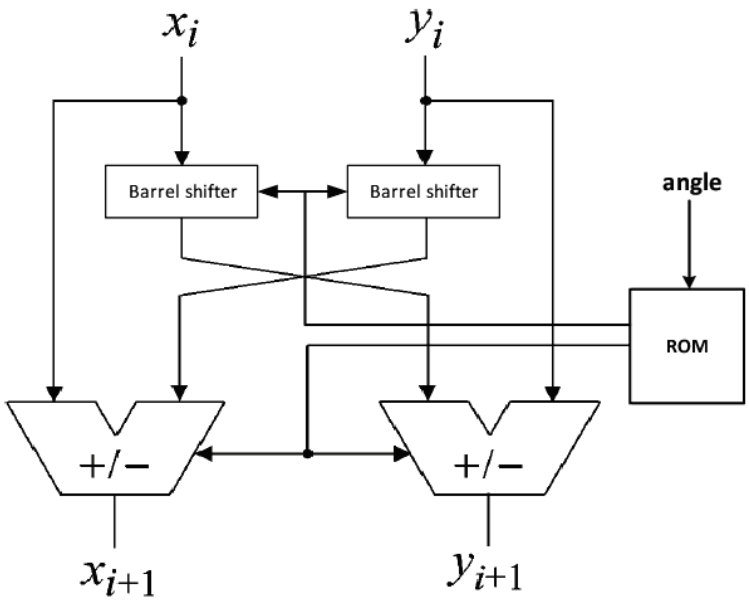


**Figure 2.5: implementation of MVR CORDIC stage**

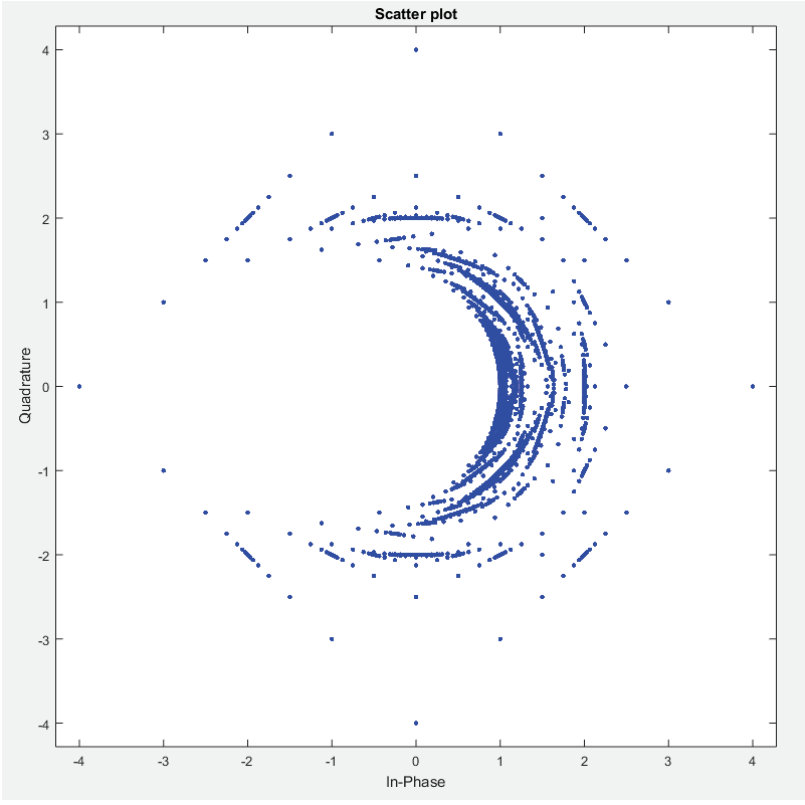


**Figure 2.6: MVR CORDIC constellation with 4 iterations**

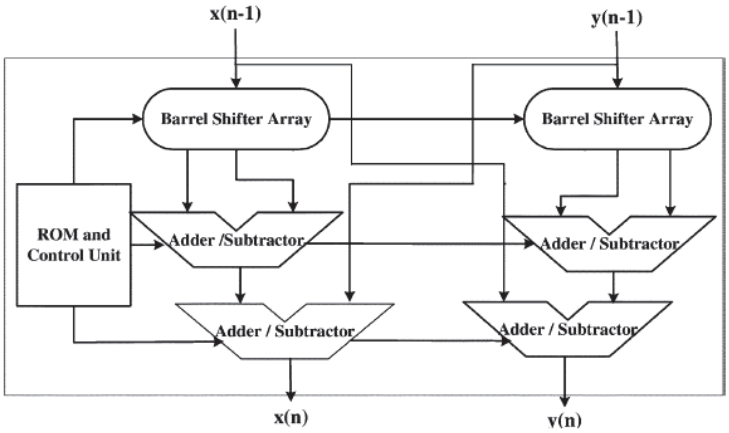


**Figure 2.7: implementation of EEAS CORDIC stage with $SPT_I = 2$**

### 2.1.4 EEAS-CORDIC

A method was proposed in [9] which increases angle space by constructing linear combinations of shifts and adds to realize $\tan(\theta)$ as follows,

$$\begin{bmatrix} x'_{m+1} \\ y'_{m+1} \end{bmatrix} = \mathrm{CG}(m) \begin{bmatrix} 1 & -SPT_I(m) \\ SPT_I(m) & 1 \end{bmatrix} \begin{bmatrix} x'_m \\ y'_m \end{bmatrix} \tag{2.18}$$

$$\mathrm{CG}(m) = \frac{1}{\sqrt{1 + \left( \sum_{i=1}^{K} d_i(m) 2^{-t_i(m)} \right)^2}} \tag{2.19}$$

$$SPT_I(m) = \sum_{i=1}^{K} d_i(m) 2^{-t_i(m)} \tag{2.20}$$

where $t(m) \in [0, N-1]$. So $x' + jy'$ can be evaluated in M iterations by,

$$\begin{bmatrix} x'_M \\ y'_M \end{bmatrix} = \prod_{m=0}^{M-1} \begin{bmatrix} 1 & -SPT_I(m) \\ SPT_I(m) & 1 \end{bmatrix} \begin{bmatrix} x_0 \\ y_0 \end{bmatrix} \tag{2.21}$$

$$\begin{bmatrix} x'_M \\ y'_M \end{bmatrix} = \left( \prod_{m=0}^{M-1} \mathrm{CG}(m) \right) \begin{bmatrix} x'_M \\ y'_M \end{bmatrix} \tag{2.22}$$

where $M < N$. The parameters of EEAS-CORDIC, $SPT_I(n)$, are chosen to minimize the rotation residual error. This method is faster than MVR-CORDIC and has a larger angle space, but it still introduces unequal gains. Hence, a compensation stage is needed. Even though [9] proposed a method to compensate for the CORDIC gain using the same EEAS-CORDIC structure, called "ET-II" scaling operation, it is still cumbersome. An example of hardware implementation of EEAS cordic stage is shown in Fig. 2.7.

Fig. 2.8 shows all the reachable points using EEAS CORDIC algorithm using 2 iterations.

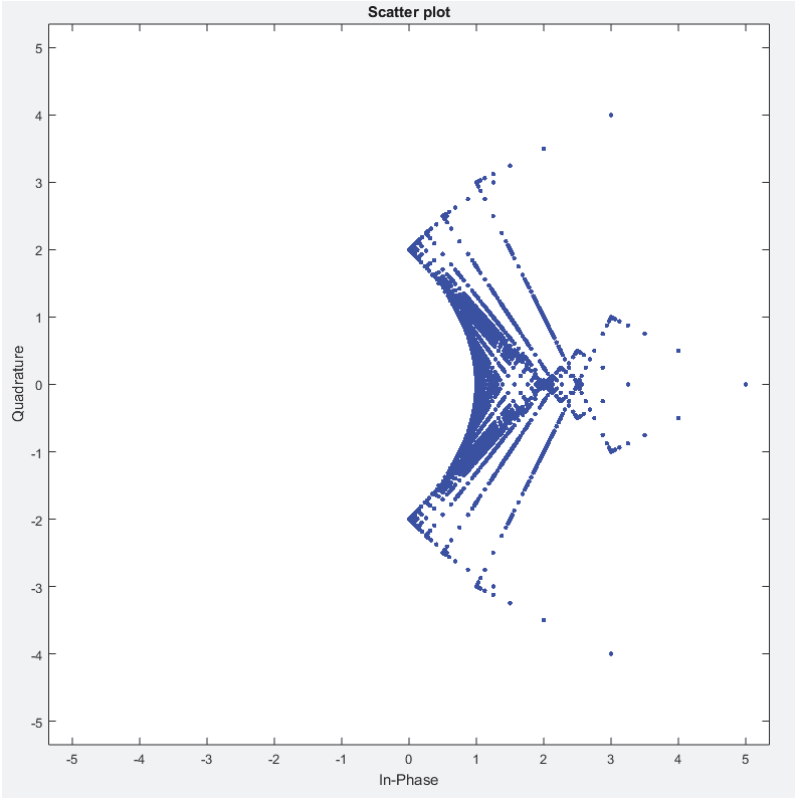


**Figure 2.8: EEAS CORDIC constellation with 2 iterations**

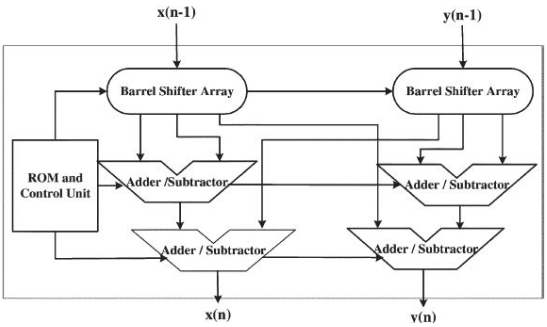


**Figure 2.9: implementation of MSR CORDIC stage with**$SPT_I = 2$ **and** $SPT_R = 1$

### 2.1.5 MSR-CORDIC

In [10], the authors proposed to mix the scaling phase with the rotation phase in order to remove the need for compensation as follows,

$$\begin{bmatrix} x'_{m+1} \\ y'_{m+1} \end{bmatrix} = \mathrm{CG}(m) \begin{bmatrix} SPT_R(m) & -SPT_I(m) \\ SPT_I(m) & SPT_R(m) \end{bmatrix} \begin{bmatrix} x'_m \\ y'_m \end{bmatrix} \tag{2.23}$$

$$\mathrm{CG}(m) = \frac{1}{\sqrt{\left(\sum_{j=1}^{L} d_j(m)2^{-r_j(m)}\right)^2 + \left(\sum_{i=1}^{K} d_i(m)2^{-t_i(m)}\right)^2}} \tag{2.24}$$

$$SPT_R(m) = \sum_{j=1}^{L} d_j(m)2^{-r_j(m)} \tag{2.25}$$

$$SPT_I(m) = \sum_{i=1}^{K} d_i(m)2^{-t_i(m)} \tag{2.26}$$

where $t(m)$ and $r(m) \in [0, N-1]$. So $x' + jy'$ can be evaluated by,

$$\begin{bmatrix} x'_M \\ y'_M \end{bmatrix} = \prod_{m=0}^{M-1} \begin{bmatrix} SPT_R(m) & -SPT_I(m) \\ SPT_I(m) & SPT_R(m) \end{bmatrix} \begin{bmatrix} x_0 \\ y_0 \end{bmatrix} \tag{2.27}$$

$$\begin{bmatrix} x_M \\ y_M \end{bmatrix} = \left(\prod_{m=0}^{M-1} \mathrm{CG}(m)\right) \begin{bmatrix} x'_M \\ y'_M \end{bmatrix} \tag{2.28}$$

where $M < N$. The MSR-CORDIC parameters are chosen in a way to have almost unity gain, hence, it removes the need for gain compensation. However, it has a detrimental effect on the SQNR of the result for the same number of iteration; since the parameters of MSR-CORDIC, $SPT_I(m)$ and $SPT_R(m)$, are chosen to minimize both the rotation and gain residual error, not only rotation residual error as in the previous techniques. An example of hardware implementation of MSR cordic stage is shown in Fig. 2.9.

Fig. 2.10 shows all the reachable points using MSR CORDIC algorithm using 2 iterations. In contrast to previous CORDICs, the reachable point can have gain lower than

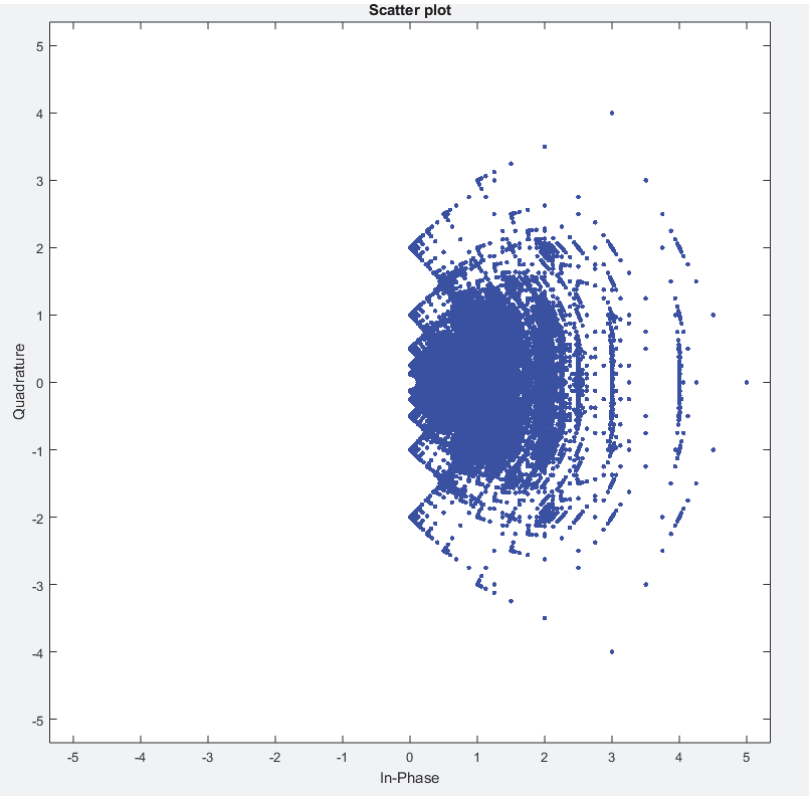


**Figure 2.10: MSR CORDIC constellation with 2 iterations, $SPT_I = 2$ and $SPT_R = 1$**

one. Therefore, designer can use this property to equalize the gain back to one by mixing iterations have gains larger than one with iterations have gains less than one.

## 2.2 Review of related FFT algorithms

Jean Joseph Fourier has described the relation between time and frequency domain in discrete form as

$$X(k) = \sum_{n=0}^{N-1} x(n) W_N^{nk} \tag{2.29}$$

where $X(k)$ is the discrete output signal in frequency domain, $x(n)$ is the discrete input signal in time domain, $n$ and $k = 0, 1, 2, ..., N-1$, and $W_N^{nk}$ is the twiddle factor which equals to $e^{-\frac{j2\pi nk}{N}}$. The complexity of the DFT equation is $O(N^2)$ and consequently an enormous difficulty faced its implementation on hardware.

In 1965, Cooley and Tukey described their algorithm to decompose the DFT equation and reduce its complexity. The radix-r FFT takes a DFT equation and divides it into

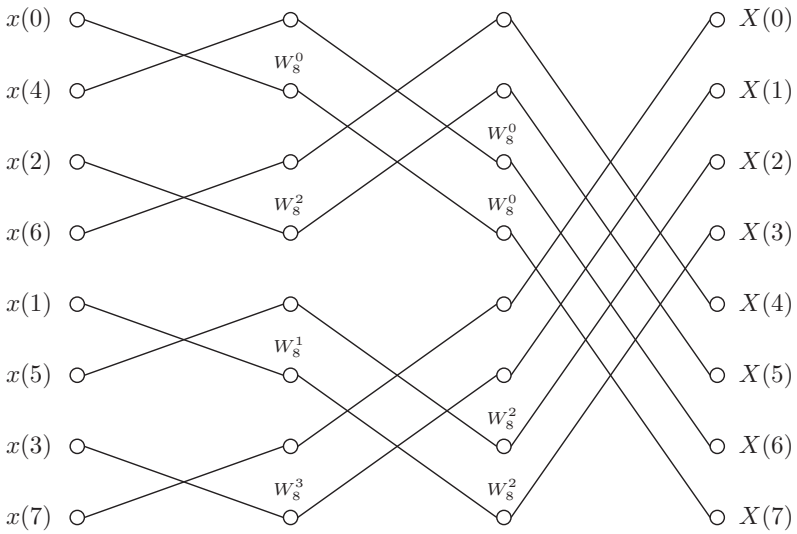


**Figure 2.11: 8 points conventional DIF radix-2 FFT**

successively smaller DFT's, allowing the process to be completed in $O(N\log(N))$ instead of $O(N^2)$ complexity. There are two types of radix-r FFT.

### 2.2.1 Decimation In Frequency (DIF)

In DIF, output sequence $X(k)$ is divided into smaller sub-sequences as shown,

$$X(r\kappa+\delta) = \sum_{\nu=0}^{r-1}\sum_{n=\nu N/r}^{((\nu+1)N/r)-1} x(n)W_N^{n(r\kappa+\delta)} \tag{2.30}$$

$$X(r\kappa+\delta) = \sum_{\nu=0}^{r-1}\sum_{\mu=0}^{(N/r)-1} x(\mu+\nu N/r)W_N^{(n+\nu N/r)(r\kappa+\delta)} \tag{2.31}$$

$$X(r\kappa+\delta) = \sum_{\mu=0}^{(N/r)-1}\left(\sum_{\nu=0}^{r-1} W_r^{\nu\delta}x(\mu+\nu N/r)\right)W_N^{n\delta}W_{N/r}^{n\kappa} \tag{2.32}$$

where r is the radix order, $\kappa$ and $\mu = 0, 1,2, ..., (N/r)-1$, $\delta$ and $\nu = 0,1, ..., r-1$. Therefore r DFTs of $\frac{N}{r}$ points is obtained and can be further decomposed into $r^2$ DFTs of $\frac{N}{r^2}$ points. After $m$ times, $m = log_r N$, recursive decomposition, the complete radix-r DIF FFT algorithm can be obtained. Fig. 2.11 illustrate 8-point radix-2 DIF FFT Signal Flow Graph (SFG) where the input is in digit-reversed radix-2 order.

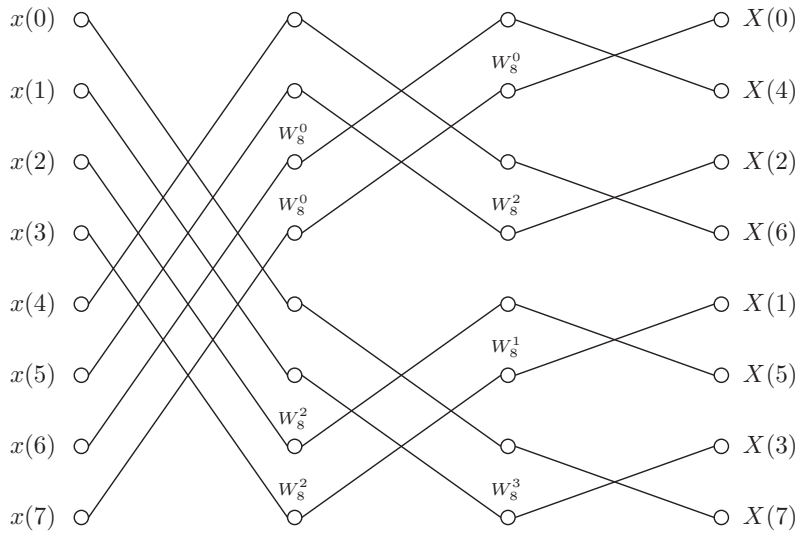


**Figure 2.12: 8 points conventional radix-2 FFT**

### 2.2.2 Decimation In Time (DIT)

In DIT, input sequence $x(n)$ is divided into smaller sub-sequences as shown,

$$X(k) = \sum_{\delta=0}^{r-1} \sum_{\mu=0}^{(N/r)-1} x(r\mu+\delta) W_N^{k(r\mu+\delta)} \tag{2.33}$$

$$X(k) = \sum_{\delta=0}^{r-1} W_N^{k\delta} \sum_{\mu=0}^{(N/r)-1} x(r\mu+\delta) W_{N/r}^{k\mu} \tag{2.34}$$

$$X(\kappa+\nu N/r) = \sum_{\delta=0}^{r-1} W_r^{\nu\delta} W_N^{\kappa\delta} \sum_{\mu=0}^{(N/r)-1} x(r\mu+\delta) W_{N/r}^{\kappa\mu}. \tag{2.35}$$

Therefore r DFTs of $\frac{N}{r}$ points is obtained and can be further decomposed into $r^2$ DFTs of $\frac{N}{r^2}$ points. After $m$ times recursive decomposition, the complete radix-r DIT FFT algorithm can be obtained. Fig. 2.12 illustrate 8-point radix-2 DIF FFT SFG where the output is in digit-reversed radix-2 order.

## 2.3 Review of related FFT hardware architecture

Since the rotation angles used in the computations are known beforehand in the FFT, these rotations can be computed using the CORDIC algorithm which provide the same

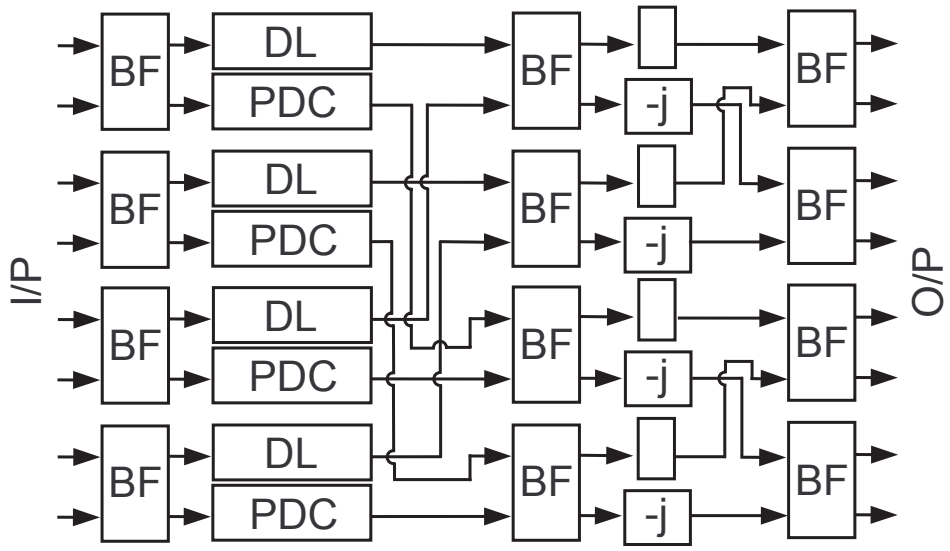


**Figure 2.13: CORDIC parallel-pipeline FFT architecture**

throughput as a complex multiplier while using less silicon area [22]. The coefficients required by the CORDIC algorithm can be pre-calculated and stored in a ROM.

As in [22, 23], ASIC pipeline FFT architectures are divided, in general, into fully parallel-pipeline, fully serial-pipeline, and hybrid parallel-serial-pipeline. The fully parallel-pipeline architecture achieves maximum throughput, minimum latency, but maximum area. Fig. 2.13 shows fully parallel-pipeline architecture where BF, PDC and DL denote to Butterfly block, pipeline dedicated CORDIC block and delay line block, respectively.

Fully serial-pipeline architecture achieves high throughput, smaller area, but relatively high latency. Fig. 2.14 shows Radix-2 Single-Path Delay Feedback (R2SDF) pipeline architectures. There is another type of pipeline architecture called Radix-2 Multiple-Path Delay Commutator (R2MDC); but this type is not usually used since it has bad memory utilization, about 50%.

Hybrid parallel-serial-pipeline architectures try to compromise between the area advantage of the fully serial-pipeline and the throughput advantage of the fully parallel-pipeline by merging the two architecture. Hybrid architecture can be described by $\ell$ which is the size of the used serial architecture. Fig. 2.16 shows a hybrid(8) architecture to compute 64 point FFT with throughput eight times the serial architecture.

Memory-based architecture achieve the smallest area but minimum throughput and high latency. It has many architectures, one of them is shown in Fig. 2.17.

The Latency $T_d$ is decomposed to 3 terms,

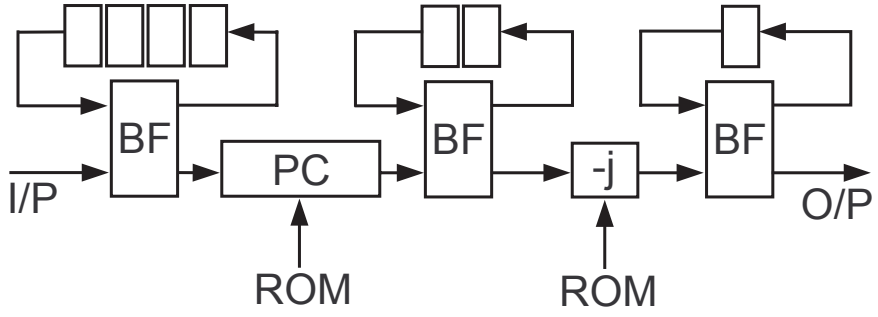


**Figure 2.14: CORDIC R2SDF FFT architecture**

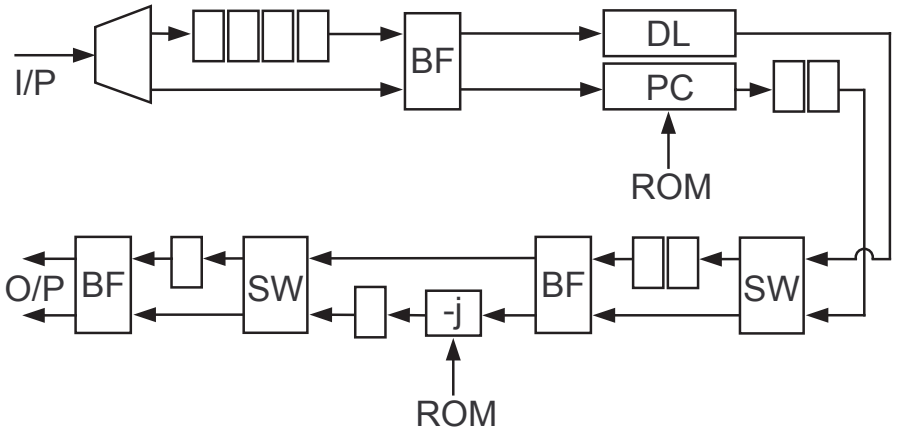


**Figure 2.15: CORDIC R2MDF FFT architecture**

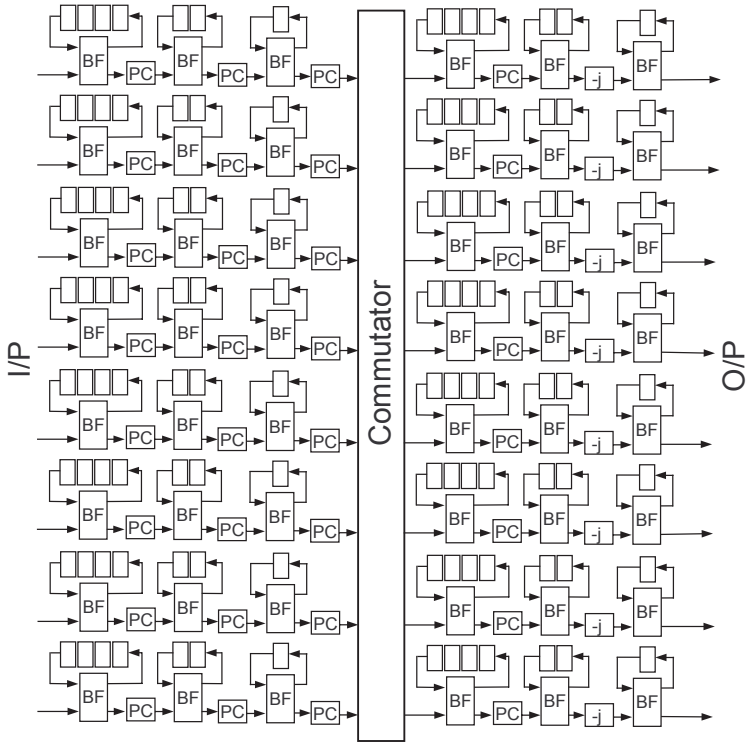


**Figure 2.16: CORDIC Hybrid parallel-serial-pipeline FFT architecture**

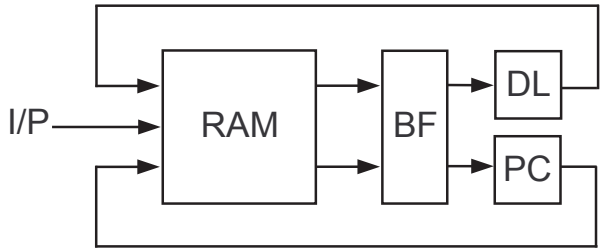


**Figure 2.17: CORDIC memory-based FFT architecture 2**

**Table 2.1: Hardware performance comparison between different FFT architectures**

| FFT Architecture | Latency (clk) | Throughput | Adders (count) |
|---|---|---|---|
| Parallel-pipeline | $(log_2(N)-2)T_{PC}+log_2(N)$ | $N$ | $\frac{N}{2}((log_2(N)-2)A_{PC}+4log_2(N))$ |
| R2SDF | $(log_2(N)-2)T_{PC}+log_2(N)+N-1$ | 1 | $(log_2(N)-2)A_{PC}+4log_2(N)$ |
| Hybrid($\ell$) | $(log_2(\ell)T_{PC}+log_2(\ell)+\ell-1)\log_\ell(N)-2T_{PC}$ | $\frac{N}{\ell}$ | $(\frac{N}{\ell}(log_2(\ell)A_{PC}+4log_2(\ell)))\log_\ell(N)-2\frac{N}{\ell}A_{PC}$ |
| MBFFT | $T_{PC}+\frac{N}{2}(log_2(N)-1)+\frac{N}{2}$ | $\frac{N}{T_{PC}+\frac{N}{2}log_2(N)}$ | $4+A_{PC}$ |

$$T_d = T_W + T_B + T_{buf} \tag{2.36}$$

where$T_w$ is the time required for the rotation operations, $T_B$ is the time required for the butterfly operations and $T_{buf}$ is the buffering time (which equals to $N\sum_{i=1}^{log_2 N}2^{-i}=N-1$). These delays belong to system level. Therefore, the interconnection delays are omitted Here. $T_{PC}$ and $A_{PC}$ is the number of clocks and adders required to perform a certain type CORDIC rotation, respectively. A comparison is done between the different architectures using conventional radix-2 FFT in table 2.1 where $T_{PC}$ and $A_{PC}$ are the number of clocks and adders required to perform a certain type CORDIC rotation, respectively. Throughput is calculated assuming the same critical path and its unit is symbol/clk.

# Chapter 3

# The Proposed Radix-r FFT: HardWare-Friendly FFT (HW-F FFT)

In this chapter, we are going to explain the general concept behind the proposed FFT algorithm. First, we will start with radix-2 topology in order to help the reader to digest the idea easily. Then, a generalization of radix-r HW-F FFT is drawn later.

## 3.1 Motivation

A thorough scrutiny of the pipeline FFT architectures shows some potential opportunity that can be exploited to enhance the area-time-power product. Those potential opportunities are summarized in the following list.

**Increasing the utilization of SDF FFT architectures**

The Single-path Delay Feedback (SDF) architecture, explained in chapter 2, consists of $\log_r(N)$ stages. Broadly, each of which has a Twiddle Factor Rotator (TWR) that can be implemented using pipeline CORDIC or complex multiplier. Each output points of a certain stage must pass through the TWR block even if it does not need to be rotated by an angle. Hence, TWRs are not working with their full capacity in SDF architecture. This raises a question: is there a way to use TWR with their full capacity, and in return the TWR increases the output points SQNR?

**Increasing the parallelism capability of fully-parallel FFT architectures**

The fully parallel-pipeline consists also of $\log_r(N)$ stages. Each stage consists of $N/r$ butterfly operation, where r is the radix of the stage. Each butterfly has r output; $r-1$ of them is followed by dedicated TWR and the remaining one is followed by a DL in seek of synchronization. DL must delay depth equals to the parallel TWR. Consequently, the faster points should wait for the slower ones. Hence, the time required to finish one FFT

is determined by the slowest path between input and output. This raises a question: can the fast paths lift some computational burdens from the slow paths in seek of speeding them up?

**CORDIC gain issue for multiplier-less designs**

Due to the used platform limitation or design restrictions, the multiplier block may not be allowed to use except in the most limited context. In such platforms, CORDIC is suitable design choice for the TWR. However, to avoid performing constant multiplication after each CORDIC rotation, FFT implementations use conventional or MSR-CORDIC [10]. Even that, MSR-CORDIC may have a complex design that is not preferred in the high frequency designs. This raises a question: is there a way to avoid using CORDIC gain compensation after each TWR implemented using unequal-gain CORDIC types?

On the light of the previous question marks, this work introduces the HardWare-Friendly FFT (HW-F FFT). The proposed algorithm addresses these question marks by modifying the FFT algorithm itself rather than modifying a specific FFT hardware architecture. First, we will try to solve the last question in radix-2, then we will generalize for radix-r FFT.

## 3.2 The Proposed Radix-2 HW-F FFT

### 3.2.1 The problem statement

How to use the unequal-gain CORDIC in radix-2 FFT without compensate the gain after each CORDIC?

### 3.2.2 the proposed solution

The proposed algorithm suggests to reorganize the rotations in the conventional radix-2 FFT such that the two butterfly operands are multiplied by twiddle factors having the same CORDIC gain. The reorganization procedure is done according to the following observations,

- the sign of an angle does not affect the CORDIC gain as in equations (2.15), (2.19) and (2.24). Therefore, CORDIC gain is always the same for both a certain angle and its negative;
- butterfly operation is a linear operation. Fig. (3.1, a) shows a butterfly with its original twiddle factors: $W_{up} = 1$ and $W_{down} = e^{j\phi}$. Fig. (3.1, b) shows the modified twiddle factors: $W_{up} = Ke^{j\theta}$ and $W_{down} = Ke^{j(\phi+\theta)}$. These twiddle factors were chosen to maintain the same difference of the old twiddle factors in Fig. (3.1, a).

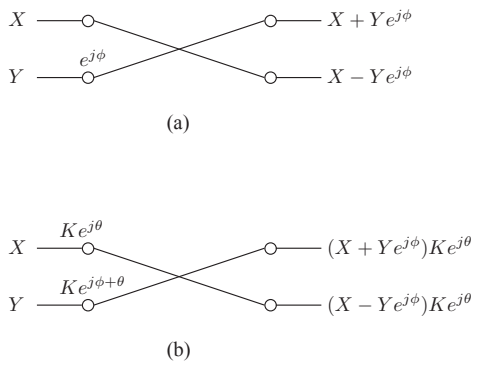


**Figure 3.1: Butterfly operation is sensitive only to the phase difference**

Hence, due to the CORDIC linearity, the output of (b) will equal to the output of (a), except of a fixed complex scale factor;

- multiplying all the points of Discrete Fourier Transform (DFT) in time domain by a constant is equivalent to multiplying the DFT's points in frequency domain by the same constant.

This thesis presents two main approaches to explain the modified radix-2 FFT algorithm: an analytic approach using equations and a graphical approach using Signal Flow Graph (SFG). Note that, in the analytical approach, CORDIC gain is omitted for simplicity.

The key idea of the proposed algorithm is in the way of synthesizing the twiddle factors via CORDIC algorithm. In conventional FFT, each twiddle factor belonging to a single butterfly is synthesized independently using CORDIC. Fig. (3.2, a) presents the complex domain with unit circle and points A and B are the target twiddle factors before a butterfly.

In conventional FFT with conventional CORDIC, CORDIC algorithm is optimized such that it finds a point with the same phase. Meanwhile, the conventional CORDIC guarantee constant gain equals to 1.647 for any target angle. This solution consumes a lot of hardware resources due to the inefficiency of the conventional CORDIC.

In conventional FFT using unequal-gain CORDIC, CORDIC algorithm is optimized such that it finds a point with the same phase, even without the same amplitude. When CORDIC settles on point for each twiddle factor, i.e. B''' for B, a compensation stage takes place to retrieves the unit amplitude instead of K1. The drawbacks of this method are

1- the compensation stage that requires additional multipliers for both real and imaginary numbers;

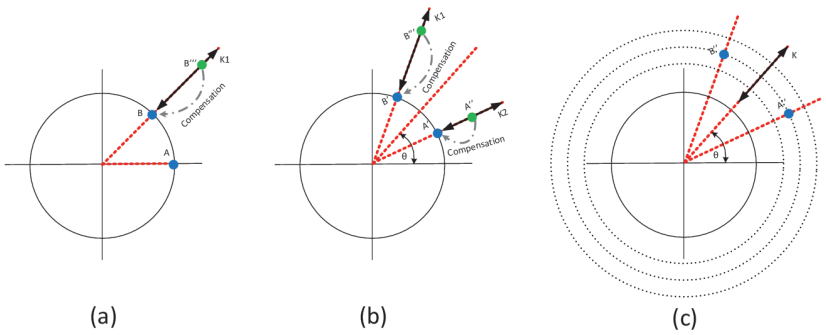


**Figure 3.2: CORDIC optimization methods between the conventional and the proposed algorithm.**

2- the inefficiency in using the CORDIC angles domain where the only degree of freedom used is gain, i.e. K1.

In the first version of the proposed FFT, the algorithm tries to mitigate the second drawback which will lead to higher resolution without increasing the number of iterations. The twiddle factors belonging to a single butterfly are synthesized jointly using CORDIC. In Fig. (3.2, b), instead of comply with the original points A and B, CORDIC algorithm tries to find any two points in the complex domain which have the same relative phase between them. Therefore, the common phase $\theta$ can be transferred after the butterfly operation to deal with it later. When CORDIC settles on point for each twiddle factor, i.e. A" for A and B"' for B, a compensation stage takes place to retrieves the unit amplitude instead of K1 and K2.

The second version of the proposed FFT tries to tackle the two drawbacks all together. It is much the same as the first version, however it tries to maintain the same relative phase and gain, even if not the unit gain, as shown in Fig. (3.2, c). Therefore, the common gain K and phase $\theta$ can be transferred after the butterfly operation to deal with them later. In the later version there is no need to compensation stage after each CORDIC stage, and instead, it is sufficient to compensate all the gains once at the end of the FFT as will be shown.

In Decimation In Time (DIT), input sequence $x(n)$ is divided into smaller subsequences. With dividing it into even and odd sequences, the DFT equation reaches to the following,

$$X(k) = \sum_{r=0}^{\frac{N}{2}-1} x(2r) W_N^{2rk} + W_N^k \sum_{r=0}^{\frac{N}{2}-1} x(2r+1) W_N^{2rk} \tag{3.1}$$

where $W_N^{2rk} = e^{-j2\pi\frac{2rk}{N}}$ is the twiddle factor. With $g(r) = x(2r)$ and $h(r) = x(2r+1)$, the proposed algorithm suggests to modify the twiddle factors as following,

$$X(k) = W_N^{\frac{k}{2}} \left( W_N^{-\frac{k}{2}} \sum_{r=0}^{\frac{N}{2}-1} g(r) W_{\frac{N}{2}}^{rk} + W_N^{\frac{k}{2}} \sum_{r=0}^{\frac{N}{2}-1} h(r) W_{\frac{N}{2}}^{rk} \right). \tag{3.2}$$

Each twiddle factor pair belonging to the next butterfly operand pair, namely $W_{up}$ and $W_{down}$, is decomposed into two parts:

- Modified Twiddle Factor (*MTF*): These are the modified twiddle factors that will by applied during the butterfly operation. The twiddle factors are chosen in a way to keep the same phase difference as the original twiddle factors. The new twiddle factors are related to the old ones as follows,

$$MTF_{up} = \sqrt{\frac{W_{up}}{W_{down}}} \tag{3.3}$$

$$MTF_{down} = \sqrt{\frac{W_{down}}{W_{up}}}. \tag{3.4}$$

- Propagating Twiddle Factor (*PTF*): As stated before, the new twiddle factors have the same phase difference as the old ones, hence, the output will be the same as the old output, but scaled with a complex scale factor. This complex scale factor is calculated as,

$$PTF_{up,down} = \sqrt{W_{up} W_{down}}. \tag{3.5}$$

Fig. 3.3 illustrates the transition of first stage in conventional 8 points radix-2 FFT to the modified one, where the circles filled with same pattern are inputs to the same butterfly in the next stage. Twiddle factors between brackets are *PTF*s and the rest are the *MTF*s. '*A*' is the CORDIC gain associated with each rotation and it has the same subscript and superscript as its generating twiddle factor. Mapping equations (3.3), (3.4) and (3.5) to Fig. 3.3, if$W_{up}$ and $W_{down}$ are $W_8^0$ and $W_8^2$ in the fifth and seventh FFT nodes at the first stage, then $MTF_{up}$, $MTF_{down}$, $PTF_{up}$ and $PTF_{down}$ are $W_8^{-1}$, $W_8^1$, $W_8^1$ and $W_8^1$, respectively. PTFs are the same for all sub FFTs inputs, where the input of the next sub FFT are shown in the dotted rectangles in Fig. 3.3.

Each of $g(r)$ and $h(r)$ in equation (3.2) represents the input points for the $\frac{N}{2}$ points DFTs, and are handled as the original input $x(n)$. For example, to generate DFT output $G(k)$ of even input points, namely $g(r)$, it will be decomposed into,

$$G(k) = W_{\frac{N}{2}}^{\frac{k}{2}} \left( W_{\frac{N}{2}}^{-\frac{k}{2}} \sum_{p=0}^{\frac{N}{4}-1} g'(p) W_{\frac{N}{4}}^{pk} + W_{\frac{N}{2}}^{\frac{k}{2}} \sum_{p=0}^{\frac{N}{4}-1} h'(p) W_{\frac{N}{4}}^{pk} \right) \tag{3.6}$$

where $g'(p) = g(2p)$ and $h'(p) = g(2p+1)$. Fig. 3.4 illustrates the transition of second stage toward the modified algorithm where the input of the next sub FFT are shown in the dotted rectangles.

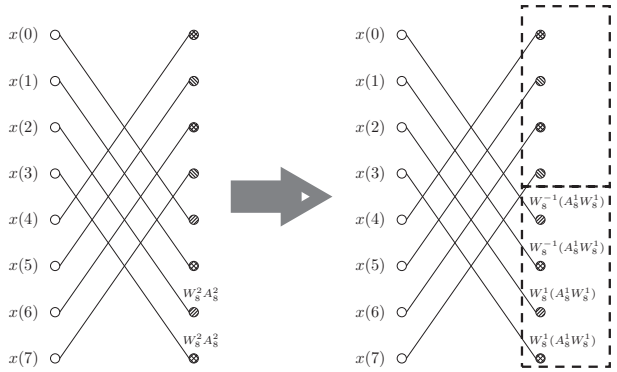


**Figure 3.3: The transition of first stage in conventional radix-2 FFT to modified one**

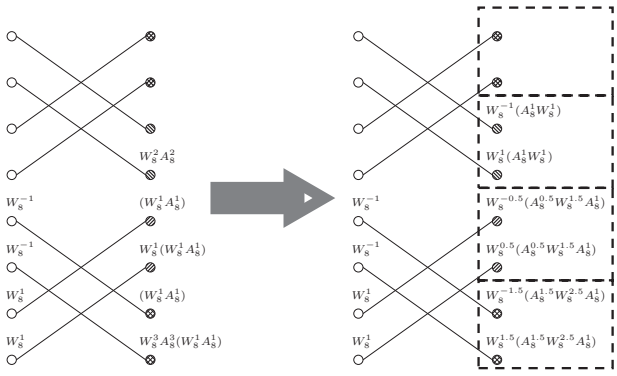


**Figure 3.4: The transition of second stage toward the modified algorithm**

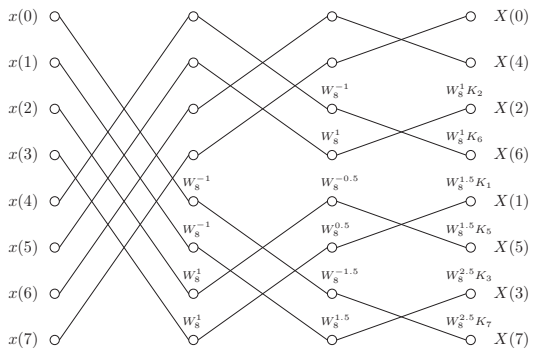


**Figure 3.5: 8 points modified DIT radix-2 FFT**

After the last stage, the scale factors are merged into one complex scale factor for each output point. This scale factor can be easily compensated. Moreover, if the FFT is used in a communications system, the scaling factors can be compensated in the channel inversion stage. Fig. **??** shows complete SFG for 8 points modified radix-2 FFT.

### 3.2.3 Exploiting more solution space

The above analysis has not exploited all the possible modifications to the twiddle factors, since the new twiddle factors phase are constrained to an angle and its negative while keeping the old phase difference between the angles used in the butterflies. However, there maybe other alternatives to choose new twiddle factors, while maintaining the same CORDIC gain. To illustrate this, Fig. 3.6 redraw the SFG of the modified 8 point FFT by replacing the twiddle factors with the rotation angles in degree. In the first stage we have rotations by 45° and −45° whose difference equals to 90°. Those angles , namely 45° and −45°, needs one CORDIC iteration to synthesize them. However, If 45° and −45° are replaced with 0° and −90°, the following is observed,

1. 0° and −90° have the same CORDIC gain;
2. they produced $PTF_{up,down} = W_8^1$ transferred to the output of its sub FFT, shown in the dotted rectangles in Fig. 3.6;
3. they can be synthesized by multiplying by 1 and $-j$, which needs no CORDIC iterations.

Fig. 3.7 illustrates the previous steps on 8 points FFT where the angles written in degrees. Generally, This will apply to larger FFT sizes. Hence, the number of complex rotations in the whole FFT without equalization is $N(\log_2(N)-3)+4$ instead of $N(\log_2(N)-2)+2$. Thus, the synthesis is more efficient.

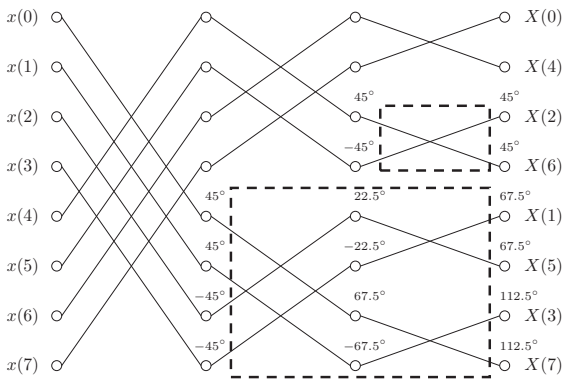


**Figure 3.6: Rotations of 8 point Modified Radix-2 FFT in degree**

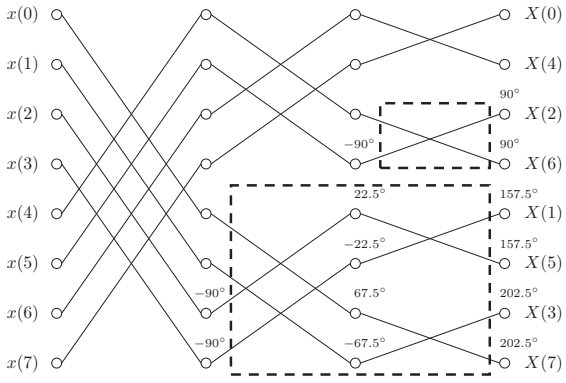


**Figure 3.7: First stage optimization in the Modified Radix-2 FFT**

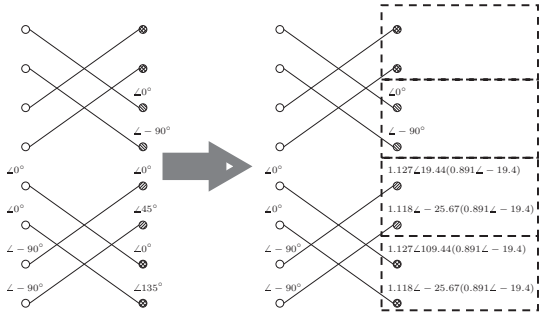


**Figure 3.8: The transition of second stage toward the modified algorithm**

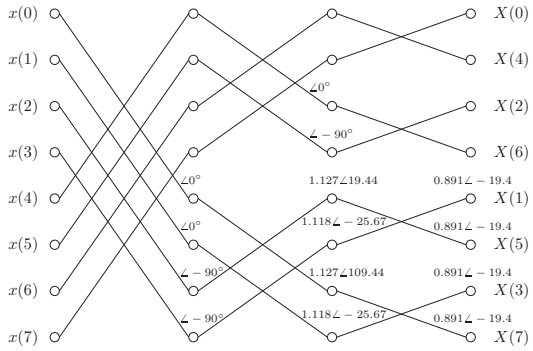


**Figure 3.9: 8 points modified DIT radix-2 FFT**

Therefore, we can use a simulation program to Fig. out what are the best modified and propagated twiddle factors for a specific butterfly according to objective function. Fig. 3.8 illustrates the transition of second stage in conventional 8 points radix-2 FFT toward the modified algorithm. The CORDIC used to synthesize the twiddle factors is 2-iterations MVR CORDIC algorithm. Twiddle factors between parentheses are *PTF*s and the rest are the *MTF*s. PTFs are the same for all sub FFTs inputs, where the input of the next sub FFT are shown in dotted rectangles in Fig. 3.8.

After the last stage, the scale factors are merged into one complex scale factor for each output point. This scale factor can be easily compensated. Moreover, if the FFT is used in a communications system, the scaling factors can be compensated in the channel inversion stage. Fig. 3.9 shows complete SFG for 8 points modified radix-2 DIT FFT.

Similarly for the radix-2 DIF FFT, the proposed algorithm suggests to modify the twiddle factors as previous; but instead of propagate PTFs to the end of the FFT , it will

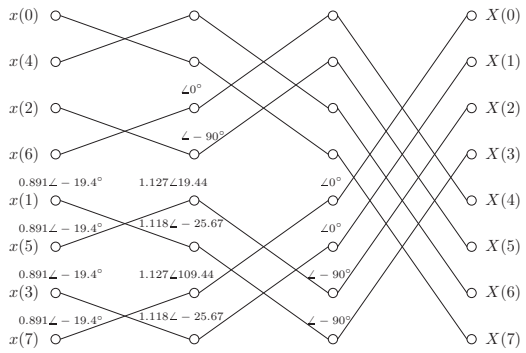


**Figure 3.10: 8 points modified DIF radix-2 FFT**

propagate them to the beginning of the FFT. Fig. 3.10 illustrate complete SFG for 8 points modified radix-2 DIF FFT.

This method can be applied to any length of FFT. The angle needed in each stage is independent of the FFT length and only depends on the stage ($m$). Although the number of complex rotations in the whole FFT is increased, each CORDIC rotation does not need to compensate its gain. Therefore, more efficient CORDIC types can be used to synthesize the complex rotations in the FFT, even if suffers from variable CORDIC gain. Excluding the last stage, the number of rotations increased from $\frac{N}{2}(\log_2(N)-1)$, in the conventional radix-2 FFT, to $N(\log_2(N)-3)+4$, in the modified radix-2 FFT.

## 3.3 Greedy Radix-r FFT

To extend the previous work, which has been published in [**CORDIC-FRIENDLY**], radix-r HW-F FFT changes the structure of the radix-r butterfly to create a new degree of freedom. In Fig. 3.11, $X_\nu$ and $Y_\mu$ represent the butterfly inputs and outputs, respectively, where $\nu$ and $\mu \in [0,1,...,r-1]$. The outputs of radix-r butterfly are governed in the following equation,

$$Y_\mu = \sum_{\nu=0}^{r-1} W_r^{\nu\mu} X_\nu e^{j\nu\phi} \tag{3.7}$$

where $e^{j\nu\phi}$ are the twiddle factors attached with the butterfly and $W_r^{\nu\mu} = e^{-\frac{j2\pi\nu\mu}{r}}$. The proposed algorithm introduces the complex scale $Ke^{j\theta}$ as shown in Fig. 3.11 (b). Therefore, instead of being bound to the original twiddle factors, optimization can be carried out to choose more suitable twiddle factors that will minimize the design area, power, or latency.

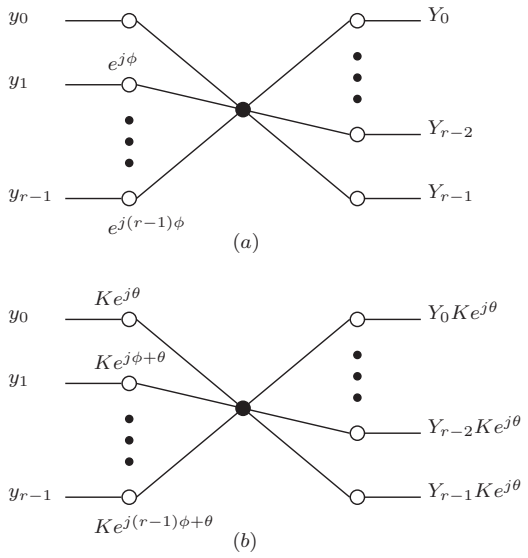


**Figure 3.11: Radix-r HWF FFT degree of freedom**

The key idea of the proposed algorithm is in the way of synthesizing the twiddle factors via CORDICs. Fig. 3.12 illustrates the optimization methodology of the proposed FFT algorithm. The blue small dots in the Fig. present the reachable CORDIC points. As an example, we chose 2-iteration MVR CORDIC [8]. The twiddle factors needed to be synthesized are presented by black points located on the intersection between the green arms and the red circle. They are named as $A_0$, $A_1$, ..., $A_{r-1}$ which are equivalent to $Ke^{j\theta}$, $Ke^{j\phi+\theta}$, ..., $Ke^{j(r-1)\phi+\theta}$ in Fig. 3.11 (b).

In the conventional radix-r FFT, the optimization methodology fixes $A_0$ to 1 and looks for the nearest feasible CORDIC point for the remaining twiddle factors, i.e. $A_1$ to $A_{r-1}$. This methodology constrains the twiddle factors to be located on the unit circle and in fixed angular positions. However in the greedy radix-r HW-F FFT, the optimization methodology looks for r feasible points that are $\phi$ radian apart from their neighbor and located on the same circle. The points that satisfy these conditions the best are the ones which minimize the quantization error, as will be shown in section 4.
One way to imagine this is that the optimization methodology searches for these points by

- expanding and contracting the red circle along with rotating the green arms,
- then collecting the nearest feasible points to the intersections,
- and finally choosing the best candidates.

By this way, the proposed radix-r FFT algorithm increases the number of candidate points significantly, and hence it is more probably to find better points to synthesis the target twiddle factors.

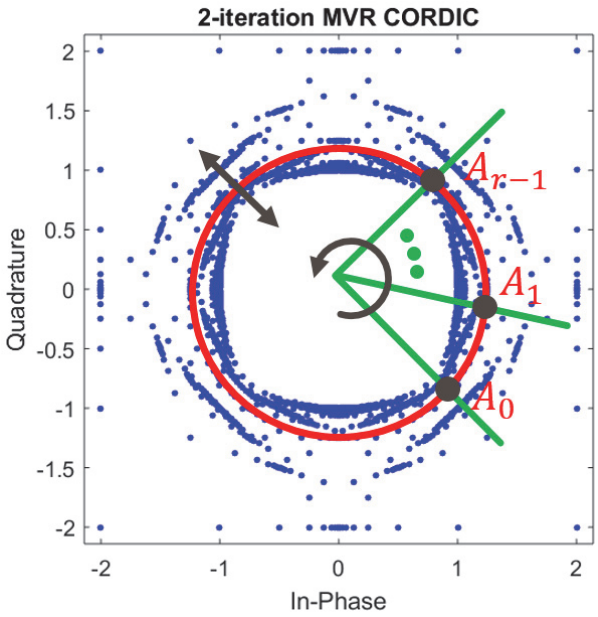


**Figure 3.12: Optimization methodology of the greedy radix-r HW-F FFT**

Keeping track with [**CORDIC-FRIENDLY**]'s notations, $Ke^{j\theta}$, $Ke^{j\phi+\theta}$,..., $Ke^{j(r-1)\phi+\theta}$ are called Modified Twiddle Factors (MTFs) and $Ke^{j\theta}$ multiplied by all the butterfly's outputs is called Propagating Twiddle Factor (PTF). MTFs are physically realized at each stage using CORDIC. On the other hand, PTFs generated from each butterfly propagate to the last stage, merge with the other PTFs after the last FFT stage. Then these scale factors can be easily compensated all once at the end.

The capability of moving the PTFs to the last FFT stage is available because of the way FFT Signal Flow Graph (SFG) is constructed and how the PTFs are chosen. To illustrate this, a 16-point radix-2 FFT is considered as an example where its SFG is presented in Fig. 3.13, where the twiddle factor $W_N^n = e^{\frac{-j2\pi n}{N}}$. The four butterflies a, b, c and d are belonging to the same butterfly group since the same twiddle factors are attached to their inputs, i.e. 1 and $W_{16}^4$. Fig 3.14 focuses on these butterflies by giving each one of them different color and omitting all the other twiddle factors for simplicity. The input of these butterflies are $z_0$, $z_1$,..., $z_7$, their attached twiddle factors, i.e. the MTFs, are $K_0e^{\psi_0}$, $K_1e^{\psi_1}$,..., $K_7e^{\psi_7}$, and their outputs are $y_0P_a$, $y_1P_b$,..., $y_7P_d$. $P_a$, $P_b$, $P_c$ and $P_d$ are the PTFs. If the four butterflies have been optimized differently, their MTFs will be different and hence their PTFs. Then, it is impossible to moving these PTFs to the last stage. However, since the are belonging to the same butterfly group, they will be optimized similarly. As a result, $K_0e^{\psi_0} = K_1e^{\psi_1} = K_2e^{\psi_2} = K_3e^{\psi_3}$, $K_4e^{\psi_4} = K_5e^{\psi_5} = K_6e^{\psi_6} = K_7e^{\psi_7}$ and $P_a = P_b = P_c = P_d$.

Hence the PTFS $P_a$, $P_b$, $P_c$ and $P_d$ can be moved to the last stage since the next operations for them are linear till the end of the FFT. Similarly, all the PTFs from the other butterflies group are transferred to the last stage in order to be compensated.

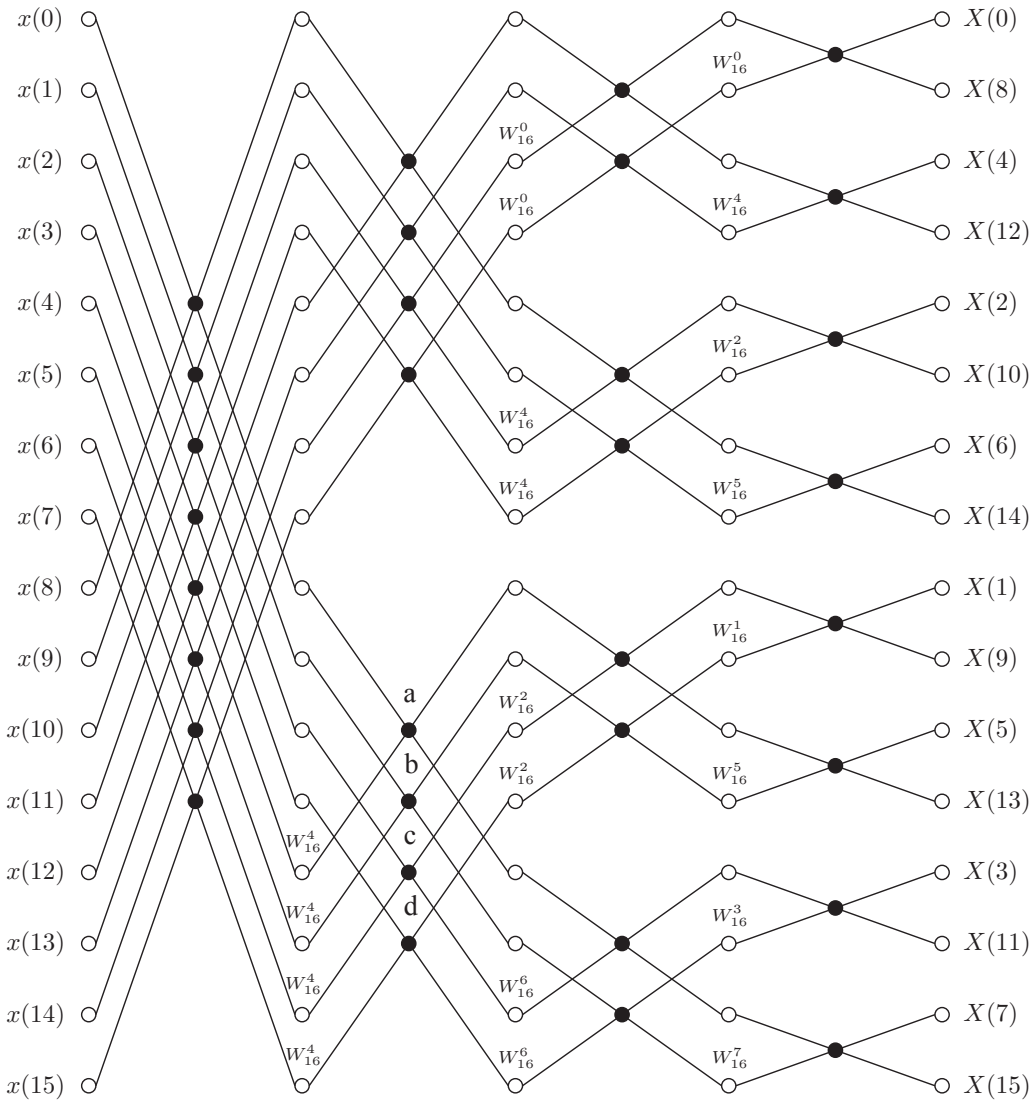


**Figure 3.13: 16-point radix-2 FFT SFG.**

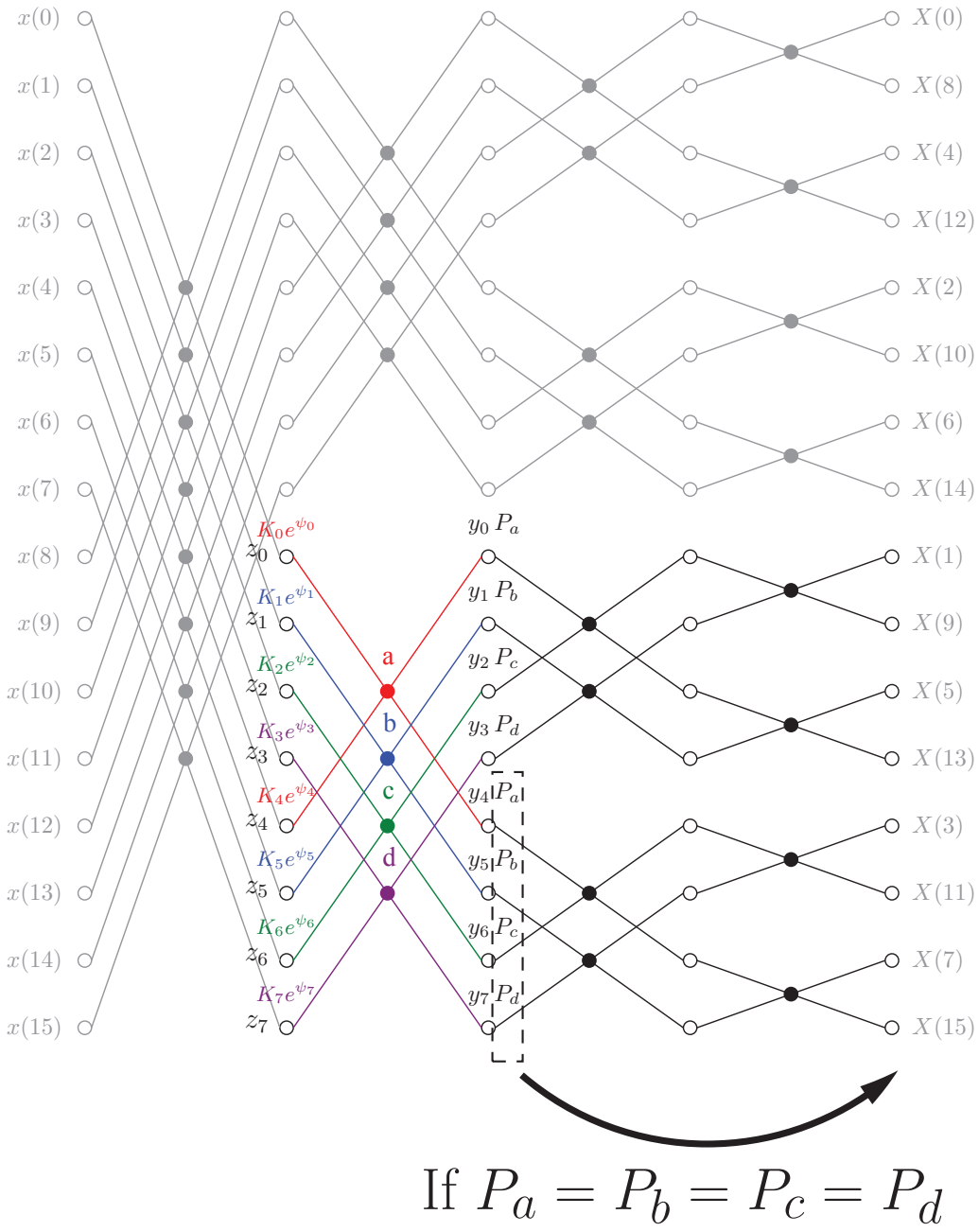


**Figure 3.14: Moving the PTFs to the last FFT stage illustration.**

# Chapter 4

# Error Analysis

On butterfly level, there are two error sources,

- angle residual error produced by non-complete angle representation;
- gain residual error produced by unequal scale factors multiplied by each butterfly operand.

Fig. 4.1 shows the CORDIC Friendly radix-r butterfly operation where $X_0$, $X_1$,..., $X_{r-1}$ are the input, $K_0e^{\psi_0}$, $K_1e^{\psi_1}$,..., $K_{r-1}e^{\psi_{r-1}}$ are the MTFs and $P$ is the PTF, and $\widehat{Y}_0$, $\widehat{Y}_1$,..., $\widehat{Y}_{r-1}$ are the desired output of the butterfly operation. Similar to the conventional radix-r butterfly, the outputs of CORDIC-Friendly radix-r butterfly are governed by the following equation,

$$\widehat{Y}_\mu P = \sum_{\nu=0}^{r-1} W_r^{\nu\mu} y_\nu K_\nu e^{j\psi_\nu} \tag{4.1}$$

The reference model is shown in Fig. 3.11 (a).

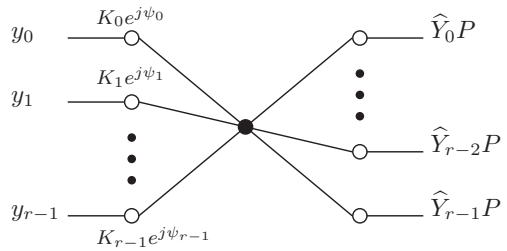


**Figure 4.1: Radix-r CORDIC-Friendly butterfly operation**

Error is defined as the difference between the proposed and reference model, as shown,

$$E_\mu = Y_\mu - \widehat{Y_\mu} \tag{4.2}$$

$$E_\mu = \overbrace{\sum_{\nu=0}^{r-1} W_r^{\nu\mu} y_\nu e^{j\nu\phi}}^{reference\,model} - \sum_{\nu=0}^{r-1} W_r^{\nu\mu} y_\nu \frac{K_\nu e^{j\psi_\nu}}{P} \tag{4.3}$$

$$E_\mu = \sum_{\nu=0}^{r-1} W_r^{\nu\mu} y_\nu (e^{j\nu\phi} - \frac{K_\nu e^{j\psi_\nu}}{P}). \tag{4.4}$$

Let $K_\nu = e^{\alpha_\nu}$ and $P = e^{\beta}$ where $\alpha \in \mathbb{R}$ and $\beta \in \mathbb{C}$, hence

$$E_\mu = \sum_{\nu=0}^{r-1} W_r^{\nu\mu} y_\nu (e^{j\nu\phi} - \frac{e^{\alpha_\nu + j\psi_\nu}}{e^{\beta}}) \tag{4.5}$$

$$E_\mu = \sum_{\nu=0}^{r-1} W_r^{\nu\mu} y_\nu e^{j\nu\phi} (1 - \frac{e^{\alpha_\nu + j\psi_\nu}}{e^{\beta + j\nu\phi}}) \tag{4.6}$$

$$E_\mu = \sum_{\nu=0}^{r-1} W_r^{\nu\mu} y_\nu e^{j\nu\phi} (1 - \frac{e^{\Omega_\nu}}{e^{\beta}}) \tag{4.7}$$

$$E_\mu = \sum_{\nu=0}^{r-1} W_r^{\nu\mu} y_\nu e^{j\nu\phi} (1 - e^{\Omega'_\nu}) \tag{4.8}$$

where $\Omega_\nu = \alpha_\nu + j(\psi_\nu - \nu\phi)$ and $\Omega'_\nu = \Omega_\nu - \beta$. Therefore the error variance is

$$\sigma^2_{E_\mu} = \sigma^2_X \sum_{\nu=0}^{r-1} |1 - e^{\Omega'_\nu}|^2. \tag{4.9}$$

Hypothetically, if $\Omega'_\nu$ was continuous, the optimal solution that minimize the the error variance should be at $\Omega'_\nu = 0$, and hence at $\Omega_0 = \Omega_1 = \cdots = \Omega_{r-1} = \beta$. However, $\Omega'_\nu$ is function in $\alpha_\nu + j\psi_\nu$ which is belonging to discreet CORDIC set. Therefore, the optimization problem of minimizing the error variance can be formulated as follows,

$$\begin{aligned} \underset{\Omega_\nu}{minimize} \quad & (\sum_{\nu=0}^{r-1} |1 - e^{\Omega_\nu - \beta}|^2) \\ subject\,to: \quad & \Omega_\nu = \alpha_\nu + j(\psi_\nu - \nu\phi) \\ & \alpha_\nu + j\psi_\nu \in CORDIC\,Set \\ & \phi \in \mathbb{R} \\ & \beta \in \mathbb{C}. \end{aligned} \tag{4.10}$$

Finding the best $\Omega_\nu$ that minimize the above expression is a cumbersome combinatorial optimization problem. On the other hand for a given $\Omega_\nu$, choosing the best $\beta$ that minimize the objective function can be done using calculus as shown in the next steps. Back to the error variance expression, it can be written as,

$$\sigma^2_{E_\mu} = \sigma^2_X \sum_{\nu=0}^{r-1} \left| 1 - \frac{e^{\Omega_\nu}}{e^\beta} \right|^2 \tag{4.11}$$

$$\sigma^2_{E_\mu} = \sigma^2_X \sum_{\nu=0}^{r-1} \left( 1 - \frac{e^{\Omega_\nu}}{e^\beta} \right) \left( 1 - \frac{(e^{\Omega_\nu})^*}{(e^\beta)^*} \right). \tag{4.12}$$

Therefore, we need to differentiate with respect to $e^\beta$ to find its value that minimize the error variance. Equation 4.11 is real-valued function of complex variables. Unfortunately, this function is not holomorphic function, since it does not satisfy Cauchy–Riemann equations. Therefore, it is not complex differentiable. One way to solve this problem is by decomposing $e^\beta$ to its real and imaginary components and differentiate with respect to them. However this process is tedious. Another way is presented in [**Stationary-points**]. It suggests to decompose the function $\sigma^2_{E_\mu}(e^\beta)$ to $\sigma^2_{E_\mu}(e^\beta, (e^\beta)^*)$ as shown in equation 4.12. Then differentiate with respect to $(e^\beta)^*$. Hence,

$$\frac{d\sigma^2_{E_\mu}}{d(e^\beta)^*} = 0 \tag{4.13}$$

$$\sum_{\nu=0}^{r-1} (e^{\Omega_\nu})^* \left( e^\beta - e^{\Omega_\nu} \right) = 0 \tag{4.14}$$

$$e^\beta = \frac{\sum_{\nu=0}^{r-1} |e^{\Omega_\nu}|^2}{\sum_{\nu=0}^{r-1} (e^{\Omega_\nu})^*}. \tag{4.15}$$

In order to simplify the combinatorial optimization problem in $\Omega_\nu$ the following lemma is proposed.

***Lemma 1:*** if we ignore the second and higer order terms, the optimization's objective function can be simplified to,

$$\underset{\Omega_\nu}{arg\,min} \left( \sum_{\nu=0}^{r-1} |1 - e^{\Omega_\nu - \beta}|^2 \right) = \underset{\Omega_\nu}{arg\,min} \left( \sum_{\nu=0}^{r-1} \left| \Omega_\nu - \overline{\Omega} \right|^2 \right) \tag{4.16}$$

where $\overline{\Omega} = \sum_{\nu=0}^{r-1} \frac{\Omega_\nu}{r}$ is the arithmetic mean of the candidate points and $\beta$ is given by $e^\beta = \frac{\sum_{\nu=0}^{r-1} |e^{\Omega_\nu}|^2}{\sum_{\nu=0}^{r-1} (e^{\Omega_\nu})^*}$.

***Proof 1:***

It is obvious that the optimum solution is when $\Omega_0 = \Omega_1 = \cdots = \Omega_{r-1} = \beta$. Thus, around the optimum solution, we can assume that $|\Omega_\nu - \beta| \ll 1$. Then using taylor expansion, $e^x = \sum_{n=0}^{\infty} \frac{x^n}{n!}$ , the previous formula can be simplified to,

$$\underset{\Omega_\nu}{arg\,min}(\sum_{\nu=0}^{r-1}|1-e^{\Omega_\nu-\beta}|^2) \quad \approx \quad \underset{\Omega_\nu}{arg\,min}(\sum_{\nu=0}^{r-1}|\Omega_\nu-\beta|^2). \tag{4.17}$$

Now, we will try to proof that $\beta \approx \sum_{m=0}^{r-1}\frac{\Omega_m}{r}$. Starting with,

$$e^{\beta} = \frac{\sum_{l=0}^{r-1}|e^{\Omega_l}|^2}{\sum_{n=0}^{r-1}(e^{\Omega_n})^*} \tag{4.18}$$

$$e^{\beta} = \sum_{l=0}^{r-1}\frac{e^{\Omega_l}}{1+\sum_{\substack{n=0\\n\neq l}}^{r-1}(e^{\Omega_n-\Omega_l})^*}, \tag{4.19}$$

we can assume that $\Omega_n - \Omega_l \ll 1$, then using Taylor expansion with second and higher order terms ignored, we get,

$$e^{\beta} \approx \sum_{l=0}^{r-1}\frac{e^{\Omega_l}}{1+\sum_{\substack{n=0\\n\neq l}}^{r-1}(1+(\Omega_n-\Omega_l)^*)} \tag{4.20}$$

$$e^{\beta} \approx \frac{1}{r}\sum_{l=0}^{r-1}\frac{e^{\Omega_l}}{1+\sum_{\substack{n=0\\n\neq l}}^{r-1}\frac{(\Omega_n-\Omega_l)^*}{r}}. \tag{4.21}$$

We can use $\frac{1}{1+x} = \sum_{n=0}^{\infty}(-1)^n x^n$ for $|x| < 1$ for more simplification as shown,

$$e^{\beta} \approx \frac{1}{r}\sum_{l=0}^{r-1}e^{\Omega_l}\left(1-\sum_{\substack{n=0\\n\neq l}}^{r-1}\frac{(\Omega_n-\Omega_l)^*}{r}\right) \tag{4.22}$$

$$e^{\beta} \approx \frac{1}{r}\sum_{l=0}^{r-1}e^{\Omega_l}\left(1+\sum_{\substack{n=0\\n\neq l}}^{r-1}\frac{(-\Omega_n+\Omega_l)^*}{r}\right). \tag{4.23}$$

Then by taking $e^{\sum_{m=0}^{r-1}\Omega_m/r}$ as a common factor we get,

$$e^{\beta} \approx \frac{1}{r}e^{\sum_{m=0}^{r-1}\Omega_m/r}\sum_{l=0}^{r-1}e^{\Omega_l-\sum\Omega_n/r}\left(1+\sum_{\substack{n=0\\n\neq l}}^{r-1}\frac{(-\Omega_n+\Omega_l)^*}{r}\right). \tag{4.24}$$

Then by using Taylor expansion again and ignoring higher order terms,

$$e^{\beta} \approx \frac{1}{r} e^{\sum \Omega_n / r} \sum_{l=0}^{r-1} \left( 1 + \Omega_l - \sum_{m=0}^{r-1} \Omega_m / r \right) \left( 1 + \sum_{\substack{n=0 \\ n \neq l}}^{r-1} \frac{(-\Omega_n + \Omega_l)^*}{r} \right) \tag{4.25}$$

$$e^{\beta} \approx \frac{1}{r} e^{\sum \Omega_n / r} \sum_{l=0}^{r-1} \left( 1 + \Omega_l - \sum_{m=0}^{r-1} \Omega_m / r + \sum_{\substack{n=0 \\ n \neq l}}^{r-1} \frac{(-\Omega_n + \Omega_l)^*}{r} \right). \tag{4.26}$$

Notice that $\sum_{l=0}^{r-1} \sum_{\substack{n=0 \\ n \neq l}}^{r-1} \frac{(-\Omega_n + \Omega_l)^*}{r} = 0$ and $\sum_{l=0}^{r-1} \Omega_l - \sum_{m=0}^{r-1} \Omega_m / r = 0$, then,

$$e^{\beta} \approx e^{\sum_{m=0}^{r-1} \Omega_m / r} \tag{4.27}$$

$$\beta \approx \sum_{m=0}^{r-1} \frac{\Omega_m}{r} = \overline{\Omega}. \tag{4.28}$$

■ (4.29)

Beside the simplification of the objective function, which results in speeding up the computer optimization program, this lemma gives some insights. As we can see from the previous formula, what matters is the variance of the target angles and not their absolute values which is the exploited degree of freedom.

Fig. 4.2 shows the SFG of 16-point conventional radix-4 FFT. After applying the proposed algorithm, Fig. 4.3 shows the SFG of 16-point CORDIC-Friendly radix-4 FFT.

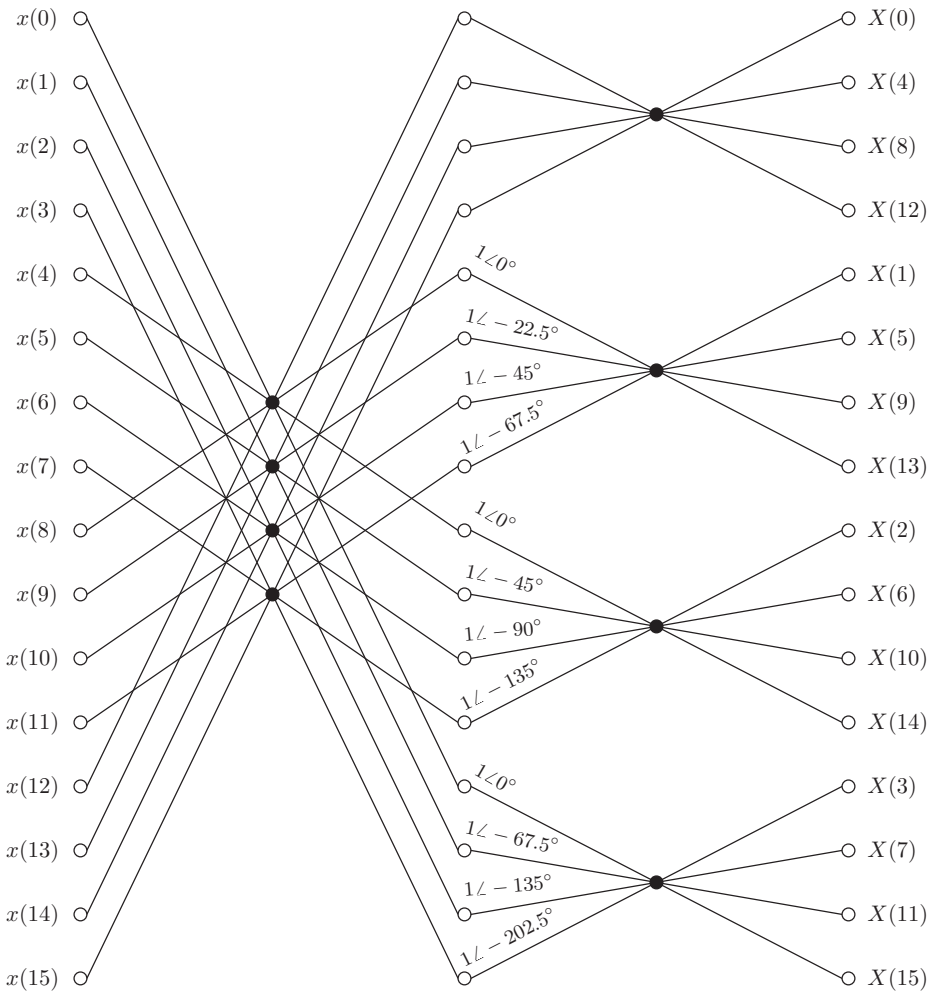


**Figure 4.2: Conventional 16-point radix-4 FFT SFG.**

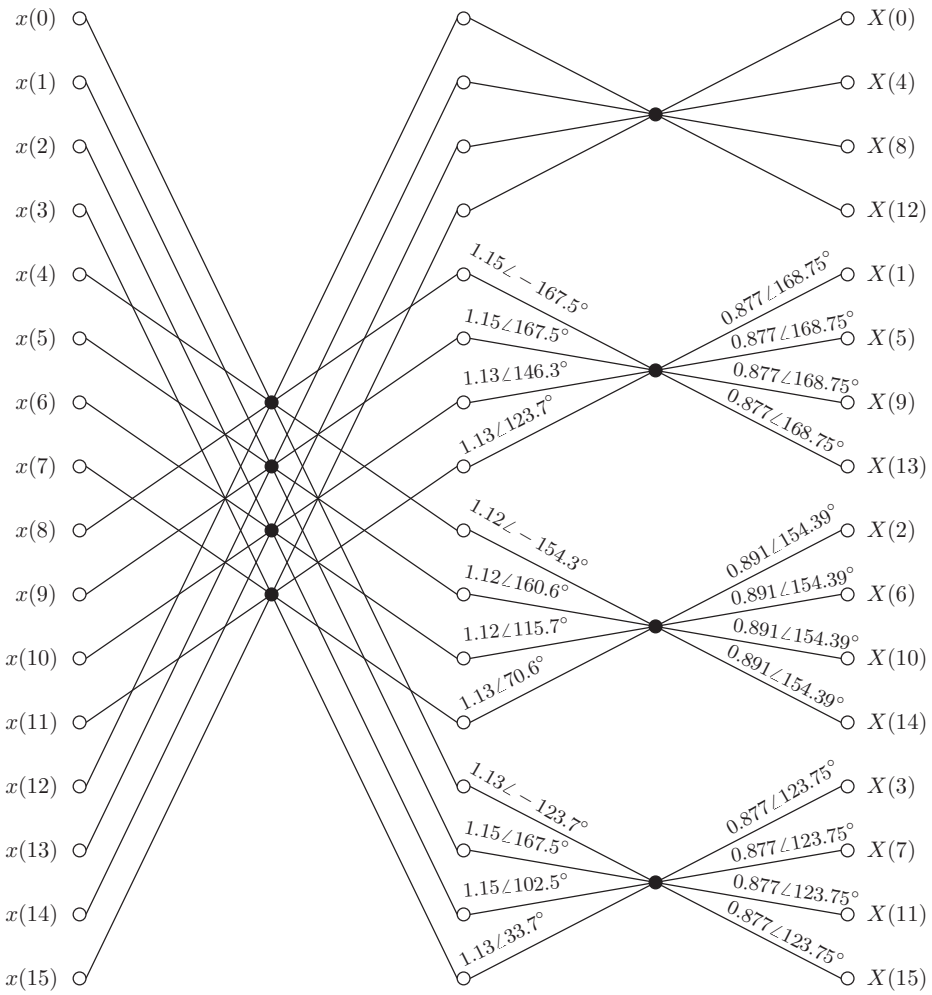


**Figure 4.3: CORDIC-Friendly 16-point radix-4 FFT SFG.**

# Chapter 5

# Optimization Methodology

## 5.1 Motivation

We ended up in the previous chapter formulating the optimal solution of the optimization problem as,

$$\begin{aligned} \underset{\Omega_\nu}{\arg\min} \quad & \sum_{\nu=0}^{r-1} \left| \Omega_\nu - \overline{\Omega} \right|^2 \\ \text{subject to} \quad & \Omega_\nu = \alpha_\nu + j(\psi_\nu - \nu\phi) \\ & \alpha_\nu + j\psi_\nu \in \text{CORDIC Set} \\ & \phi \in \mathbb{R} \\ & \overline{\Omega} = \sum_{\nu=0}^{r-1} \frac{\Omega_\nu}{r}. \end{aligned} \tag{5.1}$$

A naive solution to start with is brute-force search or exhaustive search. Brute-force search will examine all possible candidates for the solution and check which of them is satisfying the optimization statement. In our case, the number of candidates to be examined per butterfly is $S^r_{CORDIC}$, where $S_{CORDIC}$ is the size of the CORDIC set. Hence, for large CORDIC sets, brute-force is unpractical. Table 5.1 shows the combinatorial explosion in the number of candidates in case of large CORDIC sets and high radices. Therefore, it was necessary to find a smarter way to search.

**Table 5.1: Number of candidates examined per butterfly using brute-force search.**

| Radix | MVR | | | | MSR (2,1) | |
|---|---|---|---|---|---|---|
| | 1-iter | 2-iter | 3-iter | 4-iter | 1-iter | 2-iter |
| Two | $9.6\times10^2$ | $2.5\times10^5$ | $3.7\times10^7$ | $3.8\times10^9$ | $6.9\times10^6$ | $6.9\times10^{12}$ |
| Three | $3\times10^4$ | $1.3\times10^8$ | $2.3\times10^{11}$ | $2.3\times10^{14}$ | $1.8\times10^{10}$ | $1.8\times10^{19}$ |
| Four | $9.2\times10^5$ | $6.4\times10^{10}$ | $1.4\times10^5$ | $1.4\times10^{19}$ | $4.8\times10^{13}$ | $4.7\times10^{25}$ |

## 5.2 A framework to visualize the candidate solutions

A CORDIC set can be represented in two ways, either

1. Re + $j$Im, which equals to CORDIC gain $\times e^{j\text{CORDIC angle}} = e^{\alpha + j\psi}$, or
2. $\alpha + j\psi$, which named as the complex CORDIC angle.

The latter representation is used explicitly in the optimization problem, and hence, we will commonly refer to it. Fig. 5.1 (a) illustrates Re + $j$Im representation of MSR(2,1) CORDIC set. Fig. 5.1 (b) illustrates $\alpha + j\psi$ representations of the sector bounded with the green dashed ±45°lines. As a result, the transformed points of the sector are limited in the $\psi$ part to $\pm\frac{\pi}{4}$. The points included in the sector can construct the whole CORDIC set by repeating each point in $\alpha + j\psi$ representation four times and adding $\{0, j\frac{\pi}{2}, -j\frac{\pi}{2}, j\pi,\}$ on each of which.

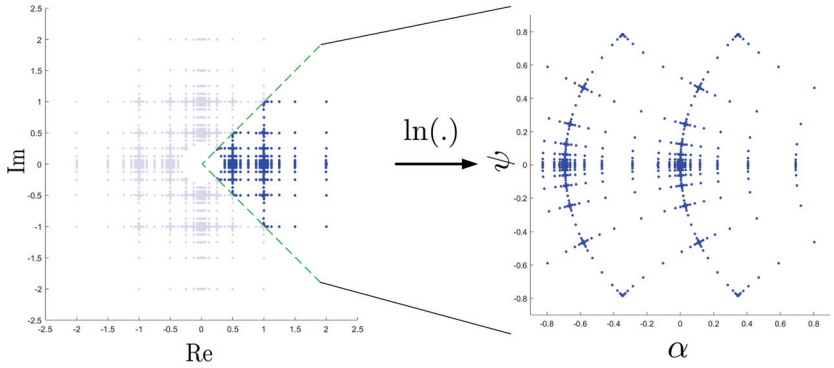


**Figure 5.1: CORDIC set representations. (a) Re + $j$Im and, (b) $\alpha + j\psi$**

In the optimization problem formula 5.1, the variable of interest is the modified twiddle factor exponent $\Omega_\nu = \alpha_\nu + j(\psi_\nu - \nu\phi) = \alpha_\nu + j\psi'_\nu$. For each butterfly, $\phi$ is known. Hence, for each butterfly arm, a shifted version of Fig. 5.1 (b) can be evaluated for each $\Omega_\nu$. For example, given radix-3 butterfly with $\phi = -\frac{\pi}{8}$ and MSR(2,1) CORDIC, the points of $\Omega_0$ are the same as Fig. 5.1 (b) and the points of $\Omega_1$ and$\Omega_2$ are equal to $\Omega_0$ with shift upward by $\frac{\pi}{8}$ and $\frac{\pi}{4}$, respectively. Fig. 5.2 (a) illustrates the previous example where $\Omega_0$, $\Omega_1$, and $\Omega_2$ points are presented in blue, black, and red, respectively.

Broadly speaking, the optimal solution that satisfies equation 5.1 consists of r neighbor points, where r is the radix; each of which belongs to different $\Omega_\nu$ . The proximity metric for the candidate points is their variance, i.e. $\sum_{\nu=0}^{r-1} \left|\Omega_\nu - \overline{\Omega}\right|^2$.

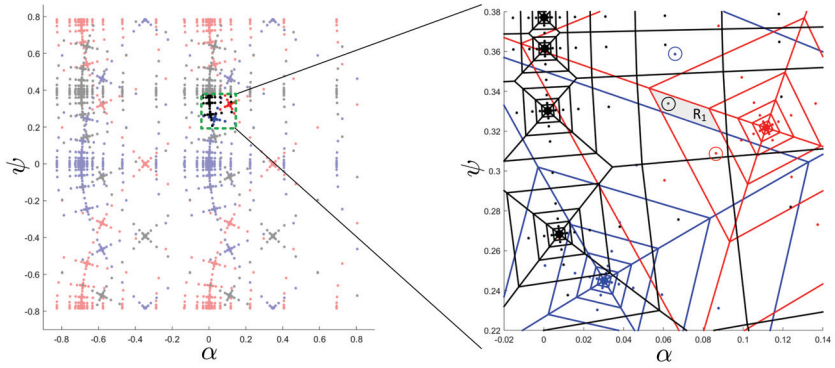


**Figure 5.2: (a) $\Omega_\nu$ of radix-3 butterfly with $\phi = -\frac{\pi}{8}$ and MSR(2,1) CORDIC, where $\Omega_0$, $\Omega_1$, and $\Omega_2$ points are presented in blue, black, and red, respectively. (b) A zoomed version of (a) with voronoi space partitioning.**

Originally, $\psi'$ has range $[-\pi, \pi)$, however as mentioned before, $\psi'$ is periodic with period equals to $\frac{\pi}{2}$. Therefore, the r neighbor points group that minimize the proximity equation is not unique, as a matter of fact, there are four copies of it. The relation between these for groups are known; each group is apart from its neighbor group by $j\frac{\pi}{2}$. Therefore, to minimize the search space to quarter, we search for the solution in $\psi' \in [-\frac{\pi}{4}, \frac{\pi}{4})$ only. This can be visualized as the domain of $\alpha + j\psi'$ is erected on the side surface of a cylinder, as shown in Fig. 5.3, where the $\alpha$ axis is parallel to the cylinder centerline and $\psi'$ wraps along the cylinder centerline.

To remove the discontinuity of at $\psi' = \pm\frac{\pi}{4}$, we search for the solution in $\psi' \in [-\frac{\pi}{4}, \frac{\pi}{4} + \Delta)$, where $\Delta$ is chosen based on how dense the CORDIC set is. The reason for this modification is to ensure that the points near one edge will see the points on the other edge as neighbors, which perfectly map the visualization in Fig. 5.3.

Next we will explore two search techniques to locate the nearest neighbor points.

## 5.3 k-Nearest Neighbors (k-NN) search technique

This algorithm, described in [28], was presented around 50 years ago to speed-up documents searching. Given a set X of $N_X$ points represented in M dimension and a distance function, k-NN search technique finds the k closest points in X to each point in the query set Y of size $N_Y$. If k = 1, then the query point or the set of points Y is simply assigned to the single nearest neighbor. Fig. 5.4 demonstrates the k-NN search problem with k=1 in two dimension space, i.e. M=2, where the red points represents the query points, blue ones represents the data points and the circles assign the nearest neighbor to query point.

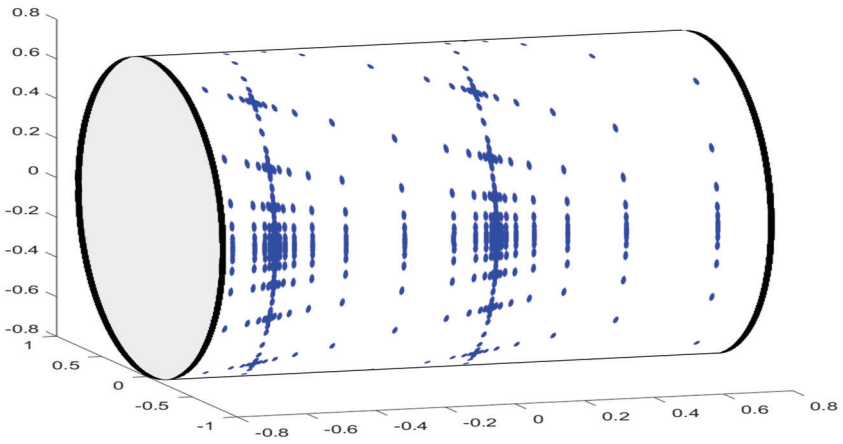


**Figure 5.3:** $\alpha + j\psi'$ **domain is circular on** $\psi'$ **axis, and hence, it can visualized as working on the side surface of a cylinder.**

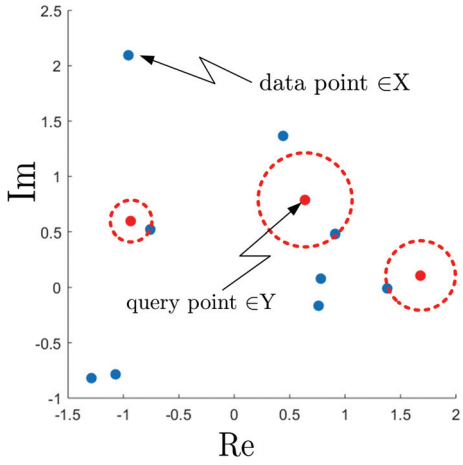


**Figure 5.4: k-NN search problem with k=1 in two dimension space, i.e. M=2, where the red points represents the query points, blue ones represents the data points and the circles assign the nearest neighbor to query point.**

k-NN search consists of two main tasks,

1. building k-d tree, which is a way to organize the points of set X in efficient way, i.e. balanced binary tree, to simplify the search procedure; the computation required to build the k-d tree is proportional to $MN_X \log(N_X)$, i.e. $O(MN_X \log(N_X))$,
2. assigning each point in the query set Y to the nearest k neighbors in set X; and thanks to the k-d tree, the complexity of this search procedure is close to $O(kN_Y \log(N_X))$.

Mapping the optimization problem of radix-2 to 1-NN problem is possible with help of the next lemma.

**Lemma 2:**

Given the optimization formula 5.1 and the radix r equals 2,

$$\arg\min_{\Omega_\nu} \sum_{\nu=0}^{1} \left|\Omega_\nu - \overline{\Omega}\right|^2 = \arg\min_{\Omega_\nu} |\Omega_0 - \Omega_1|^2 . \tag{5.2}$$

**Proof:**

Starting from the left hand side of formula 5.2,

$$\begin{aligned} \sum_{\nu=0}^{1} \left|\Omega_\nu - \overline{\Omega}\right|^2 &= \left|\Omega_0 - (\frac{\Omega_0 + \Omega_1}{2})\right|^2 + \left|\Omega_1 - (\frac{\Omega_0 + \Omega_1}{2})\right|^2 \\ &= \left|\frac{\Omega_0 - \Omega_1}{2}\right|^2 + \left|\frac{\Omega_1 - \Omega_0}{2}\right|^2 \\ &= \frac{1}{2} |\Omega_0 - \Omega_1|^2 . \end{aligned} \tag{5.3}$$

Since the positive scalar multiplication has no effect on the right hand side of formula 5.2, lemma 2 is true. ■

Using lemma 2, the optimization objective sages to finding two points $\widehat{\omega}_0 \in \Omega_0$ and $\widehat{\omega}_1 \in \Omega_1$ whose distance between them is the closest. Pseudo code for the searching algorithm is shown in Algorithm 5.1.

**Algorithm 5.1:** Optimization procedure for radix-r butterfly using Voronoi-based space partitioning.

**Input:** CORDIC set $\Omega = \{\omega_0, \omega_1, ..., \omega_n\}$ of complex numbers
$\phi$ in radian

**Output:** $\widehat{\omega}_\nu$, where $\nu \in \{0, 1, ..., r-1\}$

1 $\Omega_0 \leftarrow \Omega$
2 $\Omega_1 \leftarrow \Omega - j\phi$
3 $\left[\widehat{\Omega}_0, distance\,metric\right] \leftarrow$k-NN search($X \leftarrow \Omega_0$,$Y \leftarrow \Omega_1$,$k \leftarrow 1$)
4 $index \leftarrow$arg min($distance\,metric$)
5 $\widehat{\omega}_0 \leftarrow \widehat{\Omega}_0(index), \widehat{\omega}_1 \leftarrow \Omega_1(index)$

Even though k-NN search find the optimal solution for formula 5.1, there is no obvious way, to our knowledge, to use the same method directly to get the optimal solution for general radix-r butterfly. Hence, the following section describe another approach using Voronoi-based space partitioning.

## 5.4 Voronoi-based space partitioning technique

Given a set X of $N_X$ points, Voronoi diagram partitions a space into regions based on distance to the points of X, where a point $x_i \in$X, usually called seed, is corresponding to a region $R_i$. Voronoi partitioning ensure that all the points enclosed by $R_i$ are closer to $x_i$ than to any other seeds $x_{j\neq i} \in X$. Fig. 5.5 demonstrates Voronoi-based partitioning for two-dimension space, where (a) shows unpartitioned space with the seeds, and (b) shows the space after partitioning.

Mapping the optimization problem to Voronoi-based space partitioning problem is possible with help of the next lemmas.

**Lemma 3:**

Given the complex point $\xi$,

$$\min_{\Omega_\nu} \sum_{\nu=0}^{r-1} |\Omega_\nu - \xi|^2 \quad = \quad \sum_{\nu=0}^{r-1} \min_{\Omega_\nu} |\Omega_\nu - \xi|^2 . \tag{5.4}$$

**Proof:**

Since $\Omega_\nu$ sets are independent of each other, and $|\Omega_\nu - \xi|^2$ is positive real-valued function, lemma 3 is true. ■

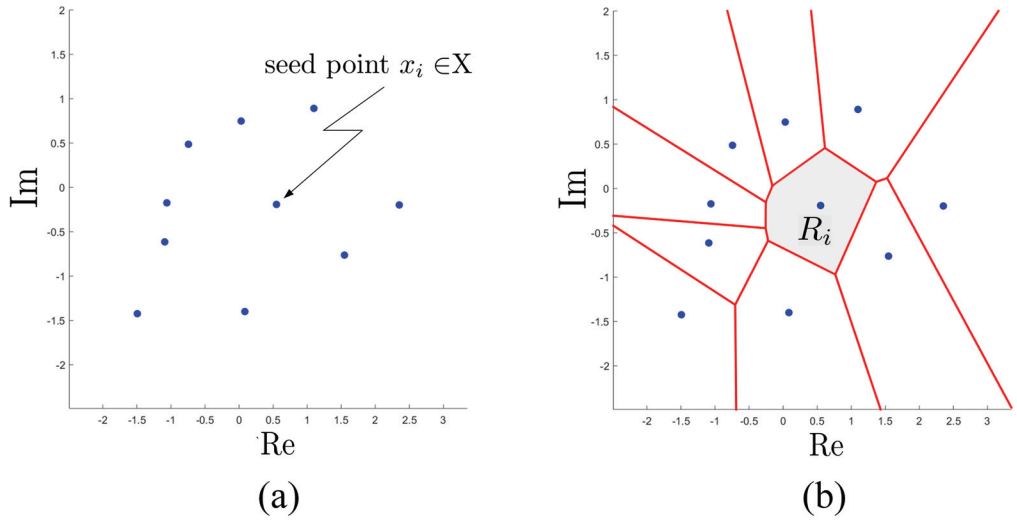


**Figure 5.5: Voronoi-based partitioning for two-dimension space. (a) unpartitioned space. (b) Voronoi-partitioned space.**

**Lemma 4:**

For a given solution candidate $\omega_\nu \in \Omega_\nu$, if

$$\arg\min_{\xi} \sum_{\nu=0}^{r-1} |\omega_\nu - \xi|^2 \quad = \quad \sum_{\nu=0}^{r-1} \frac{\omega_\nu}{r}. \tag{5.5}$$

**Proof:**

given that,

$$f \quad = \quad \sum_{\nu=0}^{r-1} (\omega_\nu - \xi)(\omega_\nu - \xi)^*. \tag{5.6}$$

In order to minimize a real-valued function of complex variables, [**Stationary-points**] suggests to decompose the function $f(|\xi|)$ to $f(\xi, \xi^*)$ , then differentiate with respect to $\xi^*$as follows,

$$\begin{aligned}
\frac{df}{d\xi^*} &= \sum_{\nu=0}^{r-1}(\xi-\omega_\nu)=0 \\
\sum_{\nu=0}^{r-1}\xi &= \sum_{\nu=0}^{r-1}\omega_\nu \\
\xi &= \sum_{\nu=0}^{r-1}\frac{\omega_\nu}{r}. \blacksquare
\end{aligned} \tag{5.7}$$

**Lemma 5:**

For a given solution candidate $\omega_\nu \in \Omega_\nu$, if

$$\omega_\nu \neq \arg\min_{\Omega_\nu} |\Omega_\nu - \overline{\omega}|^2 \tag{5.8}$$

where $\overline{\omega} = \sum_{\nu=0}^{r-1}\frac{\omega_\nu}{r}$, then the solution candidate $\omega_\nu$ is not the optimal solution for formula 5.1.

**Proof:**

We will prove this lemma by contradiction. Given that $\omega_\nu, \omega'_\nu \in \Omega_\nu$, $\omega'_\nu = \arg\min_{\Omega_\nu} |\Omega_\nu - \overline{\omega}|^2$ and $\omega_\nu \neq \omega'_\nu$, assume that $\omega_\nu$ is the optimal solution for formula 5.1. Therefore,

$$\sum_{\nu=0}^{r-1}\left|\omega'_\nu - \overline{\omega}\right|^2 < \sum_{\nu=0}^{r-1}|\omega_\nu - \overline{\omega}|^2. \tag{5.9}$$

According to lemma 4,

$$\sum_{\nu=0}^{r-1}\left|\omega'_\nu - \overline{\omega'}\right|^2 < \sum_{\nu=0}^{r-1}\left|\omega'_\nu - \overline{\omega}\right|^2 \tag{5.10}$$

and hence,

$$\sum_{\nu=0}^{r-1}\left|\omega'_\nu - \overline{\omega'}\right|^2 < \sum_{\nu=0}^{r-1}|\omega_\nu - \overline{\omega}|^2. \tag{5.11}$$

Therefore, the solution candidate $\omega_\nu$ is not the optimal solution that minimize formula 5.2. ■

Hence, if the space that has the points of $\Omega_\nu$, i.e. Fig. 5.2 (a), have been partitioned according to Voronoi, i.e. in Fig. 5.2 (b), then collect the seeds belongs to each region, and finally choose group of seed that satisfy 5.1, the optimal solution is guaranteed. Pseudo

code for the searching algorithm is shown in Algorithm 5.2, where Bentley-Ottmann algorithm [29] is a famous sweep line algorithm for listing all crossings in a set of line segments and hence can get all the regions of the Voronoi diagrams.

**Algorithm 5.2:** Optimization procedure for radix-r butterfly using Voronoi-based space partitioning.

**Input:** CORDIC set $\Omega = \{\omega_0, \omega_1, ..., \omega_n\}$ of complex numbers
$\phi$ in radian
the radix $r$

**Output:** $\widehat{\omega_\nu}$, where $\nu \in \{0, 1, ..., r-1\}$

**for** $i = 0$ *to* $r-1$ **do**
  $\Omega_i \leftarrow \Omega - j\phi \times i$
  $Voronoi\,diagram\,edges_i \leftarrow get\,Voronoi\,diagram(X \leftarrow \Omega_i)$
**end**
$\widehat{\Omega_\nu} \leftarrow$Bentley-Ottmann algorithm($X_\nu \leftarrow Voronoi\,diagram\,edges_\nu$), where $\nu \in 0, 1, ..., r-1$
$distance\,metric \leftarrow \sum_{\nu=0}^{r-1} \left|\widehat{\Omega_\nu} - \overline{\widehat{\Omega}}\right|^2$, $where\,\overline{\widehat{\Omega}} = \sum_{\nu=0}^{r-1} \frac{\widehat{\Omega_\nu}}{r}$
$index \leftarrow$arg min($distance\,metric$)
$\widehat{\omega_\nu} \leftarrow \widehat{\Omega_\nu}(index)$, where $\nu \in 0, 1, ..., r-1$

For the time constrain of my master thesis, the sub-optimal algorithm 5.3 has been used.

**Algorithm 5.3:** suboptimal Optimization procedure for radix-r butterfly using Voronoi-based space partitioning.

**Input:** CORDIC set $\Omega = \{\omega_0, \omega_1, ..., \omega_n\}$ of complex numbers
$\phi$ in radian
the radix $r$

**Output:** $\widehat{\omega}_\nu$, where $\nu \in \{0, 1, ..., r-1\}$

**for** $i = 0$ *to* $r-1$ **do**
  $\Omega_i \leftarrow \Omega - j\phi \times i$
**end**
$\Omega_{tot} \leftarrow concatenate(\Omega_\nu)$, where $\nu \in 0, 1, ..., r-1$
**for** $i = 0$ *to* $r-1$ **do**
  $\widehat{\Omega}_i \leftarrow$k-NN search($X \leftarrow \Omega_i, Y \leftarrow \Omega_{tot}, k \leftarrow 1$)
**end**
$distance\,metric \leftarrow \sum_{\nu=0}^{r-1} \left| \widehat{\Omega}_\nu - \overline{\widehat{\Omega}} \right|^2$, $where\, \overline{\widehat{\Omega}} = \sum_{\nu=0}^{r-1} \frac{\widehat{\Omega}_\nu}{r}$
$index \leftarrow$arg min($distance\,metric$)
$\widehat{\omega}_\nu \leftarrow \widehat{\Omega}_\nu(index)$, where $\nu \in 0, 1, ..., r-1$

# Chapter 6

# Evaluation

We have simulated the proposed algorithm on radix-r FFT with different CORDIC types, namely MVR-CORDIC [8], EEAS-CORDIC [9] and MSR-CORDIC [10], and complex multiplier. After that, SQNR comparison has been drawn for conventional radix-r FFT verses the hardware-friendly FFT algorithm.

As a case study, we implement the R2SDF architecture, and compare the area of the hardware between the conventional and hardware-friendly FFT.

## 6.1 SQNR simulation measurements

The simulation environment is described as follows:

- 90, -90, and 180 degree rotations do not need any CORDIC operation.
- Equalization of scaling factor is assumed to be done without error. This equalization is either done in the stage after FFT, such as channel equalization in communication system, or by a vector of MSR-CORDICs with high accuracy.
- The simulation results are calculated for both floating and fixed point; In the fixed point simulation, the data is dynamically scaled each FFT stage.
- The shifter employed in the CORDIC algorithm can shift up to 15 bits.
- The angles of rotations are chosen such that the elementary CORDIC gain is limited between $\frac{1}{3}$ and 3. This will preserve accuracy by limiting round off error when fixed point simulation is performed.
- To prevent the propagating CORDIC gain of getting bigger or smaller each stage, right and left shifts are allowed each stage; therefore the CORDIC gain at the final stage is limited between $\frac{2}{3}$ to $\frac{4}{3}$.

- SQNR is calculated using the following equation,

$$\text{SQNR} = 10\log_{10}\left(\frac{\sum \text{Signal power}}{\sum \text{Noise power}}\right). \quad (6.1)$$

- The elements of the FFT input test vectors are iid standard complex Gaussian random variables.

### 6.1.1 CORDIC case comparison

Tables 6.1, 6.2, 6.3, and 6.4 shows the SQNR performance for radix-2, radix-3, radix-4, and radix-8, respectively, hardware-friendly FFT using different lengths and various CORDIC types. Tables 6.5, 6.6, 6.7, and 6.8 shows the SQNR performance for radix-2, radix-3, radix-4, and radix-8, respectively, conventional FFT using different lengths and MSR(2,1). From these tables, we can draw the following conclusions,

- hardware-friendly FFT can use both equal-gain CORDIC, such as MSR and Volder, and unequal-gain CORDIC types, such as MVR and EEAS, whilst the conventional FFT is stuck to the equal CORDIC gain which limit the choices of the designer;
- Using the same number of iteration, hardware-friendly FFT can achieve superior SQNR comparing with the conventional FFT, and hence using the less hardware, hardware-friendly FFT can achieve the same SQNR as the conventional FFT.

**Table 6.1: Hardware-Friendly radix-2 FFT SQNR performance using MVR, EEAS(2,0), and MSR(2,1) in dB.**

| N | MVR | | | | EEAS(2,1) | | MSR(2,1) | |
|---|---|---|---|---|---|---|---|---|
| | 1 iter. | 2 iter. | 3 iter. | 4 iter. | 1 iter. | 2 iter. | 1 iter. | 2 iter. |
| 1024 | 22.8 | 46.4 | 71.3 | 93.7 | 36.9 | 101.4 | 41.9 | 114.4 |
| 512 | 23.2 | 46.8 | 71.7 | 94.1 | 37.1 | 101.7 | 42 | 114.5 |
| 256 | 23.6 | 47.2 | 72.2 | 94.5 | 37.3 | 102 | 42.1 | 114.6 |
| 128 | 24 | 47.5 | 72.8 | 95 | 37.6 | 102.3 | 42.2 | 114.7 |
| 64 | 24.5 | 47.8 | 73.7 | 95.7 | 37.8 | 102.9 | 42.4 | 114.9 |
| 32 | 25 | 48.3 | 74.3 | 96.4 | 38.2 | 103.2 | 42.8 | 115.1 |
| 16 | 26.1 | 49.1 | 75.4 | 97.2 | 39 | 104.1 | 43.5 | 115.8 |
| 8 | 28 | 51 | 77.2 | 99.6 | 40.8 | 106.4 | 45.3 | 117.7 |

**Table 6.2: Hardware-Friendly radix-3 FFT SQNR performance using MVR, EEAS(2,0), and MSR(2,1) in dB.**

| N | MVR | | | | EEAS(2,1) | | MSR(2,1) | |
|---|---|---|---|---|---|---|---|---|
| | 1 iter. | 2 iter. | 3 iter. | 4 iter. | 1 iter. | 2 iter. | 1 iter. | 2 iter. |
| 2187 | 20.3 | 32.3 | 47.4 | 66.7 | 22.8 | 67.8 | 37.4 | 92 |
| 729 | 21.2 | 33.3 | 48.1 | 67.6 | 23.8 | 69 | 38.2 | 93.2 |
| 243 | 22.4 | 34.6 | 48.8 | 68.8 | 25.1 | 70.8 | 39.1 | 94.7 |
| 81 | 24 | 36.4 | 49.8 | 70.6 | 26.8 | 73.9 | 40.1 | 97 |
| 27 | 26.5 | 39.1 | 51.4 | 72.9 | 29.7 | 80.1 | 42 | 99.6 |

**Table 6.3: Hardware-Friendly radix-4 FFT SQNR performance using MVR, EEAS(2,0), and MSR(2,1) in dB.**

| N | MVR | | | | EEAS(2,1) | | MSR(2,1) | |
|---|---|---|---|---|---|---|---|---|
| | 1 iter. | 2 iter. | 3 iter. | 4 iter. | 1 iter. | 2 iter. | 1 iter. | 2 iter. |
| 4096 | 17 | 29.4 | 42 | 51.2 | 19.6 | 54.6 | 26.8 | 76.4 |
| 1024 | 17.8 | 30.4 | 42.8 | 51.5 | 20.5 | 55.1 | 27.3 | 77.2 |
| 256 | 18.8 | 31.8 | 43.8 | 51.9 | 21.5 | 55.6 | 27.8 | 78.1 |
| 64 | 20 | 33.5 | 44.9 | 52.3 | 23 | 56.6 | 28.4 | 79.5 |
| 16 | 21.9 | 37.2 | 46.8 | 53.5 | 25.2 | 58.4 | 29.7 | 81.9 |

**Table 6.4: Hardware-Friendly radix-8 FFT SQNR performance using MVR, EEAS(2,0), and MSR(2,1) in dB.**

| N | MVR | | | | EEAS(2,1) | | MSR(2,1) | |
|---|---|---|---|---|---|---|---|---|
| | 1 iter. | 2 iter. | 3 iter. | 4 iter. | 1 iter. | 2 iter. | 1 iter. | 2 iter. |
| 512 | 17.4 | 28 | 36.9 | 48.3 | 20.5 | 46.7 | 24.9 | - |
| 64 | 20.2 | 30.9 | 39.5 | 50.2 | 23.2 | 48.7 | 26.9 | 68.2 |

**Table 6.5: Conventional radix-2 FFT SQNR performance using MSR(2,1) in dB.**

| N | MSR(2,1) | | | |
|---|---|---|---|---|
| | 1 iter. | 2 iter. | 3 iter. | 4 iter. |
| 1024 | 14.8 | 39.6 | 71 | 108 |
| 512 | 15.1 | 39.8 | 71.6 | 108.8 |
| 256 | 15.5 | 40.1 | 72 | 109.9 |
| 128 | 15.9 | 40.4 | 72.6 | 111.5 |
| 64 | 16.4 | 40.7 | 73.5 | 113 |
| 32 | 17 | 41.2 | 75.2 | 113.5 |
| 16 | 17.8 | 42.4 | 79.6 | 115 |
| 8 | 19.6 | 44.1 | 81.4 | 116.9 |

**Table 6.6: Conventional radix-3 FFT SQNR performance using MSR(2,1) in dB.**

| N | MSR(2,1) | | |
|---|---|---|---|
| | 1 iter. | 2 iter. | 3 iter. |
| 2187 | 18.3 | 46.2 | 75.6 |
| 729 | 19.2 | 46.9 | 76.4 |
| 243 | 20.3 | 47.7 | 77.6 |
| 81 | 21.7 | 49.1 | 79.3 |
| 27 | 24.4 | 51.2 | 81.8 |
| 9 | 27.8 | 60.5 | 85.9 |

**Table 6.7: Conventional radix-4 FFT SQNR performance using MSR(2,1) in dB.**

| N | MSR(2,1) | | |
|---|---|---|---|
| | 1 iter. | 2 iter. | 3 iter. |
| 4096 | 16.5 | 42.2 | 77.9 |
| 1024 | 17.1 | 42.5 | 79.4 |
| 256 | 17.7 | 42.9 | 81.8 |
| 64 | 18.5 | 43.5 | 90.5 |
| 16 | 20 | 45.3 | 96.4 |

**Table 6.8: Conventional radix-8 FFT SQNR performance using MSR(2,1) in dB.**

| N | MSR(2,1) | | |
|---|---|---|---|
| | 1 iter. | 2 iter. | 3 iter. |
| 512 | 21.1 | 46.9 | 81.3 |
| 64 | 24 | 48.4 | 91.6 |

Given 1024 point FFT size, Fig. 6.1 and 6.2 show the fixed point simulation for MVR and EEAS(2,0), and MSR(2,1), respectively. At large fixed size, the quantization noise coming from fixed point truncation is negligible beside the the quantization noise coming from the incomplete representation of the twiddle factor in CORDIC.

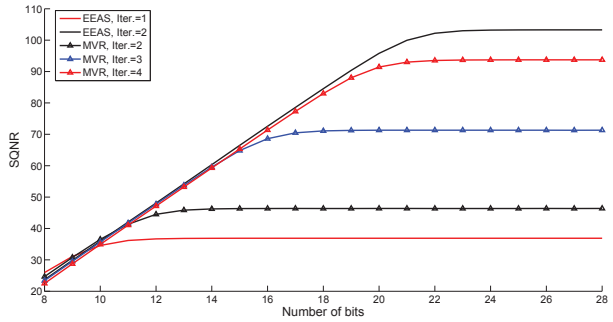


**Figure 6.1: MVR and EEAS fixed point simulation**

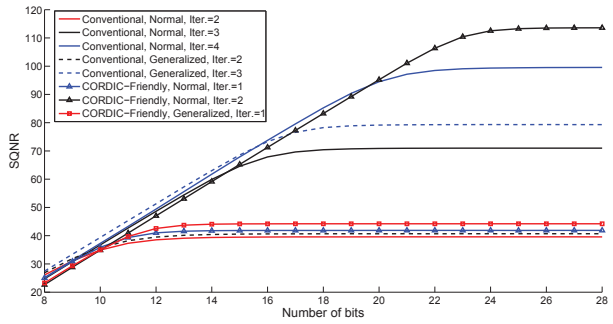


**Figure 6.2: MSR fixed point simulation**

### 6.1.2 Complex multiplier case comparison

Tables 6.9, 6.10, 6.11, and 6.12 shows the SQNR performance for radix-2, radix-3, radix-4, and radix-8, respectively, hardware-friendly FFT using different lengths and complex multiplier with different sizes for the twiddle factor part, i.e. $cos(\phi) + jsin(\phi)$. Tables 6.13, 6.14, 6.15, and 6.16 shows the SQNR performance for radix-2, radix-3, radix-4, and radix-8, respectively, conventional FFT using different lengths and complex multiplier with different sizes for the twiddle factor part. From these tables, broadly Hardware-friendly achieve the same SQNR with less multiplier sizes for radix-2 by 48%, radix-3 by 33%, radix-4 by 25%, and radix-8: 15%.

**Table 6.9: Hardware-Friendly radix-2 FFT SQNR performance using complex multipliers in dB.**

| N | Complex Multipliers | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | 10-bit | 9-bit | 8-bit | 7-bit | 6-bit | 5-bit | 4-bit | 3-bit |
| 1024 | 94.2 | 86.6 | 71.7 | 58.5 | 52.2 | 42.7 | 31.9 | 22.4 |
| 512 | 94.9 | 91 | 72.5 | 58.9 | 53.5 | 44.2 | 32.9 | 23.2 |
| 256 | 96.6 | 92.4 | 73 | 59.3 | 55 | 46.3 | 34.1 | 24.3 |
| 128 | 107 | 95.4 | 73.8 | 59.5 | 56.3 | 49.1 | 35.9 | 25.7 |
| 64 | 108.7 | 96.3 | 75.6 | 59.9 | 58.3 | 51.8 | 38.2 | 27.5 |
| 32 | 112 | 98.2 | 89.9 | 60.7 | 59.3 | 52.7 | 43 | 29.4 |
| 16 | 113.4 | 108.1 | 92.2 | 62.5 | 61.9 | 54.2 | 47.1 | 31.9 |
| 8 | 118 | 110.5 | 94.8 | 79.6 | 71.9 | 56.6 | 49 | 33.7 |

**Table 6.10: Hardware-Friendly radix-3 FFT SQNR performance using complex multipliers in dB.**

| N | Complex Multipliers | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | 10-bit | 9-bit | 8-bit | 7-bit | 6-bit | 5-bit | 4-bit | 3-bit |
| 2187 | 77.6 | 67.9 | 58.9 | 50 | 41.9 | 33.5 | 25.3 | 15.8 |
| 729 | 78.9 | 69.2 | 59.9 | 50.9 | 42.8 | 34.5 | 26.3 | 16.4 |
| 243 | 80.6 | 71 | 60.8 | 52.1 | 43.8 | 35.7 | 27.6 | 17.1 |
| 81 | 82.7 | 72.8 | 62.1 | 53.6 | 45.5 | 37.4 | 29.4 | 18 |
| 27 | 85.3 | 75.3 | 64.2 | 55.1 | 50.5 | 39.2 | 32.1 | 19.2 |

**Table 6.11: Hardware-Friendly radix-4 FFT SQNR performance using complex multiplier in dB.**

| N | Complex Multipliers | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | 10-bit | 9-bit | 8-bit | 7-bit | 6-bit | 5-bit | 4-bit | 3-bit |
| 1024 | 72.6 | 64.2 | 56.5 | 47.7 | 38.2 | 33.7 | 24.6 | 17.9 |
| 256 | 74.8 | 65.8 | 58.5 | 49.1 | 39.3 | 36 | 25.9 | 19.3 |
| 64 | 77.9 | 68.8 | 62.1 | 51.2 | 40.8 | 40.1 | 27.7 | 21.5 |
| 16 | 82.2 | 72.6 | 66.6 | 60.3 | 43.7 | 43.6 | 31 | 25.9 |

**Table 6.12: Hardware-Friendly radix-8 FFT SQNR performance using complex multiplier in dB.**

| N | Complex Multipliers | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | 10-bit | 9-bit | 8-bit | 7-bit | 6-bit | 5-bit | 4-bit | 3-bit |
| 512 | 65.1 | 58.8 | 51.4 | 45.4 | 35.6 | 31.3 | 23.1 | 16.7 |
| 64 | 68.2 | 62.6 | 54.4 | 49.5 | 38 | 35.3 | 26.2 | 19.3 |

**Table 6.13: Conventional radix-2 FFT SQNR performance using complex multiplier in dB.**

| N | Complex Multipliers | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| | 20-bit | 18-bit | 16-bit | 14-bit | 12-bit | 10-bit | 8-bit | 6-bit | 4-bit |
| 1024 | 100.4 | 87.9 | 75.8 | 64 | 51.8 | 39.8 | 28.1 | 16.7 | 6 |
| 512 | 101.3 | 88.8 | 76.6 | 64.9 | 52.7 | 40.6 | 29 | 17.6 | 6.7 |
| 256 | 102.3 | 89.7 | 77.6 | 65.9 | 53.7 | 41.6 | 30 | 18.6 | 7.4 |
| 128 | 103.5 | 90.9 | 78.6 | 67 | 54.8 | 42.8 | 31 | 19.7 | 8.3 |
| 64 | 104.8 | 92.2 | 79.8 | 68.3 | 56 | 44 | 32.4 | 20.9 | 9.4 |
| 32 | 106.3 | 93.9 | 81.3 | 69.8 | 57.5 | 45.5 | 34 | 22.4 | 10.7 |
| 16 | 108.1 | 96 | 83.2 | 71.8 | 59.4 | 47.6 | 35.9 | 24.1 | 12.5 |
| 8 | 110.7 | 98.7 | 86.1 | 74.6 | 62.3 | 50.5 | 38.5 | 26.6 | 14.9 |

**Table 6.14: Conventional radix-3 FFT SQNR performance using complex multiplier in dB.**

| N | Complex Multipliers | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| | 20-bit | 18-bit | 16-bit | 14-bit | 12-bit | 10-bit | 8-bit | 6-bit | 4-bit |
| 2187 | 106 | 93.3 | 81.4 | 70.2 | 57.2 | 45.3 | 33 | 22.1 | 10.2 |
| 729 | 107.2 | 94.5 | 82.6 | 71.3 | 58.4 | 46.4 | 34.2 | 23.3 | 11.2 |
| 243 | 108.6 | 95.9 | 84 | 72.7 | 59.7 | 47.8 | 35.6 | 24.7 | 12.5 |
| 81 | 110.3 | 97.6 | 85.8 | 74.3 | 61.4 | 49.5 | 37.4 | 26.4 | 14.1 |
| 27 | 112.7 | 100 | 88.3 | 76.3 | 63.8 | 51.8 | 39.8 | 28.4 | 16.3 |

**Table 6.15: Conventional radix-4 FFT SQNR performance using complex multiplier in dB.**

| N | Complex Multipliers | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| | 20-bit | 18-bit | 16-bit | 14-bit | 12-bit | 10-bit | 8-bit | 6-bit | 4-bit |
| 1024 | 110.3 | 97.8 | 85.7 | 73.8 | 61.6 | 49.7 | 37.6 | 26.1 | 13.4 |
| 256 | 111.8 | 99.5 | 87.1 | 75.4 | 63.2 | 51.4 | 39.2 | 27.8 | 15 |
| 64 | 113.8 | 102 | 89 | 77.4 | 65.4 | 53.7 | 41.3 | 29.7 | 17.2 |
| 16 | 117 | 105.7 | 92.6 | 80.8 | 69.5 | 57.8 | 44.5 | 32.5 | 20.8 |

**Table 6.16: Conventional radix-8 FFT SQNR performance using complex multiplier in dB.**

| N | Complex Multipliers | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| | 20-bit | 18-bit | 16-bit | 14-bit | 12-bit | 10-bit | 8-bit | 6-bit | 4-bit |
| 512 | 116 | 103.8 | 92 | 79.5 | 67.7 | 55.8 | 43.2 | 31.2 | 18.4 |
| 64 | 118.6 | 107.2 | 94.8 | 82.4 | 70.9 | 59.1 | 46.4 | 34 | 21.7 |

## 6.2 Hardware performance comparison

If the equalization is done in the FFT architecture, we can merge the CORDIC gains in the last stage with the previous stage and synthesize them by double MSR-CORDIC with double iterations number with respect to the MSR-CORDIC using in the previous stages.

For comparison, we choose (2,1) normal MSR-CORDIC to synthesize both algorithms but with 4 iterations for conventional algorithm and 2 iterations for modified algorithm. Table 6.18 and 6.17 show hardware performance gain without and with equalization. There is SQNR gain by 14 dB in all architectures . As shown, parallel-pipeline architecture's latency decreased by 38%. Thus, pipeline registers are also decreased by 38%. MBFFT has SQNR gain but almost no hardware performance gain.

**Table 6.17: Hardware performance comparison without equalization**

| H.W. metric | Latency (clk) | | | Adders (count) | | |
|---|---|---|---|---|---|---|
| FFT Arch. | Conv. | Mod. | Improv. | Conv. | Mod. | Improv. |
| Parallel-pipeline | 42 | 26 | 38.1% | 86016 | 77856 | 9.5% |
| R2SDF | 1065 | 1049 | 1.5% | 168 | 104 | 38.1% |

**Table 6.18: Hardware performance comparison with equalization**

| H.W. metric | Latency (clk) | | | Adders (count) | | |
|---|---|---|---|---|---|---|
| FFT Arch. | Conv. | Mod. | Improv. | Conv. | Mod. | Improv. |
| Parallel-pipeline | 42 | 28 | 33.3% | 86016 | 86048 | 0% |
| R2SDF | 1065 | 1051 | 1.3% | 168 | 112 | 33.33% |

A more rigorous comparison is done between the conventional FFT using complex multipliers and the hardware-friendly algorithm using 3-iterations MVR CORDIC. Both designs use R2SDF architecture with 16 bit word length and synthesized using 130n IBM technology. The results are shown in tables 6.19, 6.20 and 6.21. The area is calculated after place and route with input frequency = 100MHz. The power is calculated before place and route with input frequency = 100MHz. Dynamic power is calculated using the default switching activity derived by Synopsis tools. Even though the static power is negligible beside the dynamic power, it is reported here in order to give an intuition of the power behavior in the technologies that static power dominates.

**Table 6.19: The total area of R2SDF comparison between conventional FFT uses complex multiplier and hardware-friendly FFT uses 3-iterations MVR CORDIC.**

| Length | Conventional | Hardware-friendly | Enhancement |
|---|---|---|---|
| 16 | 0.1025 $mm^2$ | 0.073 $mm^2$ | 29.1% |
| 32 | 0.167$mm^2$ | 0.123$mm^2$ | 26.35% |
| 64 | 0.256 $mm^2$ | 0.2$mm^2$ | 22.2% |
| 128 | 0.4 $mm^2$ | 0.33$mm^2$ | 18% |
| 256 | 0.534 $mm^2$ | 0.452 $mm^2$ | 15.43% |
| 512 | 0.71 $mm^2$ | 0.618 $mm^2$ | 12.9% |
| 1024 | 0.973 $mm^2$ | 0.873$mm^2$ | 10.3% |

**Table 6.20: The Dynamic power of R2SDF comparison between conventional FFT uses complex multiplier and hardware-friendly FFT uses 3-iterations MVR CORDIC.**

| Length | Conventional | Hardware-friendly | Enhancement |
|---|---|---|---|
| 16 | 7.4 $mW$ | 5.2 $mW$ | 29.7% |
| 32 | 11 $mW$ | 8 $mW$ | 27.3% |
| 64 | 16.7 $mW$ | 12.6 $mW$ | 24.6% |
| 128 | 23.2 $mW$ | 18.4 $mW$ | 20.7% |
| 256 | 36 $mW$ | 30.4 $mW$ | 15.6% |
| 512 | 51 $mW$ | 43.6 $mW$ | 14.5% |
| 1024 | 67.2 $mW$ | 59.3 $mW$ | 11.8% |

**Table 6.21: The Static power of R2SDF comparison between conventional FFT uses complex multiplier and hardware-friendly FFT uses 3-iterations MVR CORDIC.**

| Length | Conventional | Hardware-friendly | Enhancement |
|---|---|---|---|
| 16 | 1.7 $\mu W$ | 1.1 $\mu W$ | 35.3% |
| 32 | 2.7 $\mu W$ | 1.9 $\mu W$ | 29.6% |
| 64 | 4 $\mu W$ | 2.9 $\mu W$ | 27.5% |
| 128 | 6 $\mu W$ | 4.6 $\mu W$ | 23.3% |
| 256 | 8 $\mu W$ | 6.4 $\mu W$ | 20% |
| 512 | 10.2 $\mu W$ | 8.3 $\mu W$ | 18.6% |
| 1024 | 12.5 $\mu W$ | 10.4 $\mu W$ | 16.8% |

As seen from the table, the area gain is decreasing with the increase with the FFT length. This is because the proposed algorithm enhance the computational area not the FIFO. The FIFO increases exponentially with number of stages, however, the computational area increases linearly. Charts 6.3 and 6.4 show the area breakdown of the multiplier-based and cordic-friendly implementations. As shown, in the small lengths the datapath area is dominant; hence, the enhancement gain is large.

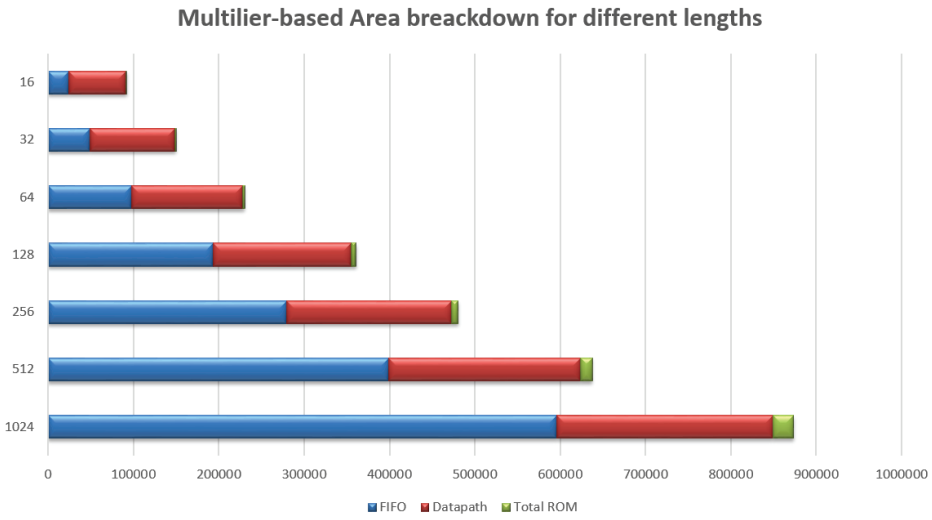


**Figure 6.3: Multiplier-based Conventional FFT Area Break down for different lengths, where area in $\mu m^2$.**

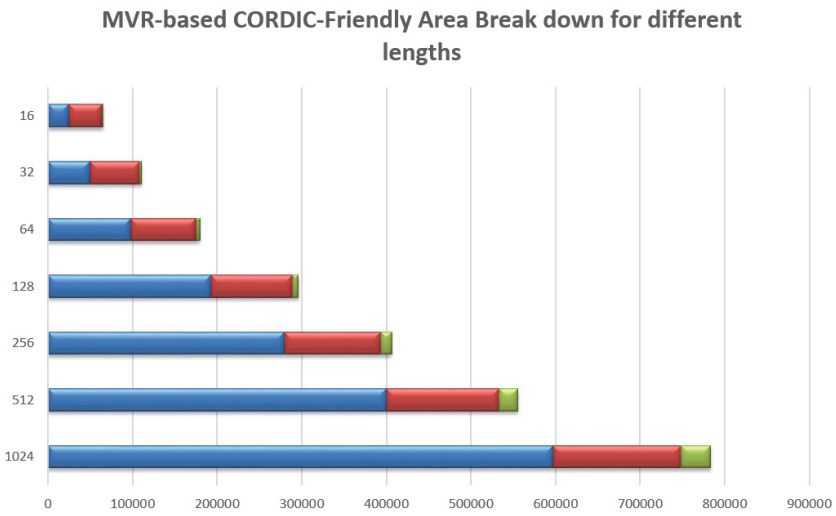


**Figure 6.4: MVR-based CORDIC-Friendly FFT Area Break down for different lengths, where area in $\mu m^2$.**

The scientific community shows interest in this problem, i.e. dominating the serial-access memory, i.e. FIFO in our case, part over the total area and power of the digital design. [30] proposes spintronic domain wall memory (DWM) based embedded memories for DSP building blocks to solve this problem. Their simulations using 65 nm technology shows promising results depicted in Fig. 6.5. The figure shows the area, power, and leakage power comparisons of R2SDF FFT processors with various FFT lengths.

Since Hardware friendly FFT enhance the computational area with almost 40% constant with the FFT length, it will enhance the total area of R2SDF by 40% using [30]'s FIFO.

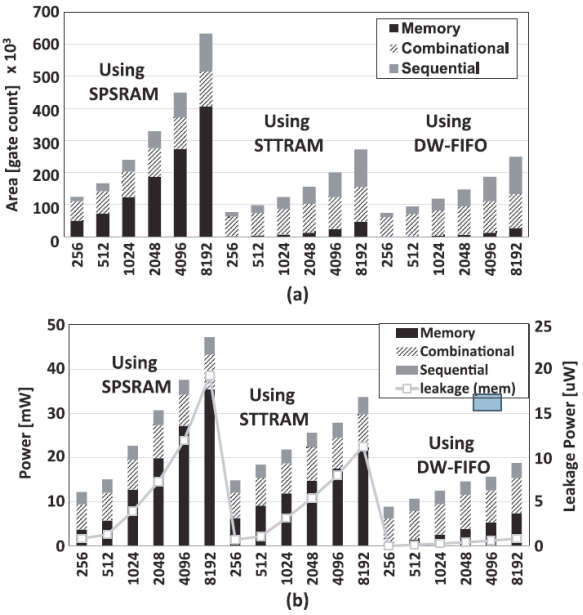


**Figure 6.5: This is the work [30]. The area, power, and leakage power comparisons of R2SDF FFT processors with various FFT lengths using DWM FIFO proposed by [30] results. (a) Area and (b) power results of 256 8192-point FFT processor using SPSRAM, STTRAM, and DW-FIFO.**

# Chapter 7

# Conclusion and Future Work

We have introduced a novel HW-friendly radix-r FFT algorithm that enables using efficient CORDIC type, even if the CORDIC type does not support equal CORDIC gain. The algorithm reorganizes the conventional FFT rotations to unify the CORDIC gain multiplied by all butterfly operands in the whole sub FFT stage. The proposed radix-2 algorithm achieves 14 dB SQNR gain, compared to conventional radix-2 FFT architecture using half the number of iterations. Moreover, for 1024 points FFT, either latency or processing area are decreased by 38% in serial-pipeline and parallel-pipeline architectures.

As extension to this work, the following points are recommended for the future work:

## 7.1 Global input-independent optimization for angles of rotations

This work has modified the radix-2 FFT algorithm to be suitable with the different CORDIC techniques. It has optimized the choosing of the CORDIC parameters to maximize SQNR of each butterfly group independent on other butterflies.

Fig. 7.1 demonstrates the optimization scheme used in this work where the butterflies with the same color have the CORDIC gain in order to propagate the CORDIC gains to the last stage. As butterfly 1 and 2 belong to the same butterflies group, their CORDIC parameters are the same and their propagation scales do not affect their successors' butterflies, namely 3 and 4. Therefore, optimization of 3 and 4 is done independent on 1 and 2.

As a future work, we suggest optimizing 1, 2, 3 and 4 jointly. Fig. 7.2 demonstrates the proposed global optimization scheme. Each butterfly can have its own CORDIC gain. Unlike the previous algorithm, butterflies 1,2,3 and 4 can have different CORDIC gains. Therefore, optimization of 3 and 4 is done jointly with the 1 and 2 in order to enhance the overall SQNR with lower number of CORDIC iterations. For example, if the error introduced by 3 and 4 is dominant compared to 1 and 2, then the angles CORDIC iterations

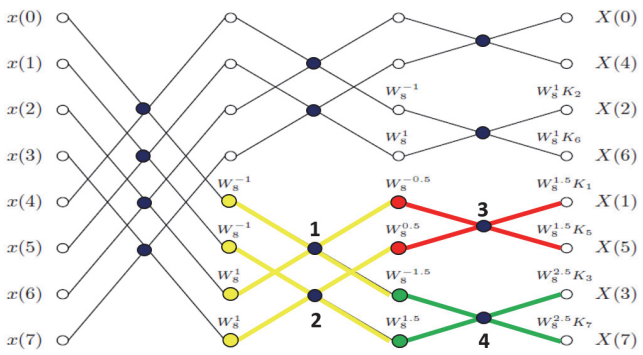


**Figure 7.1: Optimization scheme of the HW-Friendly FFT**

in 1 and 2 should be selected to enhance the errors at 3 and 4 to decrease the overall error. Hence the overall SQNR of the FFT output points can be enhanced.

## 7.2 SNR Adaptive FFT Architecture

In communication systems, receiver outputs have two sources of noise: front end thermal noise and quantization noise. The final SNR achieved by the receiver is represented as, $SNR_{out} = S/(N_{RF} + N_Q)$ , where S is the input signal power, $N_{RF}$ is the noise contribution of RF part and $N_Q$ is the quantization noise contribution of the digital part. Fig. 7.3 shows typical Bit Error Rate (BER) curves versus $SIR = S/I$ , where SIR stands for Signal to Interference Ratio. Fig. 7.3 shows that as the quantization noise of the engine decreases, a lower BER can be achieved at higher SIR; else, it will hit a fixed BER floor. Hence, although increasing the number of bits per word decreases the quantization noise, it increases the receiver power, decreases its speed and increases its area.

Since this work increases the flexability of the FFT structure, we suggest as a future work to adaptively change the number of CORDIC iterations, which will change the quantization noise, in the FFT operation depending on the received SINR (signal to interference plus noise ratio). For example, assuming unity received power, if the system operates in a good SINR environment, e.g. SINR=35 dB, the quantization noise reach almost 45 dBs without affecting the BER, thus decreasing the receiver power consumption, as shown in Fig. 7.3 point C. Moreover, If the receiver is in a low SINR environment, e.g. SINR=5 dB, the quantization noise can reach around 15 dBs, without affecting the BER, thus saving more power, as shown in Fig. 7.3 point A.

This idea falls within a larger framework of introducing hardware architectures that allow working under different SNRs in an efficient way. The main idea is to introduce quantization noise that is just a little less than the thermal noise. Increasing the tolerable quantization noise means using arithmetic and logic units that use less word length. This means using less power during the receiver operation.

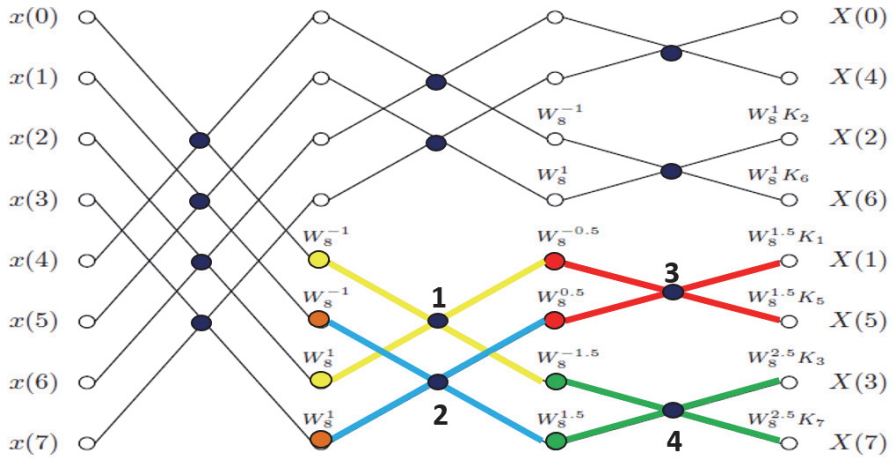


**Figure 7.2: Optimization scheme of the HW-Friendly FFT**

One way we suggest is illustrated in Fig. 7.4. Instead of following the conservative approach and doing high SQNR FFT all the time to accommodate highest SINR conditions, the FFT is done in two stages. A low resolution version of the FFT is always active, and the outputs are generated assuming a low input SINR. If the SINR measurements are high, a residual FFT Block is activated, and the outputs of this block will be added to the output of the low resolution block to get the high SQNR output. Thus, when the receiver works in a low SQNR environment, it can easily power off the residual FFT block, else it will be activated. This will decrease the average processing power at the receiver.

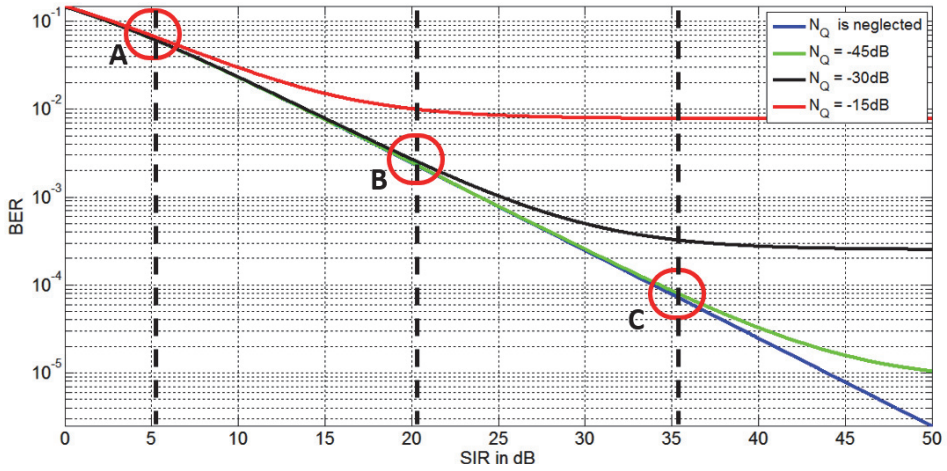


**Figure 7.3: Bit error rate under Rayleigh fading channel verses interference power for different receiver quantization noise.**

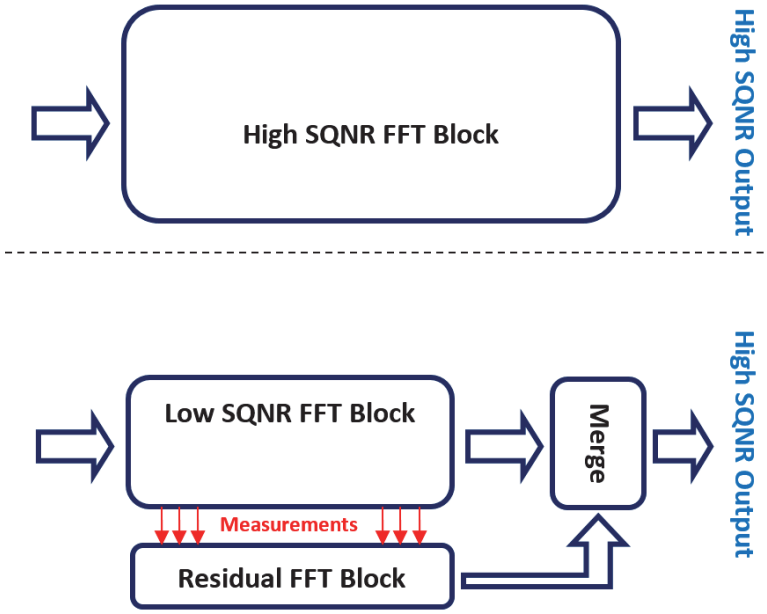


**Figure 7.4: The residual FFT architecture. Top block diagram is normal high SQNR block, lower one is residual architecture.**

# References


[1] J. W. Cooley and J. W. Tukey, "An algorithm for the machine calculation of complex fourier series," *Math. comput*, vol. 19, no. 90, pp. 297–301, 1965.

[2] S. Weinstein, "The history of orthogonal frequency-division multiplexing [history of communications]," *Communications Magazine, IEEE*, vol. 47, no. 11, pp. 26–35, Nov. 2009.

[3] H. Hassanieh, L. Shi, O. Abari, E. Hamed, and D. Katabi, "Ghz-wide sensing and decoding using the sparse fourier transform," in *INFOCOM, 2014 Proceedings IEEE*, IEEE, 2014, pp. 2256–2264.

[4] L. Shi, H. Hassanieh, A. Davis, D. Katabi, and F. Durand, "Light field reconstruction using sparsity in the continuous fourier domain," *ACM Transactions on Graphics (TOG)*, vol. 34, no. 1, p. 12, 2014.

[5] L. Shi, O. Andronesi, H. Hassanieh, B. Ghazi, D. Katabi, and E. Adalsteinsson, "Mrs sparse-fft: Reducing acquisition time and artifacts for in vivo 2d correlation spectroscopy," in *ISMRMï¿½13, Int. Society for Magnetic Resonance in Medicine Annual Meeting and Exhibition*, 2013.

[6] J. S. Walther, "A unified algorithm for elementary functions," *In Proceedings of the May 18-20, 1971, Spring Joint Computer Conference*, pp. 379–385, 1971.

[7] P. K. Meher and S. Y. Park, "Cordic designs for fixed angle of rotation," *IEEE Transactions on VLSI Systems*, vol. 21, no. 2, pp. 217–228, Feb. 2013.

[8] C.-S. Wu and A.-Y. Wu, "Modified vector rotational cordic (mvr-cordic) algorithm and architecture," *IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing*, vol. 48, no. 6, Jun. 2001.

[9] C.-S. Wu, A.-Y. Wu, and C.-H. Lin, "A high-performance/low-latency vector rotational cordic architecture based on extended elementary angle set and trellis-based searching schemes," *IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing*, vol. 50, no. 6, pp. 589–601, Sep. 2003.

[10] C.-H. Lin and A.-Y. Wu, "Mixed-scaling-rotation cordic (msr-cordic) algorithm and architecture for high-performance vector rotational dsp applications," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 52, no. 11, pp. 2385–2396, Nov. 2005.

[11] K. Kalyani, D. Sellathambi, and S. Rajaram, "Reconfigurable fft using cordic based architecture for mimo-ofdm receivers," *International Conference on Information Communication and Embedded Systems (ICICES)*, pp. 670–675, 2013.

[12] A. Malashri and C. Paramasivam, "Low power and memory efficient fft architecture using modified cordic algorithm," *International Conference on Information Communication and Embedded Systems (ICICES)*, pp. 1041–1046, 2013.

[13] S.-N. Tang, C.-H. Liao, and T.-Y. Chang, "An area- and energy-efficient multimode fft processor for wpan/wlan/wman systems," *Solid-State Circuits, IEEE Journal of*, vol. 47, no. 6, pp. 1419–1435, 2012.

[14] S. Jiangyi, T. Yinghui, W. Mingxing, and Y. Zhe, "A novel design of 1024-point pipelined fft processor based on cordic algorithm," *International Conference on Intelligent Systems Design and Engineering Application*, pp. 80–83, 2012.

[15] K. Maharatna, A. S. Dhar, and S. Banerjee, "A vlsi array architecture for realization of dft, dht, dct and dst," *Journal of Signal Processing*, vol. 81, pp. 1813–1822, 2001.

[16] A. S. Dhar and S. Banerjee, "An array architecture for fast computation of discrete hartley transform," *IEEE Transactions on Circuits and Systems*, vol. 38, no. 9, pp. 1095–1098, Sep. 1991.

[17] K. Maharatna, S. Banerjee, E. Grass, M. Krstic, and A. Troya, "Modified virtually scaling-free adaptive cordic rotator algorithm and architecture," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 15, no. 11, pp. 1463–1474, Nov. 2005.

[18] F. J. Jaime, M. A. S?nchez, J. Hormigo, J. Villalba, and E. L. Zapata, "Enhanced scaling-free cordic," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 57, no. 7, pp. 1654–1662, Jul. 2010.

[19] A. Despain, "Fourier transform computers using cordic iterations," *Computers, IEEE Transactions on*, vol. C-23, no. 10, pp. 993–1001, Oct. 1974.

[20] S. Y. Park and Y. J. Yu, "Fixed-point analysis and parameter selections of msr-cordic with applications to fft designs," *IEEE Transactions on Signal Processing*, vol. 60, no. 12, pp. 6245–6256, Dec. 2012.

[21] A. Cortés, I. Vélez, and J. F. Sevillano, "Radix $r^k$ FFTs: Matricial Representation and SDC/SDFPipeline Implementation," *Signal Processing, IEEE Transactions on*, vol. 57, no. 7, pp. 2824–2839, 2009.

[22] E. H. Wold and A. M. Despain, "Pipeline and parallel-pipeline fft processors for vlsi implementations," *IEEE Transactions on Computers*, vol. c-33, no. 5, pp. 414–426, May 1984.

[23] S. He and M. Torkelson, "A new approach to pipeline fft processor," in *IPPS*, 1996, pp. 766–770.

[24] C. Yu, M.-H. Yen, P.-A. Hsiung, and S.-J. Chen, "A low-power 64-point pipeline fft/ifft processor for ofdm applications," *Consumer Electronics, IEEE Transactions on*, vol. 57, no. 1, pp. 40–40, Feb. 2011.

[25] S. Yoshizawa, A. Orikasa, and Y. Miyanaga, "An area and power efficient pipeline fft processor for 8 by 8 mimo-ofdm systems," in *Circuits and Systems (ISCAS), 2011 IEEE International Symposium on*, May 2011, pp. 2705–2708.

[26] J. E. Volder, "The birth of cordic," *Journal of VLSI signal processing systems for signal, image and video technology*, vol. 25, no. 2, pp. 101–105, 2000.

[27] J. E. Volder, "The cordic trigonometric computing technique," *IRE Transactions on Electronic Computers*, vol. EC-8, pp. 330–334, Sep. 1959.

[28] J. H. Friedman, J. L. Bentley, and R. A. Finkel, "An Algorithm for Finding Best Matches in Logarithmic Expected Time," *ACM Transactions on Mathematical Software (TOMS)*, vol. 3, pp. 209–226, 1977.

[29] J. Bentley and T. Ottmann, "Algorithms for reporting and counting geometric intersections," *Computers, IEEE Transactions on*, vol. C-28, no. 9, pp. 643–647, Sep. 1979.

[30] J. Chung, K. Ramclam, J. Park, and S. Ghosh, "Exploiting serial access and asymmetric read/write of domain wall memory for area and energy-efficient digital signal processor design," *Circuits and Systems I: Regular Papers, IEEE Transactions on*, vol. PP, no. 99, pp. 1–12, 2015.

# Appendix A

# synopsys design compiler

## A.1 Introduction

Synopsys design compiler is a hardware synthesis tool. A hardware synthesis tool takes an RTL hardware description and a standard cell library, technology files, as input and produces a gate-level netlist as output. The resulting gate-level netlist is a completely structural description with standard cells only.

Internally, a synthesis tool performs many steps including high-level RTL optimizations, RTL to unoptimized boolean logic, technology independent optimizations, and finally technology mapping to the available standard cells. A synthesis tool does not perform place-and-route process, instead another programs is used, e.g. Cadence SoC Encounter.

The operating system we are using is Centos 6.

## A.2 Setting up the Directory

First, navigate using the terminal to the project directory you specify. You can do this by typing "cd <the project directory>". Then, run code A.1 in the terminal to set up the working directory. "Mkdir" command creates new folder(s), if they do not already exist. The mean of each folder is listed below:

- SRC: holds HDL files.
- DDC: holds Design compiler's database.
- work: holds the intermediate files from the design compiler.
- LOG: holds the general log reports for any errors or warnings.
- SYN: Synthesis subdirectory.

- SYN/NetList: holds the mapped HDL netlists.
- SYN/SDC: holds the timing constraints files for Place-and-Route tool.
- SYN/SDF: holds timing files for RTL simulation.
- SYN/RPT: holds the different reports of the mapped area, power, ... etc.
- SYN/LOG: holds the log reports generated from the synthesis process.

Code A.1: Directory structure

```
1  mkdir SRC
2  mkdir DDC
3  mkdir work
4  mkdir SYN
5  mkdir LOG
6  mkdir SYN/NetList
7  mkdir SYN/SDC
8  mkdir SYN/SDF
9  mkdir SYN/RPT
10 mkdir SYN/LOG
```

## A.3 Setup file

Then in the project directory, add code A.2 under a file named as ".synopsys_dc.setup". Notice that since the file name start with dot, the default situation that this file will be hidden. Also, you must stick to the proposed file name to avoid any errors.

The ".synopsys_dc.setup" file is the setup file for Synopsys Design Compiler. Setup file is used for initializing design parameters and variables, declare design libraries, and so on. The commands in this file are executed when Design Compiler is invoked.

In the setup file, lines started with hash symbol "#" are comments. The mean of each line is listed below:

- Line 2: append the technology files location to the search path of Synopsys Design Compiler. Without this line, Synopsys Design Compiler will not be able to find the technology files that it will synthesize against them.
- Line 5: choose the target technology library. The target technology library is the library that Design Compiler uses to build the circuit. That is, during technology mapping phase Design Compiler selects components from the library specified with the target library to build the gate-level netlist. In this example, we used standard

cells belong to 130n IBM with typical-typical corner which occurs at VDD = 1.2 V and temperature = 25°$C$. Also, we used RAM black box generated from a RAM generator associated with 130n IBM technology files.

- Line 8: refers to the symbol library that defines the schematic symbols for components in technology library. These symbols are needed for drawing design schematics.
- Line 11: refers to the link library which is used to resolve design references. That is, Design Compiler must connect all the library components to their references. This step is called linking the design or resolving references. Notice that in most cases the link library is the same as the target library, and hence we recalled the target library variable as "$target_library".
- Line 14: maps a design library to the work directory. Design Compiler uses this directory to store intermediate representations of the design generated during the synthesis process.
- Lines 17 to end: set some variables such as the designer and his company names.

Code A.2: Setup file

```
1  #Search Path:the location for technology files
2  append search_path " /root/Desktop/DC/dc/DC_files"
3
4  #Target Library
5  set target_library "scx3_cmos8rf_rvt_tt_1p2v_25c.db RAM.db"
6
7  #Symbol Library
8  set symbol_library [list ibm13rfrvt.sdb]
9
10 #Link Library --> must changed when compile with ram
11 set link_library "* $target_library"
12
13 #Work Library
14 define_design_lib work -path ./work
15
16 #Personal infos
17 set designer "Mohammed Motaz"
18 set company "Cairo_University"
19
20 #Define path directories for file locations
21 set source_path "./SRC"
22 set log_path "./LOG"
23 set ddc_path "./DDC"
```

```
24 set netlist_path "./SYN/NetList"
```

## A.4 Getting started with Synopsys design compiler

In order to lunch Synopsys design compiler command shell, type in the terminal "dc_shell"; make sure to start the terminal in the project directory. This is the mode we are going to use, however, if for any reason the GUI is needed, type "design_vision". After that, run any TCL file by typing "source filename.tcl".

The first TCL file "Initiazation.tcl" contains code A.3. The mean of some lines are elaborated below:

- Line 1: sets the PROJECT_DIR variable to the project directory you started "dc_shell" in.
- Line 7: sets all HDL entities without the top module.
- Line 8: sets the HDL top entity.
- Line 9: sets the HDL top architecture.
- Line 15 to 26: set different parameters about the clock and the reset signal such as the clock name and period and max/min input/output delays of the system. These parameters are important for the design synthesizing. You may refer to Synopsys design compiler manual for more information.
- Line 30 to 32: set the technology library name and the wire model name.

Code A.3: Initiazation file

```
1  set PROJECT_DIR [pwd]
2
3  #-----------------------------------------
4  # Design related information (can be changed)
5  #-----------------------------------------
6  ## VHDL File
7  set VHDL_SUB_ENTITY {counter FIFO Mul_ROM_Stage
8  Mul_Sel_ROM nreg_without_load Package2
9  Pipeline_Butterfly_with_Mul_engine
10 Pipeline_FFT_Stage_Mul RAM_2P Truncator}
11 set VHDL_TOP_ENTITY {R2SDF_Mul_v2}
12 set VHDL_TOP_ARCH structural_R2SDF_Mul_v2
13
14 ## Clock
```

```
set CLK_NAME CLK
set RST_NAME RST
set CLK_PERIOD 10
set INPUT_DELAY_MAX [expr 0.3*$CLK_PERIOD]
set INPUT_DELAY_MIN 1
set OUTPUT_DELAY_MAX [expr 0.3*$CLK_PERIOD]
set OUTPUT_DELAY_MIN 1
set CLK_UNCERTNTY_SETUP 0
set CLK_LATNCY 0
set CLK_TRAN 0
set OUTPUT_PORT_LOAD 0.05
set MAX_INPUT_CAP 0.05

## Libraries & operating points
## Operation_point
set TYPICAL_LIB scx3_cmos8rf_rvt_tt_1p2v_25c.db
set Operation scx3_cmos8rf_rvt_tt_1p2v_25c
set USED_WIRE_LOAD_MODEL ibm13_wl10


## Flags
# Flags that drive the script behavior
#(can be changed)
#
# DB_FORMAT (db | ddc)
# if db, use the old DB format to
#store design information
# if ddc, use the new XG format to
#store design information(recommended)
# SHARE_RESOURCES (0 | 1)
# if 1, force the tool to share
#resources as much as possible
# if 0, no resource sharing
# COMPILE_SIMPLE (0 | 1)
# if 1, only do a single compile
# with default arguments
# if 0, do a two-step compilation
#with ungrouping in between
# OPT (string)
# can be used to have different
#mapped file names
#-----------------
```

```
57
58 set DB_MODE ddc
59 set SHARE_RESOURCES 1
60 set COMPILE_SIMPLE 0
61 set OPT "_final" ;
62
63
64 #--------------
65 # File names
66 #--------------
67 set TOP_SOURCE_FILE_NAME ${VHDL_TOP_ENTITY}
68 set SUB_SOURCE_FILE_NAME ${VHDL_SUB_ENTITY}
69 set VHDL_TOP_SOURCE_FILE_NAME
70 ${TOP_SOURCE_FILE_NAME}.vhd
71 set VHDL_SUB_SOURCE_FILE_NAME {}
72 foreach FILE $SUB_SOURCE_FILE_NAME {
73         lappend VHDL_SUB_SOURCE_FILE_NAME
74         ${FILE}.vhd
75 }
76
77 set ROOT_FILE_NAME
78 ${VHDL_TOP_ENTITY}_${VHDL_TOP_ARCH}
79 set ELAB_FILE_NAME
80 ${ROOT_FILE_NAME}_elab
81 set MAPPED_FILE_NAME
82 ${ROOT_FILE_NAME}${OPT}_mapped
83 set DB_ELAB_FILE_NAME
84 ${ELAB_FILE_NAME}.$DB_MODE
85 set DB_MAPPED_FILE_NAME
86 ${MAPPED_FILE_NAME}.$DB_MODE
87 set VHDL_NETLIST_FILE_NAME
88 ${MAPPED_FILE_NAME}.vhd
89 set VLOG_NETLIST_FILE_NAME
90 ${MAPPED_FILE_NAME}.v
91 set SDF_FILE_NAME
92 ${MAPPED_FILE_NAME}.sdf
93 set SDC_FILE_NAME
94 ${MAPPED_FILE_NAME}.sdc
95 set RPT_AREA_FILE_NAME
96 ${MAPPED_FILE_NAME}_area.rpt
97 set RPT_MAX_TIMING_FILE_NAME
98 ${MAPPED_FILE_NAME}_timing_max.rpt
```

```
set RPT_MIN_TIMING_FILE_NAME
${MAPPED_FILE_NAME}_timing_min.rpt
set RPT_RESOURCES_FILE_NAME
${MAPPED_FILE_NAME}_resources.rpt
set RPT_REFERENCES_FILE_NAME
${MAPPED_FILE_NAME}_references.rpt
set RPT_CELLS_FILE_NAME
${MAPPED_FILE_NAME}_cells.rpt
set RPT_CONSTRAINTS_FILE_NAME
${MAPPED_FILE_NAME}_constraints.rpt
set RPT_POWER_FILE_NAME
${MAPPED_FILE_NAME}_power.rpt
#------------------------------
# Absolute paths
#------------------------------
set VHDL_TOP_SOURCE_FILE
${PROJECT_DIR}/SRC/${VHDL_TOP_SOURCE_FILE_NAME}
set VHDL_SUB_SOURCE_FILE {}
foreach FILE_NAME $VHDL_SUB_SOURCE_FILE_NAME {
        lappend VHDL_SUB_SOURCE_FILE
        ${PROJECT_DIR}/SRC/${FILE_NAME}
}

set VHDL_NETLIST_FILE
${PROJECT_DIR}/SYN/NetList/${VHDL_NETLIST_FILE_NAME}
set VLOG_NETLIST_FILE
${PROJECT_DIR}/SYN/NetList/${VLOG_NETLIST_FILE_NAME}
set DB_ELAB_FILE
${PROJECT_DIR}/DDC/${DB_ELAB_FILE_NAME}
set DB_MAPPED_FILE
${PROJECT_DIR}/DDC/${DB_MAPPED_FILE_NAME}
set SDF_FILE ${PROJECT_DIR}/SYN/SDF/${SDF_FILE_NAME}
set SDC_FILE ${PROJECT_DIR}/SYN/SDC/${SDC_FILE_NAME}
set RPT_AREA_FILE
${PROJECT_DIR}/SYN/RPT/${RPT_AREA_FILE_NAME}
set RPT_MAX_TIMING_FILE
${PROJECT_DIR}/SYN/RPT/${RPT_MAX_TIMING_FILE_NAME}
set RPT_MIN_TIMING_FILE
${PROJECT_DIR}/SYN/RPT/${RPT_MIN_TIMING_FILE_NAME}
set RPT_RESOURCES_FILE
${PROJECT_DIR}/SYN/RPT/${RPT_RESOURCES_FILE_NAME}
set RPT_REFERENCES_FILE
```

```
141 ${PROJECT_DIR}/SYN/RPT/${RPT_REFERENCES_FILE_NAME}
142 set RPT_CELLS_FILE
143 ${PROJECT_DIR}/SYN/RPT/${RPT_CELLS_FILE_NAME}
144 set RPT_CONSTRAINTS_FILE
145 ${PROJECT_DIR}/SYN/RPT/${RPT_CONSTRAINTS_FILE_NAME}
146 set RPT_POWER_FILE
147 ${PROJECT_DIR}/SYN/RPT/${RPT_POWER_FILE_NAME}
```

After sourcing “Initiazation.tcl”, type “source synthesis.tcl”. “synthesis.tcl” contains code A.4. The mean of some lines are elaborated below:

- Line 9: analyzes HDL files and checks for the syntax error.
- Line 13: elaborates the design and passes its parameters.
- Line 64: synthesizes the design.
- Line 67: checks that the design process passed without errors
- Line 68: link the design.
- Line 77 to 93: generates the mapped area, power and timing reports. All the reports are located in “./SYN/RPT” directory.

Code A.4: Synthesis file

```
1  ## Execution
2  #----------------------
3  # Analyze RTL source
4  #----------------------
5  set FILES $VHDL_TOP_SOURCE_FILE
6  foreach FILE_NAME $VHDL_SUB_SOURCE_FILE {
7          lappend FILES ${FILE_NAME}
8  }
9  analyze -format vhdl -lib work $FILES
10 #-----------------------
11 # Elaborate design
12 #-----------------------
13 set hdlin_optimize_pla_max_branch 1024
14 elaborate $VHDL_TOP_ENTITY -architecture
15 $VHDL_TOP_ARCH -library work -parameters
16 "N = 16, Cos_Sin_width = 22, Pipeline_Stages = 4,
17 FFT_Length = 16, FFT_Stages = 4" -update
18
```

```
19 link > ./SYN/LOG/link_elab.log
20
21 #-------------------------
22 # Define constraints
23 #-------------------------
24 ##Clock
25 create_clock -name $CLK_NAME
26 -period $CLK_PERIOD [get_ports $CLK_NAME]
27 set_clock_latency $CLK_LATNCY [get_clocks $CLK_NAME]
28 set_clock_uncertainty
29 -setup $CLK_UNCERTNTY_SETUP [get_clocks $CLK_NAME]
30 set_clock_transition $CLK_TRAN [get_clocks $CLK_NAME]
31 set_dont_touch_network [list $CLK_NAME]
32 set_drive 0 [list $CLK_NAME]
33 set_input_delay $INPUT_DELAY_MAX
34 -max -clock $CLK_NAME [all_inputs]
35 set_input_delay $INPUT_DELAY_MIN
36 -min -clock $CLK_NAME [all_inputs]
37 remove_input_delay [get_clocks $CLK_NAME]
38 set_output_delay $OUTPUT_DELAY_MAX
39 -max -clock $CLK_NAME [all_outputs]
40 set_output_delay $OUTPUT_DELAY_MIN
41 -min -clock $CLK_NAME [all_outputs]
42 set_dont_touch_network [list $RST_NAME]
43 set_drive 0 [list $RST_NAME]
44
45 ##Load & optmization
46 set_max_capacitance $MAX_INPUT_CAP [all_inputs]
47 set_load $OUTPUT_PORT_LOAD [all_outputs]
48 remove_attribute [get_clocks $CLK_NAME] max_capacitance
49 set_max_area 0
50 set_operating_conditions $Operation
51 set_wire_load_model -name $USED_WIRE_LOAD_MODEL
52 set_wire_load_mode enclosed
53
54 #---------------------------------------
55 # Save elaborated design and constraints
56 #---------------------------------------
57 write -hierarchy -format $DB_MODE
58 -output $DB_ELAB_FILE
59 #---------------------------------------
60 # Map design to gates
```

```
61 #--------------------------------------
62
63 set compile_ultra_ungroup_small_hierarchies false
64 compile_ultra -no_autoungroup -
65 area_high_effort_script
66 ##Check_Design
67 check_design > ./SYN/LOG/check_design.log
68 link > ./SYN/LOG/link_compilation.log
69 #--------------------------------------
70 # Save mapped design
71 #--------------------------------------
72 write -hierarchy -format $DB_MODE
73 -output $DB_MAPPED_FILE
74 #--------------------------------------
75 # Generate reports
76 #--------------------------------------
77 report_area -nosplit -hierarchy > $RPT_AREA_FILE
78 report_timing -path full -delay max -transition_time
79 -nets -attributes -significant_digits 5 -nosplit
80 -nworst 3 -max_paths 4
81 -sort_by group > $RPT_MAX_TIMING_FILE
82 report_timing -path full -delay min -transition_time
83 -nets -attributes -significant_digits 5 -nosplit
84 -nworst 3 -max_paths 4
85 -sort_by group > $RPT_MIN_TIMING_FILE
86 report_resources -nosplit
87 -hierarchy > $RPT_RESOURCES_FILE
88 report_reference -nosplit > $RPT_REFERENCES_FILE
89 report_cell -nosplit > $RPT_CELLS_FILE
90
91 report_constraint -all_violators
92 -significant_digits 4 > $RPT_CONSTRAINTS_FILE
93 report_power -analysis_effort low > $RPT_POWER_FILE
94 #--------------------------------------
95 # Generate VHDL netlist
96 #--------------------------------------
97 change_names -rule vhdl -hierarchy -verbose;
98 write -format vhdl -hierarchy
99 -output $VHDL_NETLIST_FILE;
100 #--------------------------------------
101 # Generate SDF data
102 #--------------------------------------
```

```
write_sdf -version 2.1 $SDF_FILE
#-------------------------------------
# Generate Verilog netlist
#-------------------------------------
remove_design -all
read_file -format $DB_MODE $DB_MAPPED_FILE;
change_names -rule verilog -hierarchy -verbose;
write -format verilog -hierarchy
-output $VLOG_NETLIST_FILE
#-------------------------------------
# Save system constraints
#-------------------------------------
write_sdc -nosplit $SDC_FILE
```

# ملخص الرسالة

تحويلة فوريه السريعة هي "العمليه الحتمية" في عالم معالجة الإشارات. كونها تستخدم في تطبيقات الصوت والصوره، أو محللات الطيف والشبكة، أو أنظمة الرادار والإتصالات، أو حتى في علم الفلك وميكانيكا الكم جعلها تكتسب هذا اللقب بلا منازع. ومن ثم، فإن تحسين محاولات فوريه السريعة من حيث المساحة، أو الوقت، أو الطاقة اللازمين له عظيم الأثر في شتى المجالات.

الهيكل القائم على شكل الفراشة في خوارزمية تحويلة فوريه السريعة هو السبب الرئيسي المسؤول عن تقليل العمليات الحسابية اللازمة لتنفيذ التحويلة. ومع ذلك، فإن الأنظمة الحديثه لا تزال متعطشة لمزيد من الخفض في العمليات الحسابية. من بين هذه العمليات، الضرب بمعاملات التحويل هي الأعلى تكلفة. من وجهة النظر التنفيذية، عمليات الضرب هذه يمكن صنعها إما باستخدام مضاعفات الأعداد العقدية أو حاسبات رقمية لدوران الإحداثيات.

هذا العمل يقدم "تحويلة فوريه السريعة صديقة المكونات المادية": وهي إعادة هيكلة لفراشة تحويلة فوريه السريعة من أجل تحقيق حاصل ضرب أقل للمساحة مع الوقت مع الطاقة مقارنة بالخوارزمية التقليدية. علاوة على ذلك، "تحويلة فوريه السريعة صديقة المكونات المادية" تسمح بإستخدام أنواع من الحاسبات الرقمية لدوران الإحداثيات التي تنتج كسب غير متكافئ من دون الحاجة إلى أي معادله بعدهم.

في دراسة حالة، تم استخدام عمارة خط انابيب المسار المنفرد الراجع. لنفس موارد المكونات المادية، فإن "تحويلة فوريه السريعة صديقة المكونات المادية" تحقق زيادة كبيرة في نسبة الإشارة إلى ضوضاء التكميم. الخوارزمية المقترحة توفر ما يصل إلى ٧٥ ديسيبل زيادة في نسبة الإشارة إلى ضوضاء التكميم مقارنة بتحويلة فوريه السريعة التقليدية عند استخدام أحجام مختلفه من تحولات فوريه السريعة وأنواع مختلفه من الحاسبات الرقمية لدوران الإحداثيات وأحجام مختلفة من مضاعفات الأعداد العقدية. في دراسة الحالة نفسها، إذا كان المطلوب الحفاظ على مستوى معين من نسبة الإشارة إلى ضوضاء التكميم فإن "تحويلة فوريه السريعة صديقة المكونات المادية" تحقق تقليل في المساحة، يصل إلى ٤٠٪، مقارنة بتحويلة فوريه السريعة التقليدية.

**مهندس:** محمد أحمد المعتزبالله السيد
**تاريخ الميلاد:** ١٩٩٠/٤/٢٨
**الجنسية:** مصري
**تاريخ التسجيل:** ٢٠١٢/١٠/١
**تاريخ المنح:** / /٢٠١٦
**الدرجة:** ماجستير العلوم
**القسم:** هندسة الإلكترونيات والإتصالات الكهربية

**المشرفون:**

أ. د. حسام على فهمي
أ. م. د. عمر أحمد نصر

**الممتحنون:**

| | |
|---|---|
| أ. د. حسام على فهمي | (المشرف الرئيسي) |
| أ. م. د. عمر أحمد نصر | (عضو) |
| أ. د. محمد فتحي أبو اليزيد | (الممتحن الداخلي) |
| أ. د. هاني فكري رجائي | (الممتحن الخارجي) |

**عنوان الرسالة:**

تسريع تحويلة فوريير السريعة

**الكلمات الدالة:**
تحويلة فورييه السريعة، الحاسبات الرقمية لدوران الإحداثيات، نسبة الإشارة إلى ضوضاء التكميم، عمارة خط انابيب المسار المنفرد الراجع

**ملخص الرسالة:**
هذا العمل يقدم "تحويلة فورييه السريعة صديقة المكونات المادية": وهي إعادة هيكلة لفراشة تحويلة فورييه السريعة من أجل تحقيق حاصل ضرب أقل للمساحة مع الوقت مع الطاقة مقارنة بالخوارزمية التقليدية. علاوة على ذلك، "تحويلة فورييه السريعة صديقة المكونات المادية" تسمح بإستخدام أنواع من الحاسبات الرقمية لدوران الإحداثيات التي تنتج كسب غير متكافئ من دون الحاجة إلى أي معادله بعدهم.
في دراسة حالة، تم استخدام عمارة خط انابيب المسار المنفرد الراجع. لنفس موارد المكونات المادية، فإن "تحويلة فورييه السريعة صديقة المكونات المادية" تحقق زيادة كبيرة في نسبة الإشارة إلى ضوضاء التكميم. الخوارزمية المقترحة توفر ما يصل إلى ٧٥ ديسيبل زيادة في نسبة الإشارة إلى ضوضاء التكميم مقارنة بتحويلة فورييه السريعة التقليدية عند استخدام أحجام مختلفه من تحولات فورييه السريعة وأنواع مختلفه من الحاسبات الرقمية لدوران الإحداثيات وأحجام مختلفة من مضاعفات الأعداد العقدية.
في دراسة الحالة نفسها، إذا كان المطلوب الحفاظ على مستوى معين من نسبة الإشارة إلى ضوضاء التكميم فإن "تحويلة فورييه السريعة صديقة المكونات المادية" تحقق تقليل في المساحة، يصل إلى ٤٠٪، مقارنة بتحويلة فورييه السريعة التقليدية.

# تسريع تحويلة فوريير السريعة


اعداد

محمد أحمد المعتزبالله السيد


رسالة مقدمة الي
كلية الهندسة – جامعة القاهرة
كجزء من متطلبات الحصول علي درجة
ماجستير العلوم
في
هندسة الإلكترونيات والإتصالات الكهربية

يعتمد من لجنة الممتحنين:

---

أ. د. حسام على فهمي – المشرف الرئيسي

---

أ. م. د. عمر أحمد نصر – عضو

---

أ. د. محمد فتحي أبو اليزيد – الممتحن الداخلي

---

أ. د. هاني فكري رجائي – الممتحن الخارجي
قسم الإلكترونيات والإتصالات الكهربية – كلية الهندسة – جامعة عين شمس


كلية الهندسة – جامعة القاهرة
الجيزة – جمهورية مصر العربية
٢٠١٦

# تسريع تحويلة فوريير السريعة


اعداد

محمد أحمد المعتزبالله السيد


رسالة مقدمة الي
كلية الهندسة – جامعة القاهرة
كجزء من متطلبات الحصول علي درجة
ماجستير العلوم
في
هندسة الإلكترونيات والإتصالات الكهربية

تحت إشراف


أ. د. حسام على فهمي
أستاذ دكتور
قسم هندسة الإلكترونيات والإتصالات الكهربية
كلية الهندسة – جامعة القاهرة

أ. م. د. عمر أحمد نصر
أستاذ مساعد
قسم هندسة الإلكترونيات والإتصالات الكهربية
كلية الهندسة – جامعة القاهرة



كلية الهندسة – جامعة القاهرة
الجيزة – جمهورية مصر العربية
٢٠١٦



# تسريع تحويلة فوريير السريعة

اعداد

محمد أحمد المعتزبالله السيد

رسالة مقدمة الي
كلية الهندسة – جامعة القاهرة
كجزء من متطلبات الحصول علي درجة
ماجستير العلوم
في
هندسة الإلكترونيات والإتصالات الكهربية

كلية الهندسة – جامعة القاهرة
الجيزة – جمهورية مصر العربية
٢٠١٦